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:

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

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.oasis-open.org/committees/sca-j/.

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

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

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

Notices

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

Deleted: 2009

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: ",
DisplayText cannot span mo
are

Table of Contents

1 Introduction ... 7
1.1 Terminology ... 7
1.2 Normative References ... 7
1.3 Non-Normative References ... 8
2 Implementation Metadata ... 9
2.1 Service Metadata ... 9
2.1.1 @Service ... 9
2.1.2 Java Semantics of a Remotable Service ... 9
2.1.3 Java Semantics of a Local Service ... 9
2.1.4 @Reference ... 10
2.1.5 @Property ... 10
2.2 Implementation Scopes: @Scope, @Init, @Destroy ... 10
2.2.1 Stateless Scope ... 10
2.2.2 Composite Scope ... 11
2.3 @AllowsPassByReference ... 11
2.3.1 Marking Services as "allows pass by reference" ... 12
2.3.2 Marking References as "allows pass by reference" ... 12
2.3.3 Applying "allows pass by reference" to Service Proxies ... 12
2.3.4 Using "allows pass by reference" to Optimize Remotable Calls ... 13
3 Interface ... 14
3.1 Java Interface Element – <interface.java> ... 14
3.2 @Remotable ... 15
3.3 @Callback ... 15
3.4 @AsyncInvocation ... 15
3.5 SCA Java Annotations for Interface Classes ... 16
3.6 Compatibility of Java Interfaces ... 16
4 SCA Component Implementation Lifecycle ... 17
4.1 Overview of SCA Component Implementation Lifecycle ... 17
4.2 SCA Component Implementation Lifecycle State Diagram ... 17
4.2.1 Constructing State ... 18
4.2.2 Injecting State ... 18
4.2.3 Initializing State ... 19
4.2.4 Running State ... 19
4.2.5 Destroying State ... 19
4.2.6 Terminated State ... 19
5 Client API ... 21
5.1 Accessing Services from an SCA Component ... 21
5.1.1 Using the Component Context API ... 21
5.2 Accessing Services from non-SCA Component Implementations ... 21
5.2.1 SCAClientFactory Interface and Related Classes ... 21
6 Error Handling ... 23
7 Asynchronous Programming ... 24
7.1 @OneWay ... 24

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

7.2 Callbacks 24
7.2.1 Using Callbacks 24
7.2.2 Callback Instance Management 26
7.2.3 Callback Injection 26
7.2.4 Implementing Multiple Bidirectional Interfaces 26
7.2.5 Accessing Callbacks 27
7.3 Asynchronous handling of Long Running Service Operations 28
7.4 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 Annotations 34
8.3 Application of Intent Annotations 34
8.3.1 Intent Annotation Examples 35
8.3.2 Inheritance and Annotation 37
8.4 Relationship of Declarative and Annotated Intents 38
8.5 Policy Set Annotations 38
8.6 Security Policy Annotations 39
8.7 Transaction Policy Annotations 40
9 Java API 42
9.1 Component Context 42
9.2 Request Context 47
9.3 ServiceReference 48
9.4 ServiceReference Interface 50
9.5 ResponseDispatch interface 51
9.6 ServiceRuntimeException 52
9.7 ServiceUnavailableException 52
9.8 InvalidServiceException 52
9.9 Constants 53
9.10 SCAClientFactory Class 53
9.11 SCAClientFactoryFinder Interface 57
9.12 SCAClientFactoryFinderImpl Class 58
9.13 NoSuchDomainException 59
9.14 NoSuchServiceException 59
10 Java Annotations 60
10.1 @AllowsPassByReference 60
10.2 @AsyncFault 61
10.3 @AsyncInvocation 62
10.4 @Authentication 62
10.5 @Authorization 63
10.6 @Callback 63
10.7 @ComponentName 65
10.8 @Confidentiality 65
10.9 @Constructor 66
10.10 @Context 67

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

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

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

1 Introduction

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

Specifically, this specification covers:

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

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

1.1 Terminology

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

1.2 Normative References

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

[ASSEMBLY] OASIS, Committee Draft 05, “SCA Assembly Model Specification Version 1.1”, January 2010. http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd05.pdf

[JAVA_CI] OASIS, Committee Draft 02, “SCA POJO Component Implementation Specification Version 1.1”, February 2010. http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd02.pdf

[SDO] SDO 2.1 Specification, http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf

[JAX-B] JAXB 2.1 Specification, http://www.jcp.org/en/jsr/detail?id=222

[WSDL] WSDL Specification, WSDL 1.1: http://www.w3.org/TR/wsdl,

[POLICY] OASIS, Committee Draft 02, “SCA Policy Framework Version 1.1”, February 2009. http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf

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

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

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

Deleted: ¶
Deleted: ,
Field Code Changed
Deleted: cd03
Field Code Changed
Deleted: cd01
Deleted: ,

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

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

49 1.3 Non-Normative References

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

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

2 Implementation Metadata

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

2.1 Service Metadata

2.1.1 @Service

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

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

2.1.2 Java Semantics of a Remotable Service

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

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

Deleted: The following snippet

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

Snippet 2-1: Remotable Java Interface

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.

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

2.1.4 @Reference

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

2.1.5 @Property

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

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

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

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

This specification defines two scopes:

- STATELESS
- COMPOSITE

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

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

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

@Destroy specifies a method called when the scope ends.

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

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

Deleted: The following snippet

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

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

Snippet 2-3: Java Component Implementation with Lifecycle Methods

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.

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

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

2.2.2 Composite Scope

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

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

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

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

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

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

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

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

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

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

2.3 @AllowsPassByReference

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

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

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

marked as “allows pass by reference” to indicate that they use input parameters, return values and exceptions in a manner that allows the SCA runtime to avoid the cost of copying mutable objects when a remotable service is called locally within the same JVM.

2.3.1 Marking Services as “allows pass by reference”

Deleted: and References

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.

Deleted: value

Deleted: below

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

2.3.2 Marking References as “allows pass by reference”

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

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

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

2.3.3 Applying “allows pass by reference” to Service Proxies

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

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

Deleted: ” (see definition below).

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

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

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

230 ### 2.3.4 Using “allows pass by reference” to Optimize Remotable Calls

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

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

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

3 Interface

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

3.1 Java Interface Element – <interface.java>

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

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

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

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

Deleted: The following

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

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

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

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
286 component concerned. [JCA30005]

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

Deleted: The following snippet

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

292 *Snippet 3-2 Example interface.java Element*

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
297 ***./services/stockquote/StockQuoteServiceCallback.class***.

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
300 @callbackInterface attribute. However, if the Java interface class identified by the @interface attribute
301 does contain a Java @Callback annotation, then the Java interface class identified by the
302 @callbackInterface attribute MUST be the same interface class. [JCA30003]

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
306 XML technologies.

307 3.2 @Remotable

308 The ***@Remotable*** annotation on a Java interface, a service implementation class, or a service reference
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
311 and exceptions are passed ***by-value***. Remotable Services are not allowed to make use of method
312 ***overloading***.

Deleted: indicates that the

313 3.3 @Callback

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
316 annotation, without any parameters, that specifies callback injection for a setter method or a field of an
317 implementation.

318 3.4 @AsyncInvocation

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

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

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

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

333 3.5 SCA Java Annotations for Interface Classes

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

Field Code Changed

336 @Intent, @Qualifier. [JCA30008]

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

339 @Intent, @Qualifier. [JCA30006]

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

Field Code Changed

342 @AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit,
343 @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service. [JCA30007]

344 3.6 Compatibility of Java Interfaces

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

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

4 SCA Component Implementation Lifecycle

This section describes the lifecycle of an SCA component implementation.

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

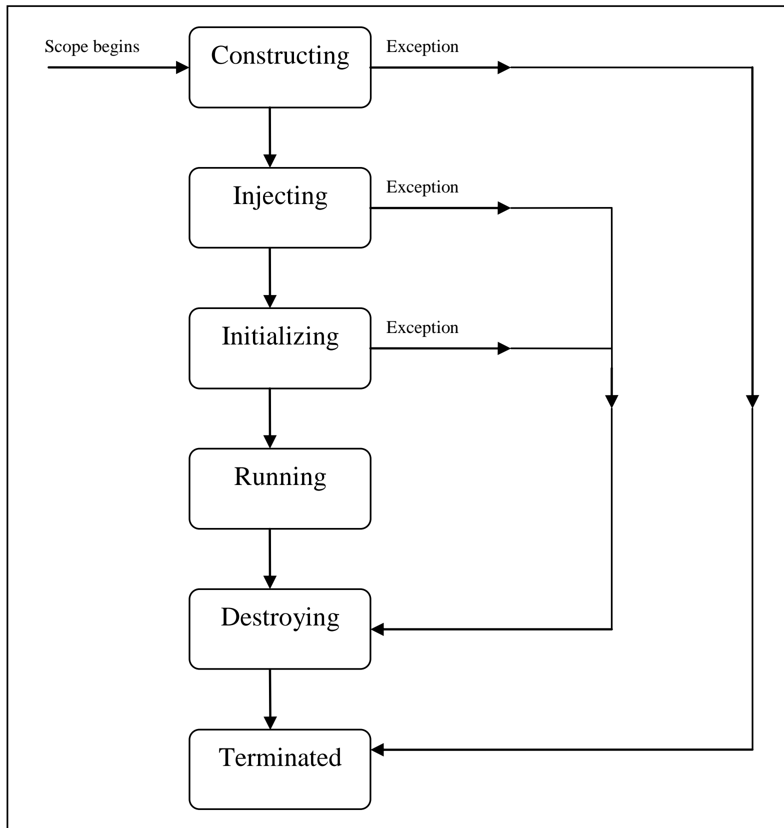
Field Code Changed

Figure 4.1 SCA - Component implementation lifecycle

Figure 4-2 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.

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



372

373

374 *Figure 4.1 SCA - Component implementation lifecycle*

375 *Figure 4-2: SCA - Component Implementation Lifecycle*

376 4.2.1 Constructing State

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

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

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

385 4.2.2 Injecting State

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

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

Copyright © OASIS® 2005, 2010. 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]
394 The SCA Runtime MUST transition the component implementation to the Initializing state. [JCA40008]
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.
400 [JCA40010] If a property or reference is unable to be injected, the SCA Runtime MUST transition the
401 component implementation to the Destroying state. [JCA40024]

402 4.2.3 Initializing State

403 When the component implementation enters the Initializing State, the SCA Runtime MUST call the
404 method annotated with @Init on the component implementation, if present. [JCA40011]
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.
408 The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component
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]
412 If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation
413 to the Running state. [JCA40014] If an exception is thrown whilst initializing, the SCA Runtime MUST
414 transition the component implementation to the Destroying state. [JCA40015]

415 4.2.4 Running State

416 When the component implementation scope ends, the SCA Runtime MUST transition the component implementation
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]

422 4.2.5 Destroying State

423 When a component implementation enters the Destroying state, the SCA Runtime MUST call the method
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
431 component implementation to the Terminated state. [JCA40020]
432 If an exception is thrown whilst destroying, the SCA Runtime MUST transition the component implementation
433 to the Terminated state. [JCA40021] The SCA Runtime MUST transition the component implementation to
434 the Terminated state. [JCA40022]

435 4.2.6 Terminated State

436 The lifecycle of the SCA Component has ended.

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

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

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

5 Client API

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

5.1 Accessing Services from an SCA Component

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

5.1.1 Using the Component Context API

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

5.2 Accessing Services from non-SCA Component Implementations

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

5.2.1 SCAClientFactory Interface and Related Classes

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

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

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

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

import java.net.URI;

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

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

   public void someMethod() {

         try {

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

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

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(HelloService.class,"HelloService");¶
String result =
helloService.hello("Hello World!");¶
```

Formatted: Bullets and Numbering

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

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

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

Snippet 5-1: Using the SCAClientFactory Interface

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

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

498
6 Error Handling

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

500 Business exceptions are thrown by the implementation of the called service method, and are defined as
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
503 component execution or problems interacting with remote services. The SCA runtime exceptions are
504 defined in the Java API section.

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

7 Asynchronous Programming

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

- support for non-blocking method calls
- callbacks

Each of these topics is discussed in the following sections.

7.1 @OneWay

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

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

Deleted: ny

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

7.2 Callbacks

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

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

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

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

7.2.1 Using Callbacks

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

Snippet 7-1 shows a scenario in which bidirectional interfaces and callbacks could be used. A client requests a quotation from a supplier. To process the enquiry and return the quotation, some suppliers might need additional information from the client. The client does not know which additional items of information will be needed by different suppliers. This interaction can be modeled as a bidirectional interface with callback requests to obtain the additional information.

Deleted: The following example

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

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      }
```

566 *Snippet 7-1: Using a Bidirectional Interface*

567

568 In Snippet 7-1, the `requestQuotation` operation requests a quotation to supply a given quantity of a
569 specified product. The QuotationCallBack interface provides a number of operations that the supplier can
570 use to obtain additional information about the client making the request. For example, some suppliers
571 might quote different prices based on the state or the ZIP code to which the order will be shipped, and
572 some suppliers might quote a lower price if the ordering company has a good credit rating. Other
573 suppliers might quote a standard price without requesting any additional information from the client.

Deleted: this example

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

Deleted: The following code snippet

576

```
577      @Callback
578      protected QuotationCallback callback;
579
580      public double requestQuotation(String productCode, int quantity) {
581          double price = getPrice(productQuote, quantity);
582          double discount = 0;
583          if (quantity > 1000 && callback.getState().equals("FL")) {
584              discount = 0.05;
585          }
586          if (quantity > 10000 && callback.getCreditRating().charAt(0) == 'A') {
587              discount += 0.05;
588          }
589          return price * (1-discount);
590      }
```

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

Deleted: ¶
The code snippet below

592

593 Snippet 7-3 is taken from the client of this example service. The client's service implementation class
594 implements the methods of the QuotationCallback interface as well as those of its own service interface
595 ClientService.

596

```
597      public class ClientImpl implements ClientService, QuotationCallback {
598
599          private QuotationService myService;
600
601          @Reference
602          public void setMyService(QuotationService service) {
603              myService = service;
604          }
```


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

605
```
    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*

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

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

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

```
@Callback
protected MyService1Callback callback1;

@Callback
protected MyService2Callback callback2;
```

Snippet 7-4: Multiple Bidirectional Interfaces in an Implementation

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

7.2.5 Accessing Callbacks

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

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

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

Deleted: The following example fragments come

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

public void someMethod() {

   MyCallback myCallback = callback.getService();    …

   myCallback.receiveResult(theResult);
}
```

Snippet 7-5: Using the Callback API

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

Alternatively, a callback can be retrieved programmatically using the ***RequestContext*** API. Snippet 7-6 shows how to retrieve a callback in a method programmatically:

Deleted: The snippet below

```
@Context
ComponentContext context;

public void someMethod() {

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

   …

   myCallback.receiveResult(theResult);
}
```

Deleted: ¶

Deleted:

Snippet 7-6: Using RequestContext to get a Callback

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

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

7.3 Asynchronous handling of Long Running Service Operations

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

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

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

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

7.4 SCA Asynchronous Service Interface

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

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

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

Snippet 7-7: Example Synchronous Java Interface Mapping

The JAX-WS client asynchronous API for the same portType adds two asynchronous forms for each synchronous method, as shown in Snippet 7-8.

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


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

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

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

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

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

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

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

The interface is annotated with the "asyncInvocation" intent.

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

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

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

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

803 For the purposes of matching interfaces (when wiring between a reference and a service, or 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>

811 – annotation with the asyncInvocation intent

812 – possible annotation with the @AsyncFault annotation

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

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

8 Policy Annotations for Java

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

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

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

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

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

8.1 General Intent Annotations

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

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

Each intent is expressed as a string. Intents are XML QNames, which consist of a Namespace URI followed by the name of the Intent. The precise form used follows the string representation used by the javax.xml.namespace.QName class, which is shown in Snippet 8-1.

Deleted: as follows:

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

Snippet 8-1: Intent Format

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

This representation is quite verbose, so we expect that reusable String constants will be defined for the namespace part of this string, as well as for each intent that is used by Java code. SCA defines constants for intents such as those in Snippet 8-2.

Deleted: the following:

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

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

Snippet 8-2: Example Intent Constants

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

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

An example of the @Requires annotation with 2 qualified intents (from the Security domain) is shown in Snippet 8-3:

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

Snippet 8-3: Multiple Intnets in One Annotation

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

Snippet 8-4 is an example of a reference requiring support for confidentiality:

```
package com.foo;

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

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

Snippet 8-4: Annotation a Reference

Users can also choose to only use constants for the namespace part of the QName, so that they can add new intents without having to define new constants. In that case, the definition of Snippet 8-4 would instead look like Snippet 8-5.

```
package com.foo;

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

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


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

```
    ...
  }
}
```

Snippet 8-5: Using Intent Constants and strings

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

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

where

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

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

8.2 Specific Intent Annotations

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

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

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

```
@Integrity
```

Snippet 8-6: Example Specific Intent Annotation

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

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

Snippet 8-7: Example Qualified Specific Intent Annotation

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

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

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

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

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

Snippet 8-8: Specific Intent Annotation Format

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

8.2.1 How to Create Specific Intent Annotations

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

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

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

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

Snippet 8-9: Defining a Specific Intent Annotation

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

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

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

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

Snippet 8-10: Multiple Qualifiers in an Annotation'

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

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

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

8.3 Application of Intent Annotations

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

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

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

SCA annotations (general or specific) are applied to the same Java element, the SCA runtime MUST NOT instantiate such an implementation class. [JCA70003]

An example of multiple policy annotations being used together is shown in Snippet 8-11:

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

Snippet 8-11: Multiple Policy Annotations

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

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. [JCA70004] This merging process does not remove or change any intents that are applied to the interface.

8.3.1 Intent Annotation Examples

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

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

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

Snippet 8-12: Merging Intents on References

Snippet 8-13 shows that mutually exclusive intents cannot be applied to the same Java element. In this example, the Java code is in error because of contradictory mutually exclusive intents "managedTransaction" and "noManagedTransaction".

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

Snippet 8-13: Mutually Exclusive Intents

Snippet 8-14 shows that intents can be applied to Java service interfaces and their methods. In this example, the effective intents for `MyService.mymethod()` are "authentication" and "confidentiality".

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

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

Field Code Changed

Deleted: follows

Deleted: Example 8.1

Deleted:

Deleted: *Example 8.1.*

Deleted: *intents*

Deleted: *references.*

Deleted: Example 8.2

Deleted:

Deleted: *Example 8.2.*

Deleted: *exclusive intents.*

Deleted: Example 8.3

Deleted: *Example 8.3.*

Deleted: *interfaces, interface methods*

Deleted: *classes.*

Deleted: Example 8.4

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

Snippet 8-15 shows that intents can be applied to Java service implementation classes. In this example, the effective intents for `MyService.mymethod()` are "authentication", "confidentiality", and "managedTransaction".

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

Snippet 8-15: Intents on Java Service Implementation Classes

Deleted: *Example 8.4.*
Deleted: *service implementation classes.*
Deleted: Example 8.5

Snippet 8-16 shows that intents can be applied to Java reference interfaces and their methods, and also to Java references. In this example, the effective intents for the method `mymethod()` of the reference `myRef` are "authentication", "integrity", and "confidentiality".

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

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

Deleted: *Example 8.5.*
Deleted: *references*
Deleted: *interfaces*
Deleted: *methods.*
Deleted: Example 8.6

Snippet 8-17 shows that intents cannot be applied to methods of Java implementation classes. In this example, the Java code is in error because of the @Authentication intent annotation on the implementation method `MyServiceImpl.mymethod()`.

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

Snippet 8-17: Intent on Implementation Method

Deleted: *Example 8.6.*
Deleted: *implementation method.*
Deleted: Example 8.7

Snippet 8-18 shows one effect of applying the SCA Policy Framework rules for merging intents within a structural hierarchy to Java service interfaces and their methods. In this example a qualified intent overrides an unqualified intent, so the effective intent for `MyService.mymethod()` is "confidentiality.message".

```
@Confidentiality("message")
public interface MyService {
```


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

```
    @Confidentiality
    public void mymethod();
}
```

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

Snippet 8-19 shows another effect of applying the SCA Policy Framework rules for merging intents within a structural hierarchy to Java service interfaces and their methods. In this example a lower-level intent causes a mutually exclusive higher-level intent to be ignored, so the effective intent for `mymethod1()` is "managedTransaction" and the effective intent for `mymethod2()` is "noManagedTransaction".

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

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

8.3.2 Inheritance and Annotation

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

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

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

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

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

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

Snippet 8-20: 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"),

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

1152 The effective intent annotation on the ***helloThere*** method of ***HelloChildService*** is @Integrity and
1153 @Authentication("transport"), the same as for this method in the ***HelloService*** class.

1154 The effective intent annotation on the ***hello*** method of ***HelloService*** is @Integrity and
1155 @Authentication("message")

1156

1157 Table 8-1 shows the equivalent declarative security interaction policy of the methods of the HelloService
1158 and HelloChildService implementations corresponding to the Java classes shown in Snippet 8-20.

Deleted: Table 8.1 below

Deleted: Example 8.9

1159

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

1160 *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.¶

1161 8.4 Relationship of Declarative and Annotated Intents

1162 Annotated intents on a Java class cannot be overridden by declarative intents in a composite document
1163 which uses the class as an implementation. This rule follows the general rule for intents that they
1164 represent requirements of an implementation in the form of a restriction that cannot be relaxed.

1165 However, a restriction can be made more restrictive so that an unqualified version of an intent expressed
1166 through an annotation in the Java class can be qualified by a declarative intent in a using composite
1167 document.

1168 8.5 Policy Set Annotations

1169 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies. For example,
1170 a concrete policy is the specific encryption algorithm to use when encrypting messages when using a
1171 specific communication protocol to link a reference to a service.
1172 Policy Sets can be applied directly to Java implementations using the ***@PolicySets*** annotation. The
1173 @PolicySets annotation either takes the QName of a single policy set as a string or the name of two or
1174 more policy sets as an array of strings:

Deleted:

1175

1176
```
'@PolicySets({' policySetQName (',' policySetQName )* '})'
```

1177 *Snippet 8-21: PolicySet Annotation Format*

1178

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:

1181

```
1182 @Reference(name="helloService", required=true)
1183 @PolicySets({ MY_NS + "WS_Encryption_Policy",
1184               MY_NS + "WS_Authentication_Policy" })
1185 public setHelloService(HelloService service) {
1186    . . .
1187 }
```

1188 *Snippet 8-22: Use of @PolicySets*

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

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

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

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

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

[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:

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

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

- message
- transport

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

Snippet 8-23 shows an example of applying security intents to the setter method used to inject a reference. Accessing the hello operation of the referenced HelloService requires both "integrity.message" and "authentication.message" intents to be honored.

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

package services.client;
// Interface for ClientService
```

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:

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

```
public interface ClientService {
   public void clientMethod();
}

// Implementation class for ClientService
package services.client;

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

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

   private HelloService helloService;

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

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

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

Deleted: ¶
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
- local

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

Snippet 8-24 shows an example of applying a transaction intent to a component implementation, where 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.*;

```


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

```
@Service(HelloService.class)
@ManagedTransaction("global")
public class HelloServiceImpl implements HelloService {

   public void someMethod() {
         …
   }
}
```

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

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

9 Java API

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

9.1 Component Context

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

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

    String getURI();

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

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

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

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

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

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

   RequestContext getRequestContext();

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

}
```

Figure 9-1: ComponentContext Interface

getURI() method:

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

Returns:

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

Parameters:

- ***none***

Exceptions:

- ***none***

- ***getService(Class<B> businessInterface, String referenceName)*** – Returns a proxy for the reference defined by the current component. The getService() method takes as its

Deleted: The following Java code
Deleted:
Deleted:
Deleted:
Deleted: ¶
Deleted: ¶
Deleted:
Deleted: () - returns
Deleted: domain

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

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]

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

Deleted:)–

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

Returns:

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

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

  The ComponentContext.getService method MUST return null if the multiplicity of the reference named by the referenceName parameter is 0..1 and the reference has no target service configured. [JCA80010]

Parameters:

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

Exceptions:

- [JCA80001]
- The ComponentContext.getService method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter. [JCA80011]
- The ComponentContext.getService method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter. [JCA80012]

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

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

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

Returns:

- ***ServiceReference<B>*** which is a ServiceReference proxy object for the reference, which implements the interface contained in the businessInterface parameter.
  The ComponentContext.getServiceReference method MUST return a ServiceReference object typed by the interface provided by the businessInterface parameter, for the reference named by the referenceName parameter with the interface defined by the businessInterface parameter when that reference has a target service configured. [JCA80013]
  The ComponentContext.getServiceReference method MUST return null if the multiplicity of the reference named by the referenceName parameter is 0..1 and the reference has no target service configured. [JCA80007]

Parameters:

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

Exceptions:

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

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

- The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter does not have an interface of the type defined by the businessInterface parameter. [JCA80005]
- The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the component does not have a reference with the name provided in the referenceName parameter. [JCA80006]

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

Deleted: –

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

Deleted: business interface type and a reference name.

Returns:

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

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

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

Parameters:

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

Exceptions:

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

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

Deleted: –

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

Deleted: service references for a business interface type and a reference name.

Returns:

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

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

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

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

1444 Parameters:

1445
- ***Class<B> businessInterface*** - the Java interface for the service reference

1446
- ***String referenceName*** - the name of the service reference

1447 Exceptions:

1448-1449
- The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1. [JCA80021]

1450-1452
- The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter. [JCA80022]

1453-1455
- The ComponentContext.getServiceReferences 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. [JCA80023]

1456

1457 ***createSelfReference(Class<B> businessInterface) method:***

Deleted: – Returns a ServiceReference

1458-1459 Returns a ServiceReference object that can be used to invoke this component over the designated service.

1460 Returns:

1461-1463
- ***ServiceReference<B>*** which is a ServiceReference object for the service of this component which has the supplied business interface. If the component has multiple services with the same business interface the SCA runtime can return a ServiceReference for any one of them.

1464-1466 The ComponentContext.createSelfReference method MUST return a ServiceReference object typed by the interface defined by the businessInterface parameter for one of the services of the invoking component which has the interface defined by the businessInterface parameter. [JCA80024]

1467 Parameters:

1468
- ***Class<B> businessInterface*** - the Java interface for the service

1469 Exceptions:

1470-1472
- 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. [JCA80025]

1473

1474 ***createSelfReference(Class<B> businessInterface, String serviceName) method:***

Deleted: –

1475-1476 Returns a ServiceReference that can be used to invoke this component over the designated service. The serviceName parameter explicitly declares the service name to invoke

1477 Returns:

1478-1479
- ***ServiceReference<B>*** which is a ServiceReference proxy object for the reference, which implements the interface contained in the businessInterface parameter.

1480-1483 The ComponentContext.createSelfReference method MUST return a ServiceReference object typed by the interface defined by the businessInterface parameter for the service identified by the serviceName of the invoking component and which has the interface defined by the businessInterface parameter. [JCA80026]

1484 Parameters:

1485
- ***Class<B> businessInterface*** - the Java interface for the service reference

1486
- ***String serviceName*** - the name of the service reference

1487 Exceptions:

1488-1490
- The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the component does not have a service with the name identified by the serviceName parameter. [JCA80027]

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

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

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

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

Returns:

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

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

Parameters:

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

Exceptions:

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

getRequestContext() method:

Returns the RequestContext for the current SCA service request.

Returns:

- ***RequestContext*** which is the RequestContext object for the current SCA service invocation. ***null*** if there is no current request or if the context is unavailable.

  When marked for eager initialization with an @EagerInit annotation, the composite scoped instance MUST be created when its containing component is started. [JCA80002]

Parameters:

- ***none***

Exceptions:

- ***none***

cast(B target) method:

Casts a type-safe reference to a ServiceReference

Returns:

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

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

Parameters:

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

1537 • ***B target*** - a type safe reference proxy object which implements the business interface B

1538 Exceptions:

1539 • The ComponentContext.cast method MUST throw an IllegalArgumentException if the supplied target
1540 parameter is not an SCA reference proxy object. [JCA80033]

1541 A component can access its component context by defining a field or setter method typed by
1542 ***org.oasisopen.sca.ComponentContext*** and annotated with ***@Context***. To access a target service, the
1543 component uses ***ComponentContext.getService(..).***

1544 Snippet 9-1 shows an example of component context usage in a Java class using the @Context
1545 annotation.

Deleted: The following

```
1546    private ComponentContext componentContext;
1547
1548    @Context
1549    public void setContext(ComponentContext context) {
1550       componentContext = context;
1551    }
1552
1553    public void doSomething() {
1554       HelloWorld service =
1555          componentContext.getService(HelloWorld.class,"HelloWorldComponent");
1556       service.hello("hello");
1557    }
```

Deleted:

1558 *Snippet 9-1: ComponentContext Injection Example*

1559 Similarly, non-SCA client code can use the ComponentContext API to perform operations against a
1560 component in an SCA domain. How the non-SCA client code obtains a reference to a ComponentContext
1561 is runtime specific.

1562 9.2 Request Context

1563 Figure 9-2 shows the ***RequestContext*** interface:
1564

Deleted: The following

```
1565    package org.oasisopen.sca;
1566
1567    import javax.security.auth.Subject;
1568
1569    public interface RequestContext {
1570
1571       Subject getSecuritySubject();
1572
1573       String getServiceName();
1574          <CB> ServiceReference<CB> getCallbackReference();
1575          <CB> CB getCallback();
1576       <B> ServiceReference<B> getServiceReference();
1577
1578    }
1579
```

Deleted: ¶

1580 The RequestContext interface has the following methods:

1581 • ***getSecuritySubject()*** – Returns the JAAS Subject of the current request (see the JAAS
1582 Reference Guide [JAAS] for details of JAAS)

1583 • ***getServiceName()*** – Returns the name of the service on the Java implementation the
1584 request came in on

1585 • ***getCallbackReference()*** – Returns a service reference to the callback as specified by the
1586 caller. This method returns null when called for a service request whose interface is not
1587 bidirectional or when called for a callback request.

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

- ***getCallback()*** – Returns a proxy for the callback as specified by the caller. Similar to the getCallbackReference() method, this method returns null when called for a service request whose interface is not bidirectional or when called for a callback request.
- ***getServiceReference()*** – When invoked during the execution of a service operation, the RequestContext.getServiceReference method MUST return a ServiceReference that represents the service that was invoked. [JCA80003]

9.3 ServiceReference

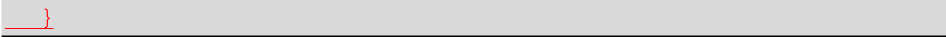


Figure 9-2: RequestContext Interface

getSecuritySubject ( ) method:

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

Returns:

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

  null if there is no subject for the request.

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

Parameters:

- ***none***

Exceptions:

- ***none***

getServiceName ( ) method:

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

Returns:

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

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

Parameters:

- ***none***

Exceptions:

- ***none***

getCallbackReference ( ) method:

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

Returns:

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

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

1631 ***null*** if the invoked service has an interface which is not bidirectional or if the getCallbackReference()
1632 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 The RequestContext.getCallbackReference method MUST return a ServiceReference object typed by
1635 the interface of the callback supplied by the client of the invoked service, or null if either the invoked
1636 service is not bidirectional or if the method is invoked from a thread that is not processing a service
1637 operation. [JCA80036]

1638 Parameters:

1639 • ***none***

1640 Exceptions:

1641 • ***none***

1642

1643 ***getCallback ( ) method:***

1644 Returns a proxy for the callback for the invoked service as specified by the service client.

1645 Returns:

1646 • ***CB*** proxy object for the callback for the invoked service as supplied by the service client. It is typed
1647 with the callback interface.

1648 ***null*** if the invoked service has an interface which is not bidirectional or if the getCallback() method is
1649 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 The RequestContext.getCallback method MUST return a reference proxy object typed by the
1652 interface of the callback supplied by the client of the invoked service, or null if either the invoked
1653 service is not bidirectional or if the method is invoked from a thread that is not processing a service
1654 operation. [JCA80037]

1655 Parameters:

1656 • ***none***

1657 Exceptions:

1658 • ***none***

1659

1660 ***getServiceReference ( ) method:***

1661 Returns a ServiceReference object for the service that was invoked.

1662 Returns:

1663 • ***ServiceReference<B>*** which is a service reference for the invoked service. It is typed with the
1664 interface of the service.

1665 ***null*** if the method is invoked from a thread that is not processing a service operation or a callback
1666 operation.

1667 When invoked during the execution of a service operation, the RequestContext.getServiceReference
1668 method MUST return a ServiceReference that represents the service that was invoked. [JCA80003]

1669 When invoked during the execution of a callback operation, the RequestContext.getServiceReference
1670 method MUST return a ServiceReference that represents the callback that was invoked. [JCA80038]

1671 When invoked from a thread not involved in the execution of either a service operation or of a
1672 callback operation, the RequestContext.getServiceReference method MUST return null. [JCA80039]

1673 Parameters:

1674 • ***none***

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

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

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

```
package org.oasisopen.sca;

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


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

Deleted: ¶

Deleted: }¶
¶
The

Figure 9-3: ServiceReference Interface

Deleted: *interface has the following methods:*

getService( ) method:

Deleted: **()** -

Returns a type-safe reference to the target of this reference. The instance returned is guaranteed to implement the business interface for this reference. The value returned is a proxy to the target that implements the business interface associated with this reference.

Returns:

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

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

Parameters:

- ***none***

Exceptions:

- ***none***

getBusinessInterface( ) method:

Deleted: **()** –

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

Deleted: reference

Returns:

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

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

Parameters:

- ***none***

Exceptions:

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

1720 • ***none***

1721 9.5 ResponseDispatch interface

1722 The ***ResponseDispatch*** interface is shown in Figure 9-4:

1723

```
1724 package org.oasisopen.sca;
1725
1726 public interface ResponseDispatch<T> {
1727     void sendResponse(T res);
1728     void sendFault(Throwable e);
1729     Map<String, Object> getContext();
1730 }
```

1731 *Figure 9-4: ResponseDispatch Interface*

1732

1733 ***sendResponse (T response) method:***

1734 Sends the response message from an asynchronous service method. This method can only be invoked
1735 once for a given ResponseDispatch object and cannot be invoked if sendFault has previously been
1736 invoked for the same ResponseDispatch object.

1737 Returns:

1738 • ***void***

1739 The ResponseDispatch.sendResponse() method MUST send the response message to the client of
1740 an asynchronous service. [JCA50057]

1741 Parameters:

1742 • ***T*** - an instance of the response message returned by the service operation

1743 Exceptions:

1744 • The ResponseDispatch.sendResponse() method MUST throw an InvalidStateException if either the
1745 sendResponse method or the sendFault method has already been called once. [JCA80058]

1746

1747 ***sendFault (Throwable e) method:***

1748 Sends an exception as a fault from an asynchronous service method. This method can only be invoked
1749 once for a given ResponseDispatch object and cannot be invoked if sendResponse has previously been
1750 invoked for the same ResponseDispatch object.

1751 Returns:

1752 • ***void***

1753 The ResponseDispatch.sendFault() method MUST send the supplied fault to the client of an
1754 asynchronous service. [JCA80059]

1755 Parameters:

1756 • ***e*** - an instance of an exception returned by the service operation

1757 Exceptions:

1758 • The ResponseDispatch.sendFault() method MUST throw an InvalidStateException if either the
1759 sendResponse method or the sendFault method has already been called once. [JCA80060]

1760

1761 ***getContext () method:***

1762 Obtains the context object for the ResponseDispatch method

1763 Returns:

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

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

Parameters:

- ***none***

Exceptions:

- ***none***

9.6 ServiceRuntimeException

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

```
package org.oasisopen.sca;

public class ServiceRuntimeException extends RuntimeException {
   …
}
```

Figure 9-5: ServiceRuntimeException

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

9.7 ServiceUnavailableException

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

```
package org.oasisopen.sca;

public class ServiceUnavailableException extends ServiceRuntimeException {
   …
}
```

Figure 9-6: ServiceUnavailableException

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

9.8 InvalidServiceException

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

```
package org.oasisopen.sca;

public class InvalidServiceException extends ServiceRuntimeException {
   …
}
```

Figure 9-7: InvalidServiceException

Deleted: The following snippet
Deleted: The following snippet
Deleted: }¶
Deleted: The following snippet
Deleted: }¶

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

1806 This exception signals that the ServiceReference is no longer valid. This can happen when the target of
1807 the reference is undeployed. This exception is not transient and therefore is unlikely to be resolved by
1808 retrying the operation and will most likely require human intervention.

1809 9.9 Constants

1810 The SCA ***Constants*** interface defines a number of constant values that are used in the SCA Java APIs
1811 and Annotations. Figure 9-8 shows the Constants interface:

Deleted: The following snippet

```
1812 package org.oasisopen.sca;
1813
1814 public interface Constants {
1815
1816     String SCA_NS = "http://docs.oasis-open.org/ns/opencsa/sca/200912";
1817
1818     String SCA_PREFIX = "{"+SCA_NS+"}";
1819
1820     String SERVERAUTHENTICATION = SCA_PREFIX + "serverAuthentication";
1821     String CLIENTAUTHENTICATION = SCA_PREFIX + "clientAuthentication";
1822     String ATLEASTONCE = SCA_PREFIX + "atLeastOnce";
1823     String ATMOSTONCE = SCA_PREFIX + "atMostOnce";
1824     String EXACTLYONCE = SCA_PREFIX + "exactlyOnce";
1825     String ORDERED = SCA_PREFIX + "ordered";
1826     String TRANSACTEDONEWAY = SCA_PREFIX + "transactedOneWay";
1827     String IMMEDIATEONEWAY = SCA_PREFIX + "immediateOneWay";
1828     String PROPAGATESTRANSACTION = SCA_PREFIX + "propagatesTransaction";
1829     String SUSPENDSTRANSACTION = SCA_PREFIX + "suspendsTransaction";
1830     String ASYNCINVOCATION = SCA_PREFIX + "asyncInvocation";
1831     String SOAP = SCA_PREFIX + "SOAP";
1832     String JMS = SCA_PREFIX + "JMS";
1833     String NOLISTENER = SCA_PREFIX + "noListener";
1834     String EJB = SCA_PREFIX + "EJB";
1835
1836 }
```

Deleted:

Deleted: =

Deleted: 200903

Deleted: }

1837 *Figure 9-8: Constants Interface*

1838 9.10 SCAClientFactory Class

1839 The SCAClientFactory class provides the means for client code to obtain a proxy reference object for a
1840 service within an SCA Domain, through which the client code can invoke operations of that service. This
1841 is particularly useful for client code that is running outside the SCA Domain containing the target service,
1842 for example where the code is "unmanaged" and is not running under an SCA runtime.

1843 The SCAClientFactory is an abstract class which provides a set of static newInstance(...) methods which
1844 the client can invoke in order to obtain a concrete object implementing the SCAClientFactory interface for
1845 a particular SCA Domain. The returned SCAClientFactory object provides a getService() method which
1846 provides the client with the means to obtain a reference proxy object for a service running in the SCA
1847 Domain.

1848 The SCAClientFactory class is shown in Figure 9-9:

1849

```
1850 package org.oasisopen.sca.client;
1851
1852 import java.net.URI;
1853 import java.util.Properties;
1854
1855 import org.oasisopen.sca.NoSuchDomainException;
1856 import org.oasisopen.sca.NoSuchServiceException;
1857 import org.oasisopen.sca.client.SCAClientFactoryFinder;
1858 import org.oasisopen.sca.client.impl.SCAClientFactoryFinderImpl;
1859
```


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

```
public abstract class SCAClientFactory {

    protected static SCAClientFactoryFinder factoryFinder;

    private URI domainURI;

    private SCAClientFactory() {
    }

    protected SCAClientFactory(URI domainURI)
         throws NoSuchDomainException {
        this.domainURI = domainURI;
    }

    protected URI getDomainURI() {
        return domainURI;
    }

    public static SCAClientFactory newInstance( URI domainURI )
         throws NoSuchDomainException {
        return newInstance(null, null, domainURI);
    }

    public static SCAClientFactory newInstance(Properties properties,
                                                                URI domainURI)
         throws NoSuchDomainException {
        return newInstance(properties, null, domainURI);
    }

    public static SCAClientFactory newInstance(ClassLoader classLoader,
                                                                URI domainURI)
         throws NoSuchDomainException {
        return newInstance(null, classLoader, domainURI);
    }

    public static SCAClientFactory newInstance(Properties properties,
                                               ClassLoader classLoader,
                                               URI domainURI)
         throws NoSuchDomainException {
        final SCAClientFactoryFinder finder =
            factoryFinder != null ? factoryFinder :
                new SCAClientFactoryFinderImpl();
        final SCAClientFactory factory
            = finder.find(properties, classLoader, domainURI);
        return factory;
    }

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

Figure 9-9: SCAClientFactory Class

newInstance (URI domainURI) method:

Obtains a object implementing the SCAClientFactory class.

Returns:

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

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

Parameters:

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

1919 • ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1920 Exceptions:

1921 • The SCAClientFactory.newInstance(URI) method MUST throw a NoSuchDomainException if the
1922 domainURI parameter does not identify a valid SCA Domain. [JCA80043]

1923

1924 ***newInstance(Properties properties, URI domainURI) method:***

1925 Obtains a object implementing the SCAClientFactory class, using a specified set of properties.

1926 Returns:

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

1928 The SCAClientFactory.newInstance(Properties, URI) method MUST return an object which
1929 implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.
1930 [JCA80044]

1931 Parameters:

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

1934 • ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1935 Exceptions:

1936 • The SCAClientFactory.newInstance(Properties, URI) method MUST throw a
1937 NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.
1938 [JCA80045]

1939

1940 ***newInstance(Classloader classLoader, URI domainURI) method:***

1941 Obtains a object implementing the SCAClientFactory class using a specified classloader.

1942 Returns:

1943 • ***object*** which implements the SCAClientFactory class

1944 The SCAClientFactory.newInstance(Classloader, URI) method MUST return an object which
1945 implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.
1946 [JCA80046]

1947 Parameters:

1948 • ***classLoader*** - a ClassLoader to use when creating the object which implements the
1949 SCAClientFactory class.

1950 • ***domainURI*** - a URI for the SCA Domain which is targeted by the returned SCAClient object

1951 Exceptions:

1952 • The SCAClientFactory.newInstance(Classloader, URI) method MUST throw a
1953 NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.
1954 [JCA80047]

1955

1956 ***newInstance(Properties properties, Classloader classLoader, URI domainURI) method:***

1957 Obtains a object implementing the SCAClientFactory class using a specified set of properties and a
1958 specified classloader.

1959 Returns:

1960 • ***object*** which implements the SCAClientFactory class

1961 The SCAClientFactory.newInstance(Properties, Classloader, URI) method MUST return an object
1962 which implements the SCAClientFactory class for the SCA Domain identified by the domainURI
1963 parameter. [JCA80048]

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

Parameters:

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

Exceptions:

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

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

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

Returns:

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

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

Parameters:

- ***interfaze*** - a Java interface class which is the business interface of the target service
- ***serviceURI*** - a String containing the relative URI of the target service within its SCA Domain.

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

Exceptions:

- The SCAClientFactory.getService method MUST throw a NoSuchServiceException if a service with the relative URI serviceURI and a business interface which matches interfaze cannot be found in the SCA Domain targeted by the SCAClient object. [JCA80051]
- The SCAClientFactory.getService method MUST throw a NoSuchServiceException if the domainURI of the SCAClientFactory does not identify a valid SCA Domain. [JCA80052]

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

getDomainURI() method:

Obtains the Domain URI value for this SCAClientFactory

Returns:

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

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

Parameters:

- ***none***

Exceptions:

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

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

private SCAClientFactory() method:

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

factoryFinder protected field:

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

9.11 SCAClientFactoryFinder Interface

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

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

public interface SCAClientFactoryFinder {

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

Figure 9-10: SCAClientFactoryFinder Interface

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

Obtains an implementation of the SCAClientFactory interface.

Returns:

- ***SCAClientFactory*** implementation object

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

Parameters:

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

Exceptions:

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

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

2053
9.12 SCAClientFactoryFinderImpl Class

2054 2055 2056
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.

2057

2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069
```
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 {...}
   ...
}
```

2070
Snippet 9-2: SCAClientFactoryFinderImpl Class

2071

2072
SCAClientFactoryFinderImpl () method:

2073
Public constructor for the SCAClientFactoryFinderImpl.

2074
Returns:

2075
- ***SCAClientFactoryFinderImpl*** which implements the SCAClientFactoryFinder interface

2076
Parameters:

2077
- ***none***

2078
Exceptions:

2079
- ***none***

2080

2081
find (Properties, ClassLoader, URI) method:

2082 2083
Obtains an implementation of the SCAClientFactory interface. It discovers a provider's SCAClientFactory implementation by referring to the following information in this order:

2084 2085 2086 2087
1. The org.oasisopen.sca.client.SCAClientFactory property from the Properties specified on the newInstance() method call if specified
2. The org.oasisopen.sca.client.SCAClientFactory property from the System Properties
3. The META-INF/services/org.oasisopen.sca.client.SCAClientFactory file

2088
Returns:

2089
- ***SCAClientFactory*** implementation object

2090
Parameters:

2091 2092 2093 2094 2095
- ***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

2096
Exceptions:

2097
- ***ServiceRuntimeException*** - if the SCAClientFactory implementation could not be found

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

2098 ## 9.13 NoSuchDomainException

2099 Figure 9-11 shows the ***NoSuchDomainException***:

2100

```
2101 package org.oasisopen.sca;
2102
2103 public class NoSuchDomainException extends Exception {
2104     ...
2105 }
```

2106 *Figure 9-11: NoSuchDomainException Class*

2107

2108 This exception indicates that the Domain specified could not be found.

2109 ## 9.14 NoSuchServiceException

2110 Figure 9-12 shows the ***NoSuchServiceException***:

2111

```
2112 package org.oasisopen.sca;
2113
2114 public class NoSuchServiceException extends Exception {
2115     ...
2116 }
```

2117 *Figure 9-12: NoSuchServiceException Class*

2118

2119 This exception indicates that the service specified could not be found.

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

10 Java Annotations

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

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

Field Code Changed

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]

Field Code Changed

10.1 @AllowsPassByReference

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

Deleted: The following Java code

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

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

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

The @AllowsPassByReference annotation has the attribute:

Deleted: following

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

The @AllowsPassByReference annotation MUST only annotate the following locations:

Field Code Changed

Formatted: Bullets and Numbering

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]

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

2165 The "allows pass by reference" marking of a method implementation of a remotable service is determined
2166 as follows:

2167 1. If the method has an @AllowsPassByReference annotation, the method is marked "allows pass by
2168 reference" if and only if the value of the method's annotation is true.
2169 2. Otheriwse, if the class has an @AllowsPassByReference annotation, the method is marked "allows
2170 pass by reference" if and only if the value of the class's annotation is true.
2171 3. Otherwise, the method is not marked "allows pass by reference".

2172 The "allows pass by reference" marking of a reference for a remotable service is determined as follows:

2173 1. If the reference has an @AllowsPassByReference annotation, the reference is marked "allows pass
2174 by reference" if and only if the value of the reference's annotation is true.
2175 2. Otherwise, if the service implementation class containing the reference has an
2176 @AllowsPassByReference annotation, the reference is marked "allows pass by reference" if and only
2177 if the value of the class's annotation is true.
2178 3. Otherwise, the reference is not marked "allows pass by reference".

2179 Snippet 10-1 shows a sample where @AllowsPassByReference is defined for the implementation of a
2180 service method on the Java component implementation class.

Deleted: ¶
The following snippet

2181

```
2182 @AllowsPassByReference
2183 public String hello(String message) {
2184    …
2185 }
```

2186 *Snippet 10-1: Use of @AllowsPassByReference on a Method*

Deleted: ¶
The following snippet

2187

2188 Snippet 10-2 shows a sample where @AllowsPassByReference is defined for a client reference of a Java
2189 component implementation class.

2190

```
2191 @AllowsPassByReference
2192 @Reference
2193 private StockQuoteService stockQuote;
```

2194 *Snippet 10-2: Use of @AllowsPassByReference on a Reference*

2195 10.2 @AsyncFault

2196 Figure 10-2 defines the ***@AsyncFault*** annotation:

2197

```
2198 package org.oasisopen.sca.annotation;
2199
2200 import static java.lang.annotation.ElementType.METHOD;
2201 import static java.lang.annotation.RetentionPolicy.RUNTIME;
2202
2203 import java.lang.annotation.Inherited;
2204 import java.lang.annotation.Retention;
2205 import java.lang.annotation.Target;
2206
2207 @Inherited
2208 @Target({METHOD})
2209 @Retention(RUNTIME)
2210 public @interface AsyncFault {
2211
2212    Class<?>[] value() default {};
2213
2214 }
```


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

Figure 10-2: AsyncFault Annotation

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

10.3 @AsyncInvocation

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

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

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

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

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

    boolean value() default true;
}
```

Figure 10-3: AsyncInvocation Annotation

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

10.4 @Authentication

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

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

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

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

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


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

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

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

Figure 10-4: Authentication Annotation

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

Deleted: that

Deleted: invocation requires

Deleted:

Deleted: and

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

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

Deleted: The following Java code

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

2320

```
2321 package org.oasisopen.sca.annotation;
2322
2323 import static java.lang.annotation.ElementType.FIELD;
2324 import static java.lang.annotation.ElementType.METHOD;
2325 import static java.lang.annotation.ElementType.TYPE;
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.

2341 The @Callback annotation has the attribute:

Deleted: following

2342
- ***value*** – the name of a Java class file containing the callback interface

Deleted: ¶

2343 The @Callback annotation can also be used to annotate a method or a field of an SCA implementation
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 any
2346 attributes. [JCA90046] When used to annotate a method or a field of an implementation class for injection
2347 of a callback object, the type of the method or field MUST be the callback interface of at least one
2348 bidirectional service offered by the implementation class. [JCA90054] When used to annotate a setter
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
2354 class that has COMPOSITE scope. [JCA90057]

Deleted: An

2355 Snippet 10-3 shows an example use of the @Callback annotation to declare a callback interface.

Deleted: follows:

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 }
```

2372 *Snippet 10-3: Use of @Callback*

Deleted: ¶
In this example, the

2373

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

2374 The implied component type is for Snippet 10-3 is shown in Snippet 10-4.

Deleted: :

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

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

Deleted: 200903

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

10.7 @ComponentName

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

Deleted: The following Java code

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

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

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

}
```

Deleted: ElementType.TYPE

Figure 10-7: ComponentName Annotation

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

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

Deleted: The following snippet

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

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

Deleted: ¶
The following snippet

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

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

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

10.8 @Confidentiality

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

Deleted: The following Java code

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	}
```

2451 *Figure 10-8: Confidentiality Annotation*

2452

2453 The ***@Confidentiality*** annotation is used to indicate the need for confidentiality. See the SCA Policy
2454 Framework Specification [POLICY] for details on the meaning of the intent. See the section on Application
2455 of Intent Annotations for samples of how intent annotations are used in Java.

Deleted: that
Deleted: invocation requires
Deleted: ¶
Deleted: and details

2456 ## 10.9 @Constructor

2457 Figure 10-9 defines the ***@Constructor*** annotation:

Deleted: The following Java code

2458

```
2459	package org.oasisopen.sca.annotation;
2460
2461	import static java.lang.annotation.ElementType.CONSTRUCTOR;
2462	import static java.lang.annotation.RetentionPolicy.RUNTIME;
2463	import java.lang.annotation.Retention;
2464	import java.lang.annotation.Target;
2465
2466	@Target(CONSTRUCTOR)
2467	@Retention(RUNTIME)
2468	public @interface Constructor { }
```

2469 *Figure 10-9: Constructor Annotation*

2470

2471 The @Constructor annotation is used to mark a particular constructor to use when instantiating a Java
2472 component implementation. If a constructor of an implementation class is annotated with @Constructor

Field Code Changed

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

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

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

Deleted: The following snippet

```
public class HelloServiceImpl implements HelloService {

   public HelloServiceImpl(){
    ...
    }

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

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

Snippet 10-7: Use of @Constructor

10.10 @Context

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

Deleted: The following Java code

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

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

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

}
```

Figure 10-10: Context Annotation

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

The @Context annotation has no attributes.

Snippet 10-8 shows a ComponentContext field definition sample.

Deleted: The following snippet

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

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

Deleted: ¶
The following snippet

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

Snippet 10-9 shows a RequestContext field definition sample.

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

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

10.11 @Destroy

Deleted: The following Java code

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

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

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

Figure 10-11 defines the ***@Destroy*** annotation:

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

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

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

}
```

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

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

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

Figure 10-11: Destroy Annotation

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

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

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

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


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

```
    …
}
```

Snippet 10-10: Use of @Destroy

10.12 @EagerInit

Figure 10-12: EagerInit Annotation defines the **@EagerInit** annotation:

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

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

@Target(TYPE)
@Retention(RUNTIME)
public @interface EagerInit {

}
```

Figure 10-12: EagerInit Annotation

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

10.13 @Init

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

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

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

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

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

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

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


}
```

Deleted: The following Java code

Field Code Changed

Deleted: The following Java code

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

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

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

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

Figure 10-13: Init Annotation

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

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

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

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

Snippet 10-11: Use of @Init

10.14 @Integrity

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

Deleted: The following Java code

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

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

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

@Inherited
@Target({TYPE, FIELD, METHOD, PARAMETER})
@Retention(RUNTIME)
@Intent(Integrity.INTEGRITY)
public @interface Integrity {
    String INTEGRITY = SCA_PREFIX + "integrity";
    String INTEGRITY_MESSAGE = INTEGRITY + ".message";
    String INTEGRITY_TRANSPORT = INTEGRITY + ".transport";

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


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

```
}
```

Figure 10-14: Integrity Annotation

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 "";

    /**
     * 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

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

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

10.16 @ManagedSharedTransaction

Figure 10-16 defines the @ManagedSharedTransaction annotation:

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


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

```
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  }
```

2744 *Figure 10-16: ManagedSharedTransaction Annotation*

2745

2746 The ***@ManagedSharedTransaction*** annotation is used to indicate the need for a distributed and globally
2747 coordinated ACID transaction. See the SCA Policy Framework Specification [POLICY] for details on the
2748 meaning of the intent. See the section on Application of Intent Annotations for samples of how intent
2749 annotations are used in Java.

2750 10.17 @ManagedTransaction

2751 Figure 10-17 defines the @ManagedTransaction annotation:

2752

```
2753  import static java.lang.annotation.ElementType.FIELD;
2754  import static java.lang.annotation.ElementType.METHOD;
2755  import static java.lang.annotation.ElementType.PARAMETER;
2756  import static java.lang.annotation.ElementType.TYPE;
2757  import static java.lang.annotation.RetentionPolicy.RUNTIME;
2758  import static org.oasisopen.sca.Constants.SCA_PREFIX;
2759
2760  import java.lang.annotation.Inherited;
2761  import java.lang.annotation.Retention;
2762  import java.lang.annotation.Target;
2763
2764  /**
2765   * The @ManagedTransaction annotation is used to indicate the
2766   * need for an ACID transaction environment.
2767   */
2768  @Inherited
2769  @Target({TYPE, FIELD, METHOD, PARAMETER})
2770  @Retention(RUNTIME)
2771  @Intent(ManagedTransaction.MANAGEDTRANSACTION)
2772  public @interface ManagedTransaction {
2773      String MANAGEDTRANSACTION = SCA_PREFIX + "managedTransaction";
2774      String MANAGEDTRANSACTION_MESSAGE = MANAGEDTRANSACTION + ".local";
2775      String MANAGEDTRANSACTION_TRANSPORT = MANAGEDTRANSACTION + ".global";
2776
2777      /**
2778       * List of managedTransaction qualifiers (such as "global" or "local").
```


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

```
     *
     * @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;
```


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

```
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  }
```

2850 *Figure 10-19: NoManagedTransaction Annotation*

2851

2852 The **@NoManagedTransaction** annotation is used to indicate that the component does not want to run in
2853 an ACID transaction. See the SCA Policy Framework Specification [POLICY] for details on the meaning
2854 of the intent. See the section on Application of Intent Annotations for samples of how intent annotations
2855 are used in Java.

2856 10.20 @OneWay

2857 Figure 10-20 defines the ***@OneWay*** annotation:

Deleted: The following Java code

2858

```
2859  package org.oasisopen.sca.annotation;
2860
2861  import static java.lang.annotation.ElementType.METHOD;
2862  import static java.lang.annotation.RetentionPolicy.RUNTIME;
2863  import java.lang.annotation.Retention;
2864  import java.lang.annotation.Target;
2865
2866  @Target(METHOD)
2867  @Retention(RUNTIME)
2868  public @interface OneWay {
2869
2870
2871  }
```

2872 *Figure 10-20: OneWay Annotation*

Deleted: ¶
The @OneWay annotation is used on a Java interface or class method to indicate that invocations will be dispatched in a non-blocking fashion as described in the section on Asynchronous Programming.¶

2873

2874 A method annotated with @OneWay MUST have a void return type and MUST NOT have declared
2875 checked exceptions. [JCA90055]

2876 When a method of a Java interface is annotated with @OneWay, the SCA runtime MUST ensure that all
2877 invocations of that method are executed in a non-blocking fashion, as described in the section on
2878 Asynchronous Programming. [JCA90056]

2879 The @OneWay annotation has no attributes.

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

Deleted: The following snippet

2881

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

```
package services.hello;

import org.oasisopen.sca.annotation.OneWay;

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

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)
```


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

```
public @interface Property {

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

Figure 10-22: Property Annotation

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

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

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

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

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

The @Property annotation has the attributes:

- ***name (0..1)*** – the name of the property. For a field annotation, the default is the name of the field of the Java class. For a setter method annotation, the default is the JavaBeans property name [JAVABEANS] corresponding to the setter method name. For a @Property annotation applied to a constructor parameter, there is no default value for the name attribute and the name attribute MUST be present. [JCA90013]
- ***required (0..1)*** – a boolean value which specifies whether injection of the property value is required or not, where true means injection is required and false means injection is not required. Defaults to true. For a @Property annotation applied to a constructor parameter, the required attribute MUST NOT have the value false. [JCA90014]

  }

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

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

Snippet 10-13 shows a property field definition sample.

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

The following snippet shows a property setter sample

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

Field Code Changed

Deleted: following

Deleted: *optional*

Deleted: *optional*

Deleted: ¶
The following snippet

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

2985 For a @Property annotation, if the type of the Java class field or the type of the input parameter of
2986 the setter method or constructor is defined as an array or as any type that extends or implements
2987 java.util.Collection, then the SCA runtime MUST introspect the component type of the
2988 implementation with a <property/> element with a @many attribute set 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 java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation
2997 with a <property/> element with a @many attribute set to true, otherwise @many MUST be set to false.
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 }
3008 ...
```

3009 *Snippet 10-14: Use of @Property with a Collection*

3010 10.23 @Qualifier

3011 Figure 10-23 defines the ***@Qualifier*** annotation:

Deleted: The following Java code

3012

```
3013 package org.oasisopen.sca.annotation;
3014
3015 import static java.lang.annotation.ElementType.METHOD;
3016 import static java.lang.annotation.RetentionPolicy.RUNTIME;
3017
3018 import java.lang.annotation.Retention;
3019 import java.lang.annotation.Target;
3020
3021 @Target(METHOD)
3022 @Retention(RUNTIME)
3023 public @interface Qualifier {
3024 }
```

3025 *Figure 10-23: Qualifier Annotation*

3026

3027 The @Qualifier annotation is applied to an attribute of a specific intent annotation definition, defined using
3028 the @Intent annotation, to indicate that the attribute provides qualifiers for the intent. The @Qualifier
3029 annotation MUST be used in a specific intent annotation definition where the intent has qualifiers.
3030 [JCA90015]

Field Code Changed

3031 See the section "How to Create Specific Intent Annotations" for details and samples of how to define new
3032 intent annotations.

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

3033 ## 10.24 @Reference

3034 Figure 10-24 defines the ***@Reference*** annotation:

3035

```
3036 package org.oasisopen.sca.annotation;
3037
3038 import static java.lang.annotation.ElementType.FIELD;
3039 import static java.lang.annotation.ElementType.METHOD;
3040 import static java.lang.annotation.ElementType.PARAMETER;
3041 import static java.lang.annotation.RetentionPolicy.RUNTIME;
3042 import java.lang.annotation.Retention;
3043 import java.lang.annotation.Target;
3044 @Target({METHOD, FIELD, PARAMETER})
3045 @Retention(RUNTIME)
3046 public @interface Reference {
3047
3048    String name() default "";
3049    boolean required() default true;
3050 }
```

3051 *Figure 10-24: Reference Annotation*

3052

3053 The @Reference annotation type is used to annotate a Java class field, a setter method, or a constructor
3054 parameter that is used to inject a service that resolves the reference. The interface of the service injected
3055 is defined by the type of the Java class field or the type of the input parameter of the setter method or
3056 constructor.

3057 The @Reference annotation MUST NOT be used on a class field that is declared as final. [JCA90016]

3058 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:

3060 • ***name : String (0..1)*** – the name of the reference. For a field annotation, the default is the name of the
3061 field of the Java class. For a setter method annotation, the default is the JavaBeans property name
3062 corresponding to the setter method name. For a @Reference annotation applied to a constructor
3063 parameter, there is no default for the name attribute and the name attribute MUST be present.
3064 [JCA90018]

3065 • ***required (0..1)*** – a boolean value which specifies whether injection of the service reference is
3066 required or not, where true means injection is required and false means injection is not required.
3067 Defaults to true. For a @Reference annotation applied to a constructor parameter, the required
3068 attribute MUST have the value true. [JCA90019]

3069 Snippet 10-15 shows a reference field definition sample.

3070

```
3071 @Reference(name="stockQuote", required=true)
3072 protected StockQuoteService stockQuote;
```

3073 *Snippet 10-15: Use of @Reference on a Field*

3074

3075 Snippet 10-16 shows a reference setter sample

3076

```
3077 @Reference(name="stockQuote", required=true)
3078 public void setStockQuote( StockQuoteService theSQService ) {
3079     ...
3080 }
```

3081 *Snippet 10-16: Use of @Reference on a Setter*

Deleted: The following Java code
Field Code Changed
Deleted: following
Deleted: *optional*
Deleted: *optional*
Deleted: ¶ The following snippet
Deleted: ¶ *The following snippet*
Deleted: ¶ *The following fragment from*
Deleted: *component implementation*

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

3082

3083 Snippet 10-17 shows a sample of a service reference using the @Reference annotation. The name of the reference is “helloService” and its type is HelloService. The clientMethod() calls the “hello” operation of the service referenced by the helloService reference.

```
package services.hello;

private HelloService helloService;

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

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

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

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

Deleted: The following snippet

Deleted: above

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

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

</componentType>
```

Deleted: 200903

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

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

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]

Field Code Changed

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

Deleted: The following fragment from a component implementation

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

   public void clientMethod() {

```


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

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

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

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

Snippet 10-20 shows the XML representation of the component type reflected from for the former component implementation fragment. There is no need to author this component type in this case since it can be reflected from the Java class.

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

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

</componentType>
```

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

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

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

10.24.1 Reinjection

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

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

1. The component MUST NOT be STATELESS scoped.
2. The reference MUST use either field-based injection or setter injection. References that are injected through constructor injection MUST NOT be changed.

[JCA90025]

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

If a reference target changes and the reference is not reinjected, the reference MUST continue to work as if the reference target was not changed. [JCA90026]

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

3185 If an operation is called on a reference where the target of that reference has been undeployed, the SCA
3186 runtime SHOULD throw an InvalidServiceException. [JCA90027] If an operation is called on a reference
3187 where the target of the reference has become unavailable for some reason, the SCA runtime SHOULD
3188 throw a ServiceUnavailableException. [JCA90028] If the target service of the reference is changed, the
3189 reference MUST either continue to work or throw an InvalidServiceException when it is invoked.
3190 [JCA90029] If it doesn't work, the exception thrown will depend on the runtime and the cause of the
3191 failure.

3192 A ServiceReference that has been obtained from a reference by ComponentContext.cast() corresponds
3193 to the reference that is passed as a parameter to cast(). If the reference is subsequently reinjected, the
3194 ServiceReference obtained from the original reference MUST continue to work as if the reference target
3195 was not changed. [JCA90030] If the target of a ServiceReference has been undeployed, the SCA runtime
3196 SHOULD throw a InvalidServiceException when an operation is invoked on the ServiceReference.
3197 [JCA90031] If the target of a ServiceReference has become unavailable, the SCA runtime SHOULD
3198 throw a ServiceUnavailableException when an operation is invoked on the ServiceReference.
3199 [JCA90032] If the target service of a ServiceReference is changed, the reference MUST either continue
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
3203 getServiceReference() MUST correspond to the current configuration of the domain. This applies whether
3204 or not reinjection has taken place. [JCA90034] If the target of a reference or ServiceReference accessed
3205 through the component context by calling getService() or getServiceReference() has been undeployed or
3206 has become unavailable, the result SHOULD be a reference to the undeployed or unavailable service,
3207 and attempts to call business methods SHOULD throw an InvalidServiceException or a
3208 ServiceUnavailableException. [JCA90035] If the target service of a reference or ServiceReference
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
3215 collection change when the wiring changes or the targets change, then for references that use setter
3216 injection, the setter method MUST be called by the SCA runtime for any change to the contents.
3217 [JCA90038] A reinjected array or Collection for a reference MUST NOT be the same array or Collection
3218 object previously injected to the component. [JCA90039]

3219

	Effect on		
Change event	Injected Reference or ServiceReference	Existing ServiceReference Object**	Subsequent invocations of ComponentContext.getServiceReference() or getService()
Change to the target of the reference	can be reinjected (if other conditions* apply). If not reinjected, then it continues to work as if the reference target was not changed.	continue to work as if the reference target was not changed.	Result corresponds to the current configuration of the domain.
Target service undeployed	Business methods throw InvalidServiceException.	Business methods throw InvalidServiceException.	Result is a reference to the undeployed service. Business methods throw InvalidServiceException.
Target service becomes	Business methods throw ServiceUnavailableExce	Business methods throw ServiceUnavailableExce	Result is be a reference to the unavailable service. Business methods throw

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

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().			

3220 *Table 10-1Reinjection Effects*

3221 10.25 @Remotable

3222 Figure 10-25 defines the ***@Remotable*** annotation:

Deleted: The following Java code

3223

```
3224 package org.oasisopen.sca.annotation;
3225
3226 import static java.lang.annotation.ElementType.TYPE;
3227 import static java.lang.annotation.RetentionPolicy.RUNTIME;
3228 import java.lang.annotation.Retention;
3229 import java.lang.annotation.Target;
3230
3231
3232 @Target(TYPE)
3233 @Retention(RUNTIME)
3234 public @interface Remotable {
3235
3236 }
```

3237 *Figure 10-25: Remotable Annotation*

3238

3239 The @Remotable annotation is used to indicate that an SCA service interface is remotable. The
3240 @Remotable annotation is valid only on a Java interface, a Java class, a field, a setter method, or a constructor
3241 parameter. It MUST NOT appear anywhere else. [JCA90053] A remotable service can be published
3242 externally as a service and MUST be translatable into a WSDL portType. [JCA90040]

Deleted: annotate a Java

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

Field Code Changed

3243 The @Remotable annotation has no attributes. When placed on a Java service interface, it indicates that
3244 the interface is remotable. When placed on a Java service implementation class, it indicates that all SCA
3245 service interfaces provided by the class (including the class itself, if the class defines an SCA service
3246 interface) are remotable. When placed on a service reference, it indicates that the interface for the
3247 reference is remotable.

3248 Snippet 10-21 shows the Java interface for a remotable service with its @Remotable annotation.

Deleted: The following snippet

3249

```
3250 package services.hello;
3251
```


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

```
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.

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

Deleted: The following snippet

Deleted: remotable

```
package services.hello;

import org.oasisopen.sca.annotation.*;

public interface HelloService {

   String hello(String message);
}

package services.hello;

import org.oasisopen.sca.annotation.*;

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

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

Deleted: @Remotable¶

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

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.

```
package services.hello;

import org.oasisopen.sca.annotation.*;

public interface HelloService {
```


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

```
    String hello(String message);
}

package services.hello;

import org.oasisopen.sca.annotation.*;

public class HelloClient {

    @Remotable
    @Reference
    protected HelloService myHello;

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

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

10.26 @Requires

Figure 10-26 defines the ***@Requires*** annotation:

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

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

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)
3369    @Retention(RUNTIME)
3370    public @interface Scope {
3371
3372       String value() default "STATELESS";
3373    }
```

3374 *The @Scope annotation MUST only be used on a service's implementation class. It is an error to use this*
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]

3379 The @Scope annotation has the attribute:

3380 • ***value*** – the name of the scope.

3381 SCA defines the following scope names, but others can be defined by particular Java-based
3382 implementation types

3383 STATELESS

3384 COMPOSITE

3385 The default value is STATELESS.

3386 Snippet 10-24 shows a sample for a COMPOSITE scoped service implementation:

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*

3401 10.28 @Service

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;
```

Deleted: following

Deleted:

Deleted: :

Deleted:

Deleted:

Deleted: The following snippet

Deleted: The following Java code

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

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

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

Figure 10-28: Service Annotation

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

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

11 WSDL to Java and Java to WSDL

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

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

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

11.1 JAX-WS Annotations and SCA Interfaces

A Java class or interface used to define an SCA interface can contain JAX-WS annotations. In addition to affecting the Java to WSDL mapping defined by the JAX-WS specification [JAX-WS] these annotations can impact the SCA interface. An SCA runtime MUST apply the JAX-WS annotations as described in Table 11-1 and Table 11-2 when introspecting a Java class or interface class. [JCA100011] This could mean that the interface of a Java implementation is defined by a WSDL interface declaration.

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

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

		[JCA100013]
	endpointInterface	A Java class annotated with the @WebService annotation with its endpointInterface attribute set MUST have its interface defined by the referenced interface instead of annotated Java class. [JCA100014]
	portName	None
@WebMethod		
	operationName	**Sets operation name**
	action	None
	exclude	**Method is excluded from the interface.**
@OneWay		The SCA runtime MUST treat an @org.oasisopen.sca.annotation.OneWay annotation as a synonym for the @javax.jws.OneWay annotation. [JCA100002]
@WebParam		
	name	**Sets parameter name**
	targetNamespace	None
	mode	**Sets directionality of parameter**
	header	A Java class or interface containing an @WebParam annotation with its header attribute set to "true" MUST be treated as if the SOAP intent is applied to the Java class or interface. [JCA100015]
	partName	**Overrides name**

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

@WebResult		
	name	**Sets parameter name**
	targetNamespace	None
	header	A Java class or interface containing an @WebResult annotation with its header attribute set to "true" MUST be treated as if the SOAP intent is applied to the Java class or interface. [JCA100016]
	partName	**Overrides name**
@SOAPBinding		A Java class or interface containing an @SOAPBinding annotation MUST be treated as if the SOAP intent is applied to the Java class or interface. [JCA100021]
	style	
	use	
	parameterStyle	
@HandlerChain		None
	file	
	name	

3480 *Table 11-1: JSR 181 Annotations and SCA Interfaces*

3481

Annotation	*Property*	*Impact to SCA Interface*
@ServiceMode		A Java class containing an @ServiceMode annotation MUST be treated as if the SOAP intent is applied to the Java class. [JCA100017]

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

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

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

Annotation	*Property*	*Impact to SCA Interface*
	name	
@WebServiceProvider		A class annotated with @WebServiceProvider MUST be treated as if annotated with the SCA @Remotable annotation. [JCA100019]
	wsdlLocation	A Java class annotated with the @WebServiceProvider annotation with its wsdlLocation attribute set MUST have its interface defined by the referenced WSDL definition is used instead of the annotated Java class. [JCA100020]
	serviceName	None
	portName	None
	targetNamespace	None
@BindingType		None
	value	
@WebServiceRef		See JEE specification
	name	
	wsdlLocation	
	type	
	value	
	mappedName	
@WebServiceRefs		See JEE specification

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

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

3482 Table 11-2: JSR 224 Annotations and SCA Interfaces

3483 11.2 JAX-WS Client Asynchronous API for a Synchronous Service

3484 The JAX-WS specification defines a mapping of a synchronous service invocation, which provides a client
3485 application with a means of invoking that service asynchronously, so that the client can invoke a service
3486 operation and proceed to do other work without waiting for the service operation to complete its
3487 processing. The client application can retrieve the results of the service either through a polling
3488 mechanism or via a callback method which is invoked when the operation completes.

3489 For SCA service interfaces defined using interface.java, the Java interface MUST NOT contain the
3490 additional client-side asynchronous polling and callback methods defined by JAX-WS. [JCA100006] For
3491 SCA reference interfaces defined using interface.java, the SCA runtime MUST support a Java interface
3492 which contains the additional client-side asynchronous polling and callback methods defined by JAX-WS.
3493 [JCA100007] If the additional client-side asynchronous polling and callback methods defined by JAX-WS
3494 are present in the interface which declares the type of a reference in the implementation, SCA Runtimes
3495 MUST NOT include these methods in the SCA reference interface in the component type of the
3496 implementation. [JCA100008]
3497 The additional client-side asynchronous polling and callback methods defined by JAX-WS are recognized
3498 in a Java interface according to the steps:

Field Code Changed

Field Code Changed

Deleted: ¶

Deleted: as follows

3499 For each method M in the interface, if another method P in the interface has

3500 a. a method name that is M's method name with the characters "Async" appended, and
3501 b. the same parameter signature as M, and
3502 c. a return type of Response<R> where R is the return type of M

3503 then P is a JAX-WS polling method that isn't part of the SCA interface contract.

3504 For each method M in the interface, if another method C in the interface has

3505 a. a method name that is M's method name with the characters "Async" appended, and
3506 b. a parameter signature that is M's parameter signature with an additional final parameter of
3507 type AsyncHandler<R> where R is the return type of M, and
3508 c. a return type of Future<?>

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

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

As an example, an interface can be defined in WSDL as shown in Snippet 11-1:

Deleted: follows

```
<!-- WSDL extract -->
<message name="getPrice">
 <part name="ticker" type="xsd:string"/>
</message>

<message name="getPriceResponse">
 <part name="price" type="xsd:float"/>
</message>

<portType name="StockQuote">
 <operation name="getPrice">
    <input message="tns:getPrice"/>
    <output message="tns:getPriceResponse"/>
 </operation>
</portType>
```

Snippet 11-1: Example WSDL Interface

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

Deleted: following

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

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

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

Deleted: following

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

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

SCA runtimes MUST support the use of the JAX-WS client asynchronous model. [JCA100009] If the client implementation uses the asynchronous form of the interface, the two additional getPriceAsync() methods can be used for polling and callbacks as defined by the JAX-WS specification.

Field Code Changed

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 which contains the server-side asynchronous methods defined by SCA. [JCA100010]

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

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

3558 Generating an asynchronous service method from a WSDL request/response operation follows the
3559 algorithm described in the same section.

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

12 Conformance

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

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

Deleted: For

Deleted: , the specification text is

Deleted: s

Deleted: the code artifacts

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

a) SCA Java XML Document,

b) SCA Java Class

c) SCA Runtime.

12.1 SCA Java XML Document

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

Deleted: or an SCA ConstrainingType Document

Deleted: or SCA ConstrainingType Document

12.2 SCA Java Class

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

12.3 SCA Runtime

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

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

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
     OASIS trademark, IPR and other policies apply.  -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
   elementFormDefault="qualified">

   <include schemaLocation="sca-core-1.1-cd05.xsd"/>

   <!-- Java Interface -->
   <element name="interface.java" type="sca:JavaInterface"
            substitutionGroup="sca:interface"/>
   <complexType name="JavaInterface">
      <complexContent>
         <extension base="sca:Interface">
            <sequence>
               <any namespace="##other" processContents="lax" minOccurs="0"
                    maxOccurs="unbounded"/>
            </sequence>
            <attribute name="interface" type="NCName" use="required"/>
            <attribute name="callbackInterface" type="NCName"
                       use="optional"/>
         </extension>
      </complexContent>
   </complexType>

</schema>
```

Deleted: 2009

Deleted: 200903

Deleted: 200903

Deleted: cd03

Deleted: <anyAttribute namespace="##other" processContents="lax"/>¶

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

B. Java Classes and Interfaces

B.1 SCAClient Classes and Interfaces

B.1.1 SCAClientFactory Class

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

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

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

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

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

public abstract class SCAClientFactory {

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

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

    /**
```


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

```
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
3732      * @param domainURI      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      *
```


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

```
     * @param properties    Properties that may be used when
     * creating a new instance of the SCAClient
     * @param classLoader   ClassLoader that may be used when
     * creating a new instance of the SCAClient
     * @param domainURI      URI of the target domain for the SCAClient
     * @return A new SCAClient instance
     */
    public static SCAClientFactory newInstance(Properties properties,
                                               ClassLoader classLoader,
                                               URI domainURI)
         throws NoSuchDomainException {
        final SCAClientFactoryFinder finder =
            factoryFinder != null ? factoryFinder :
                new SCAClientFactoryFinderImpl();
        final SCAClientFactory factory
            = finder.find(properties, classLoader, domainURI);
        return factory;
    }

    /**
     * Returns a reference proxy that implements the business interface <T>
     * of a service in the SCA Domain handled by this SCAClientFactory
     *
     * @param serviceURI the relative URI of the target service. Takes the
     * form componentName/serviceName.
     * Can also take the extended form componentName/serviceName/bindingName
     * to use a specific binding of the target service
     *
     * @param interfaze The business interface class of the service in the
     * domain
     * @param <T> The business interface class of the service in the domain
     *
     * @return a proxy to the target service, in the specified SCA Domain
     * that implements the business interface <B>.
     * @throws NoSuchServiceException Service requested was not found
     * @throws NoSuchDomainException Domain requested was not found
     */
    public abstract <T> T getService(Class<T> interfaze, String serviceURI)
        throws NoSuchServiceException, NoSuchDomainException;
}
```

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.
```


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

```
 */
public interface SCAClientFactoryFinder {

   /**
    * Method for finding the SCAClientFactory for a given Domain URI using
    * a specified set of properties and a a specified ClassLoader
    * @param properties - properties to use - may be null
    * @param classLoader - ClassLoader to use - may be null
    * @param domainURI - the Domain URI - must be a valid SCA Domain URI
    * @return - the SCAClientFactory or null if the factory could not be
    * @throws - NoSuchDomainException if the domainURI does not reference
    * a valid SCA Domain
    * found
    */
    SCAClientFactory find(Properties properties,
                          ClassLoader classLoader,
                          URI domainURI )
         throws NoSuchDomainException ;
}
```

B.1.3 SCAClientFactoryFinderImpl class

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

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

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

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

import org.oasisopen.sca.client.SCAClientFactoryFinder;

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

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

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


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

```
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      private static final String SCA_CLIENT_FACTORY_PROVIDER_META_INF_SERVICE
3881          = "META-INF/services/" + SCA_CLIENT_FACTORY_PROVIDER_KEY;
3882
3883      /**
3884       * Public Constructor
3885       */
3886      public SCAClientFactoryFinderImpl() {
3887      }
3888
3889      /**
3890       * Creates an instance of the SCAClientFactorySPI implementation.
3891       * This discovers the SCAClientFactorySPI Implementation and instantiates
3892       * the provider's implementation.
3893       *
3894       * @param properties    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
3900       * Implementation.
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 =
3910                  discoverProviderFactoryImplClass(properties, classLoader);
3911           final Class<? extends SCAClientFactory> factoryImplClass
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;
```


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

```
    }

    /**
     * Attempts to discover the class name for the SCAClientFactorySPI
     * implementation from the specified Properties, the System Properties
     * or the specified ClassLoader.
     *
     * @return The class name of the SCAClientFactorySPI implementation
     * @throw ServiceRuntimeException Failed to find implementation for
     * SCAClientFactorySPI.
     */
    private static String
        discoverProviderFactoryImplClass(Properties properties,
                                             ClassLoader classLoader)
        throws ServiceRuntimeException {
        String providerClassName =
          checkPropertiesForSPIClassName(properties);
        if (providerClassName != null) {
            return providerClassName;
        }

        providerClassName =
          checkPropertiesForSPIClassName(System.getProperties());
        if (providerClassName != null) {
            return providerClassName;
        }

        providerClassName = checkMETAINFServicesForSIPClassName(classLoader);
        if (providerClassName == null) {
            throw new ServiceRuntimeException(
                "Failed to find implementation for SCAClientFactory");
        }

        return providerClassName;
    }

    /**
     * Attempts to find the class name for the SCAClientFactorySPI
     * implementation from the specified Properties.
     *
     * @return The class name for the SCAClientFactorySPI implementation
     * or <code>null</code> if not found.
     */
    private static String
        checkPropertiesForSPIClassName(Properties properties) {
        if (properties == null) {
            return null;
        }

        final String providerClassName =
          properties.getProperty(SCA_CLIENT_FACTORY_PROVIDER_KEY);
        if (providerClassName != null && providerClassName.length() > 0) {
            return providerClassName;
        }

        return null;
    }

    /**
     * Attempts to find the class name for the SCAClientFactorySPI
     * implementation from the META-INF/services directory
     *
     * @return The class name for the SCAClientFactorySPI implementation or
     * <code>null</code> if not found.
```

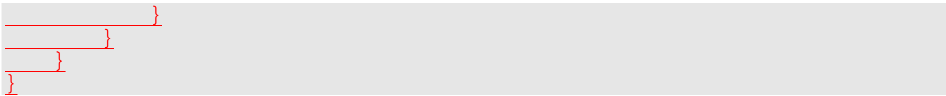

Copyright © OASIS® 2005, 2010. All Rights Reserved.

```
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         * 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         */
```


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

```
4057     private static Class<? extends SCAClientFactory>
4058         loadProviderFactoryClass(String factoryImplClassName,
4059                                  ClassLoader classLoader)
4060         throws ServiceRuntimeException {
4061
4062         try {
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
4086      * class to instantiate.
4087      * @return An instance of the SCAClientFactorySPI Implementation class
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         try {
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(
4104                "Failed to instantiate SCAClientFactory implementation class "
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                                                                         ex);
```


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



```
            }
        }
    }
}
```

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

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

- Implement their SCAClientFactory implementation class

  Vendors need to provide a subclass of SCAClientFactory that is capable of looking up Services in their SCA Runtime. Vendors need to subclass SCAClientFactory and implement the getService() method so that it creates reference proxies to services in SCA Domains handled by their SCA runtime(s).

- Configure the Vendor SCAClientFactory implementation class so that it gets used

  Vendors have several options:

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

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

  Option 2: Provide a META-INF/services file

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

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

    Vendors inject an instance of the vendor implementation of SCAClientFactoryFinder into the factoryFinder field of the SCAClientFactory abstract class. The reference implementation of SCAClientFactoryFinder is not used in this scenario. The vendor implementation of SCAClientFactoryFinder can find the vendor implementation(s) of SCAClientFactory by any means.

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

4149 # C. Conformance Items

4150 This section contains a list of conformance items for the SCA-J Common Annotations and APIs
4151 specification.

4152

Conformance ID	Description
[JCA20001]	Remotable Services MUST NOT make use of *method overloading*.
[JCA20002]	the SCA runtime MUST ensure that a stateless scoped implementation instance object is only ever dispatched on one thread at any one time.
[JCA20003]	within the SCA lifecycle of a stateless scoped implementation instance, the SCA runtime MUST only make a single invocation of one business method.
[JCA20004]	Where an implementation is used by a "domain level component", and the implementation is marked "Composite" scope, the SCA runtime MUST ensure that all consumers of the component appear to be interacting with a single runtime instance of the implementation.
[JCA20005]	When the implementation class is marked for eager initialization, the SCA runtime MUST create a composite scoped instance when its containing component is started.
[JCA20006]	If a method of an implementation class is marked with the @Init annotation, the SCA runtime MUST call that method when the implementation instance is created.
[JCA20007]	the SCA runtime MAY run multiple threads in a single composite scoped implementation instance object and the SCA runtime MUST NOT perform any synchronization.
[JCA20008]	Where an implementation is marked "Composite" scope and it is used by a component that is nested inside a composite that is used as the implementation of a higher level component, the SCA runtime MUST ensure that all consumers of the component appear to be interacting with a single runtime instance of the implementation. There can be multiple instances of the higher level component, each running on different nodes in a distributed SCA runtime.
[JCA20009]	The SCA runtime MAY use by-reference semantics when passing input parameters, return values or exceptions on calls to remotable services within the same JVM if both the service method implementation and the service proxy used by the client are marked “allows pass by reference”.
[JCA20010]	The SCA runtime MUST use by-value semantics when passing input parameters, return values and exceptions on calls to remotable services within the same JVM if the service method implementation is not marked “allows pass by reference” or the service proxy used by the client is not marked “allows pass by reference”.
[JCA30001]	The value of the @interface attribute MUST be the fully qualified name of the Java interface class
[JCA30002]	The value of the @callbackInterface attribute MUST be the fully

Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed

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

	qualified name of a Java interface used for callbacks
[JCA30003]	if the Java interface class identified by the @interface attribute does contain a Java @Callback annotation, then the Java interface class identified by the @callbackInterface attribute MUST be the same interface class.
[JCA30004]	The interface.java element MUST conform to the schema defined in the sca-interface-java.xsd schema.
[JCA30005]	The value of the @remotable attribute on the <interface.java/> element does not override the presence of a @Remotable annotation on the interface class and so if the interface class contains a @Remotable annotation and the @remotable attribute has a value of "false", then the SCA Runtime MUST raise an error and MUST NOT run the component concerned.
[JCA30006]	A Java interface referenced by the @interface attribute of an <interface.java/> element MUST NOT contain the following SCA Java annotations: @Intent, @Qualifier.
[JCA30007]	A Java interface referenced by the @interface attribute of an <interface.java/> element MUST NOT contain any of the following SCA Java annotations: @AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service.
[JCA30009]	The SCA Assembly Model specification [ASSEMBLY] defines a number of criteria that need to be satisfied in order for two interfaces to be compatible or have a compatible superset or subset relationship. If these interfaces are both Java interfaces, compatibility also means that every method that is present in both interfaces is defined consistently in both interfaces with respect to the @OneWay annotation, that is, the annotation is either present in both interfaces or absent in both interfaces.
[JCA30010]	If the identified class is annotated with either the JAX-WS @WebService or @WebServiceProvider annotations and the annotation has a non-empty ***wsdlLocation*** property, then the SCA Runtime MUST act as if an <interface.wsdl/> element is present instead of the <interface.java/> element, with an @interface attribute identifying the portType mapped from the Java interface class and containing @requires and @policySets attribute values equal to the @requires and @policySets attribute values of the <interface.java/> element.
[JCA40001]	The SCA Runtime MUST call a constructor of a component implementation.
[JCA40002]	When the constructor completes successfully, the SCA Runtime MUST transition the component implementation to the Injecting state.
[JCA40003]	When the Constructing state, the SCA Runtime MUST transition the component implementation to the Injecting state.
[JCA40004]	If an exception is in the Injecting state, the SCA Runtime MUST transition the component implementation.

Field Code Changed
Field Code Changed
Field Code Changed
Field Code Changed
Deleted: implementation class
Deleted: or the @callbackInterface
Deleted: any of
Deleted: @AllowsPassByReference, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Init,
Deleted: Property, @
Deleted: , @Reference, @Scope, @Service
Field Code Changed
Deleted: callbackInterface
Deleted: Callback, @
Field Code Changed
Deleted: A Java interface referenced by the @callbackInterface attribute of an <interface.java/> element MUST NOT contain any of the following SCA Java annotations:¶ @AllowsPassByReference, @Callback, @ComponentName, @Constructor, @Context, @Destroy, @EagerInit, @Init, @Intent, @Property, @Qualifier, @Reference, @Scope, @Service.
Field Code Changed
Deleted: The SCA Runtime MUST call a constructor of the component implementation at the start of the Constructing state.
Field Code Changed
Deleted: perform any construc ... [1]
Deleted: the
Deleted: at the start of the ... [2]
Field Code Changed
Deleted: The SCA Runtime M ... [3]
Field Code Changed
Deleted: If an exception is thr ... [4]
Deleted: constructor complete ... [5]
Deleted: Terminated
Field Code Changed
Deleted: When a component ... [6]
Deleted: thrown whilst
Deleted: Constructing
Deleted: first inject all field and ... [7]
Deleted: to the Terminated state

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

[JCA40005]	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.
[JCA40006]	When a component implementation instance is in the Injecting state, the SCA Runtime MUST inject all field and setter references that are present into the component implementation, after all the properties have been injected.
[JCA40007]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation is in the Injecting state.
[JCA40008]	The SCA Runtime MUST transition the component implementation to the Initializing state.
[JCA40009]	When the injection of properties or references, the SCA Runtime MUST transition the component implementation to the Destroying state.
[JCA40010]	If an exception is thrown whilst injecting properties or references, the SCA Runtime MUST transition the component implementation to the Destroying state.
[JCA40011]	When the component implementation enters the Initializing State, the SCA Runtime MUST call the method annotated with @Init on the component implementation, if present.
[JCA40012]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation instance is in the Initializing state.
[JCA40013]	Once the method annotated with @Init completes successfully, the SCA Runtime MUST transition the component implementation to the Running state.
[JCA40014]	If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation to the Running state.
[JCA40015]	If an exception is thrown whilst initializing, the SCA Runtime MUST transition the component implementation to the Destroying state.
[JCA40016]	When the component implementation scope ends, the SCA Runtime MUST transition the component implementation to the Destroying state.
[JCA40017]	When a component implementation scope ends, the SCA Runtime MUST transition the component implementation, if present.
[JCA40018]	When a component implementation enters the Destroying state, the SCA Runtime MUST call the method annotated with @Destroy on the component implementation, if present.
[JCA40019]	The SCA Runtime MUST NOT invoke Service methods on the component implementation when the component implementation instance is in the Destroying state.
[JCA40020]	Once the method annotated with @Destroy completes successfully, the SCA Runtime MUST transition the component implementation to the Terminated state.
[JCA40021]	If an exception is thrown whilst destroying, the SCA Runtime MUST transition the component implementation to the Terminated state.
[JCA40022]	The SCA Runtime MUST transition the component implementation to

Field Code Changed

Deleted: first …references …, after all the properties have been inj ... [8]

Field Code Changed

Deleted: The SCA Runtime MUST ensure that the correct synchronization model is used so that all injected properties and references are made visible to the component implementation without requiring the component implementation developer to do any specific synchronization.

Field Code Changed

Deleted: ensure that the correct synchronization model is used so that all injected properties and references are made visible to…without requiring…developer to do any ... [9]

Field Code Changed

Deleted: When the injection (... [10]

Field Code Changed

Deleted: If an exception is th ... [11]

Field Code Changed

Deleted: When the compone ... [12]

Field Code Changed

Deleted: If a component ... [13]

Field Code Changed

Deleted: If a component ... [14]

Field Code Changed

Deleted: The SCA Runtime M ... [15]

Field Code Changed

Deleted: Once the method ... [16]

Field Code Changed

Deleted: The SCA Runtime M ... [17]

Field Code Changed

Deleted: The SCA Runtime M ... [18]

Field Code Changed

Deleted: the …enters the De ... [19]

Field Code Changed

Deleted: If a component ... [20]

Field Code Changed

Deleted: If a component ... [21]

Field Code Changed

Deleted: The SCA Runtime M ... [22]

Field Code Changed

Deleted: Once the method ... [23]

Field Code Changed

Deleted: If an exception is th ... [24]

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

the Terminated state.

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

[JCA40024] If a property or reference is unable to be injected, the SCA Runtime MUST transition the component implementation to the Destroying state.

[JCA60001] When a bidirectional service is invoked, the SCA runtime MUST inject a callback reference for the invoking service into all fields and setter methods of the service implementation class that are marked with a @Callback annotation and typed by the callback interface of the bidirectional service, and the SCA runtime MUST inject null into all other fields and setter methods of the service implementation class that are marked with a @Callback annotation.

[JCA60002] When a non-bidirectional service is invoked, the SCA runtime MUST inject null into all fields and setter methods of the service implementation class that are marked with a @Callback annotation.

[JCA60003] The SCA asynchronous service Java interface mapping of a WSDL request-response operation MUST appear as follows:

The interface is annotated with the "asyncInvocation" intent.

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

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

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

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

[JCA60006] For the purposes of matching interfaces (when wiring between a reference and a service, or when using an implementation class by a component), an interface which has one or more methods which follow the SCA asynchronous service pattern MUST be treated as if those methods are mapped as the equivalent synchronous methods, as follows:

Asynchronous service methods are characterized by:

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

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 ... [28]

Field Code Changed

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

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

– a last input parameter with a type of ResponseDispatch<X>
– annotation with the asyncInvocation intent
– possible annotation with the @AsyncFault annotation

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

[JCA70001]	SCA identifies annotations that correspond to intents by providing an @Intent annotation which MUST be used in the definition of a specific intent annotation.
[JCA70002]	• 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.
[JCA70003]	SCA annotations (general or specific) are applied to the same Java element, the SCA runtime MUST NOT instantiate such an implementation class.
[JCA70004]	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.
[JCA70005]	• A method annotated with @Destroy MAY have any access modifier and MUST have a void return type and no arguments.
[JCA70006]	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.
[JCA80001]	
[JCA80002]	When marked for eager initialization with an @EagerInit annotation, the composite scoped instance MUST be created when its containing component is started.
[JCA80003]	When invoked during the execution of a service operation, the RequestContext.getServiceReference method MUST return a ServiceReference that represents the service that was invoked.
[JCA80004]	The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter has multiplicity greater than one.
[JCA80005]	The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the reference named by the referenceName parameter does not have an interface of the type defined by the businessInterface parameter.
[JCA80006]	The ComponentContext.getServiceReference method MUST throw an IllegalArgumentException if the component does not have a reference with the name provided in the referenceName parameter.
[JCA80007][JCA80007]	The ComponentContext.getServiceReference method MUST return null if the multiplicity of the reference named by the referenceName parameter is 0..1 and the reference has no target service configured.
[JCA80008]	The ComponentContext.getURI method MUST return the absolute URI of the component in the SCA Domain.

Field Code Changed

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

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 introspec ... [30]

Field Code Changed

Deleted: Where multiple intent

Deleted: MUST NOT be used on static methods

Deleted: on static fields. It is an error

Deleted: use an SCA annotation on a static method or a static field ... [31]

Deleted: compute the combined intents for the Java element b ... [32]

Field Code Changed

Deleted: If intent annotations are specified on both an interface ... [33]

Field Code Changed

Deleted: The @PolicySets annotation MUST NOT be app ... [34]

Field Code Changed

Deleted: If the @PolicySets annotation is specified on both ... [35]

Deleted: The ComponentContext.getServic ... [36]

Field Code Changed

Deleted: The ComponentContext.getReque ... [37]

Field Code Changed

Deleted: A method marked with the @Init annotation MAY have a ... [38]

Field Code Changed

Deleted: If there is a method annotated with @Init that matc ... [39]

Field Code Changed

Deleted: The @Property annotation MUST NOT be used on a clas ... [40]

Field Code Changed

Deleted: For a @Property annotation applied to a constr ... [41]

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

[JCA80009]	The ComponentContext.getService method MUST return the proxy object implementing the interface provided by the businessInterface parameter, for the reference named by the referenceName parameter with the interface defined by the businessInterface parameter when that reference has a target service configured.
[JCA80010]	The ComponentContext.getService method MUST return null if the multiplicity of the reference named by the referenceName parameter is 0..1 and the reference has no target service configured.
[JCA80011]	The ComponentContext.getService method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter.
[JCA80012]	The ComponentContext.getService method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.
[JCA80013]	The ComponentContext.getServiceReference method MUST return a ServiceReference object typed by the interface provided by the businessInterface parameter, for the reference named by the referenceName parameter with the interface defined by the businessInterface parameter when that reference has a target service configured.
[JCA80014]	The ComponentContext.getServices method MUST return a collection containing one proxy object implementing the interface provided by the businessInterface parameter for each of the target services configured on the reference identified by the referenceName parameter.
[JCA80015]	The ComponentContext.getServices method MUST return an empty collection if the service reference with the name supplied in the referenceName parameter is not wired to any target services.
[JCA80016]	The ComponentContext.getServices method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1.
[JCA80017]	The ComponentContext.getServices method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter.
The ComponentContext.getServices method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.[JCA80018]	The ComponentContext.getServices method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.
[JCA80019]	The ComponentContext.getServiceReferences method MUST return a collection containing one ServiceReference object typed by the interface provided by the businessInterface parameter for each of the target services configured on the reference identified by the

Field Code Changed

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

Field Code Changed

Deleted: The @Qualifier annotation MUST be used in a specific intent annotation definition where the intent has qualifiers.

Field Code Changed

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

Field Code Changed

Deleted: For a @Reference annotation applied to a constructor parameter, there is no default for the name attribute and the name attribute MUST be present.

Field Code Changed

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

Field Code Changed

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

Field Code Changed

Deleted: If the type of a reference is defined as an array or as any type that extends or implements ... [42]

Field Code Changed

Deleted: An unwired

Deleted: a multiplicity of 0..1 MUST be presented to the implemen ... [43]

Deleted: SCA runtime as null (either via injection or via API call).

Field Code Changed

Deleted: An unwired

Deleted: a multiplicity of 0..n MUST be presented to the implemen ... [44]

Deleted: SCA runtime as an empty array or empty collection (eith ... [45]

Field Code Changed

Deleted: References MAY be reinjected by an SCA runtime ... [46]

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

	referenceName parameter.
[JCA80020]	The ComponentContext.getServiceReferences method MUST return an empty collection if the service reference with the name supplied in the referenceName parameter is not wired to any target services.
[JCA80021]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the reference identified by the referenceName parameter has multiplicity of 0..1 or 1..1.
[JCA80022]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the component does not have a reference with the name supplied in the referenceName parameter.
[JCA80023]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the service reference with the name supplied in the referenceName does not have an interface compatible with the interface supplied in the businessInterface parameter.
[JCA80024]	The ComponentContext.createSelfReference method MUST return a ServiceReference object typed by the interface defined by the businessInterface parameter for one of the services of the invoking component which has the interface defined by the businessInterface parameter.
[JCA80025]	The ComponentContext.getServiceReferences method MUST throw an IllegalArgumentException if the component does not have a service which implements the interface identified by the businessInterface parameter.
[JCA80026]	The ComponentContext.createSelfReference method MUST return a ServiceReference object typed by the interface defined by the businessInterface parameter for the service identified by the serviceName of the invoking component and which has the interface defined by the businessInterface parameter.
[JCA80027]	The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the component does not have a service with the name identified by the serviceName parameter.
[JCA80028]	The ComponentContext.createSelfReference method MUST throw an IllegalArgumentException if the component service with the name identified by the serviceName parameter does not implement a business interface which is compatible with the supplied businessInterface parameter.
[JCA80029]	The ComponentContext.getProperty method MUST return an object of the type identified by the type parameter containing the value specified in the component configuration for the property named by the propertyName parameter or null if no value is specified in the configuration.
[JCA80030]	The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component does not have a property with the name identified by the propertyName parameter.
[JCA80031]	The ComponentContext.getProperty method MUST throw an IllegalArgumentException if the component property with the name identified by the propertyName parameter does not have a type which is compatible with the supplied type parameter.

Field Code Changed

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

Field Code Changed

Deleted: 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.

Field Code Changed

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.

Field Code Changed

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.

Field Code Changed

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.

Field Code Changed

Deleted: A ServiceReference that has been obtained from a refe ... [47]

Field Code Changed

Deleted: If the target of a ServiceReference has been ... [48]

Field Code Changed

Deleted: If the target of a ServiceReference has becom ... [49]

Field Code Changed

Deleted: If the target service of a ServiceReference is changed ... [50]

Field Code Changed

Deleted: A reference or ServiceReference accessed t ... [51]

Field Code Changed

Deleted: If the target of a reference or ServiceReference accesse ... [52]

Field Code Changed

Deleted: If the target service of a reference or ServiceReferenc ... [53]

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

[JCA80032]	The ComponentContext.cast method MUST return a ServiceReference object which is typed by the same business interface as specified by the reference proxy object supplied in the target parameter.
[JCA80033]	The ComponentContext.cast method MUST throw an IllegalArgumentException if the supplied target parameter is not an SCA reference proxy object.
[JCA80034]	The RequestContext.getSecuritySubject method MUST return the JAAS subject of the current request, or null if there is no subject or null if the method is invoked from code not processing a service request or callback request.
[JCA80035]	The RequestContext.getServiceName method MUST return the name of the service for which an operation is being processed, or null if invoked from a thread that is not processing a service operation or a callback operation.
[JCA80036]	The RequestContext.getCallbackReference method MUST return a ServiceReference object typed by the interface of the callback supplied by the client of the invoked service, or null if either the invoked service is not bidirectional or if the method is invoked from a thread that is not processing a service operation.
[JCA80037]	The RequestContext.getCallback method MUST return a reference proxy object typed by the interface of the callback supplied by the client of the invoked service, or null if either the invoked service is not bidirectional or if the method is invoked from a thread that is not processing a service operation.
[JCA80038]	When invoked during the execution of a callback operation, the RequestContext.getServiceReference method MUST return a ServiceReference that represents the callback that was invoked.
[JCA80039]	When invoked from a thread not involved in the execution of either a service operation or of a callback operation, the RequestContext.getServiceReference method MUST return null.
[JCA80040]	The ServiceReference.getService method MUST return a reference proxy object which can be used to invoke operations on the target service of the reference and which is typed with the business interface of the reference.
[JCA80041]	The ServiceReference.getBusinessInterface method MUST return a Class object representing the business interface of the reference.
[JCA80042]	The SCAClientFactory.newInstance(URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.
[JCA80043]	The SCAClientFactory.newInstance(URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.
[JCA80044]	The SCAClientFactory.newInstance(Properties, URI) method MUST return an object which implements the SCAClientFactory class for the SCA Domain identified by the domainURI parameter.
[JCA80045]	The SCAClientFactory.newInstance(Properties, URI) method MUST throw a NoSuchDomainException if the domainURI parameter does

Field Code Changed

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

Field Code Changed

Deleted: 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.

Field Code Changed

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

Field Code Changed

Deleted: A remotable service can be published externally as a service and MUST be translatable into a WSDL portType.

Field Code Changed

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

Field Code Changed

Deleted: An implementation class need not be declared as implementing all of the interfaces implied by the services declared in its @Service annotation, but all n ... [54]

Field Code Changed

Deleted: A @Service annota ... [55]

Field Code Changed

Deleted: A @Service annota ... [56]

Field Code Changed

Deleted: A component ... [57]

Field Code Changed

Deleted: When used to anno ... [58]

Field Code Changed

Deleted: For a @Property ... [59]

Field Code Changed

Deleted: If the name attribute ... [60]

Field Code Changed

Deleted: If the names attribu ... [61]

Field Code Changed

Deleted: The number of Strin ... [62]

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

	not identify a valid SCA Domain.
[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.
[JCA80047]	The SCAClientFactory.newInstance(Classloader, URI) method MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.
[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.
[JCA80049]	The SCAClientFactory.newInstance(Properties, Classloader, URI) MUST throw a NoSuchDomainException if the domainURI parameter does not identify a valid SCA Domain.
[JCA80050]	The SCAClientFactory.getService method MUST return a proxy object which implements the business interface defined by the interfaze parameter and which can be used to invoke operations on the service identified by the serviceURI parameter.
[JCA80051]	The SCAClientFactory.getService method MUST throw a NoSuchServiceException if a service with the relative URI serviceURI and a business interface which matches interfaze cannot be found in the SCA Domain targeted by the SCAClient object.
[JCA80052]	The SCAClientFactory.getService method MUST throw a NoSuchServiceException if the domainURI of the SCAClientFactory does not identify a valid SCA Domain.
[JCA80053]	The SCAClientFactory.getDomainURI method MUST return the SCA Domain URI of the Domain associated with the SCAClientFactory object.
[JCA80054]	The SCAClientFactory.getDomainURI method MUST throw a ***NoSuchServiceException*** if the domainURI of the SCAClientFactory does not identify a valid SCA Domain.
The implementation of the SCAClientFactoryFinder.find method MUST return an object which is an implementation of the SCAClientFactory interface, for the SCA Domain represented by the doaminURI parameter, using the supplied properties and classloader. [JCA80055]	The implementation of the SCAClientFactoryFinder.find method MUST return an object which is an implementation of the SCAClientFactory interface, for the SCA Domain represented by the doaminURI parameter, using the supplied properties and classloader.
[JCA80056]	The implementation of the SCAClientFactoryFinder.find method MUST throw a ServiceRuntimeException if the SCAClientFactory implementation could not be found.
[JCA50057]	The ResponseDispatch.sendResponse() method MUST send the response message to the client of an asynchronous service.
[JCA80058]	The ResponseDispatch.sendResponse() method MUST throw an InvalidStateException if either the sendResponse method or the

Field Code Changed

Deleted: The @Service annotation MUST NOT specify Void.class in conjunction with any other service class or interface.

Field Code Changed

Deleted: The @AllowsPassByReference annotation MAY be placed on an individual method of a remotable service implementation, on a service implementation class, or on an individual reference for a remotable service. When applied to a reference, it MAY appear anywhere that the @Remotable annotation MAY appear. It MUST NOT appear anywhere else.

Field Code Changed

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

Field Code Changed

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

Field Code Changed

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

Field Code Changed

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

Field Code Changed

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

Field Code Changed

Deleted: For SCA service interfaces defined using interface.java, the Java interface MUST NOT contain ... [63]

Field Code Changed

Deleted: For SCA reference interfaces defined using ... [64]

Field Code Changed

Deleted: If the additional client-side asynchronous polling and call ... [65]

Field Code Changed

Deleted: SCA runtimes MUST support the use of the JAX-W ... [66]

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

	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

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

	required attribute MUST have the value true.
[JCA90020]	If the type of a reference is not an array or any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity= 0..1 if the @Reference annotation required attribute is false and with @multiplicity=1..1 if the @Reference annotation required attribute is true.
[JCA90021]	If the type of a reference is defined as an array or as any type that extends or implements java.util.Collection, then the SCA runtime MUST introspect the component type of the implementation with a <reference/> element with @multiplicity=0..n if the @Reference annotation required attribute is false and with @multiplicity=1..n if the @Reference annotation required attribute is true.
[JCA90022]	An unwired reference with a multiplicity of 0..1 MUST be presented to the implementation code by the SCA runtime as null (either via injection or via API call).
[JCA90023]	An unwired reference with a multiplicity of 0..n MUST be presented to the implementation code by the SCA runtime as an empty array or empty collection (either via injection or via API call).
[JCA90024]	References MAY be reinjected by an SCA runtime after the initial creation of a component if the reference target changes due to a change in wiring that has occurred since the component was initialized.
[JCA90025]	In order for reinjection to occur, the following MUST be true: 1. The component MUST NOT be STATELESS scoped. 2. The reference MUST use either field-based injection or setter injection. References that are injected through constructor injection MUST NOT be changed.
[JCA90026]	If a reference target changes and the reference is not reinjected, the reference MUST continue to work as if the reference target was not changed.
[JCA90027]	If an operation is called on a reference where the target of that reference has been undeployed, the SCA runtime SHOULD throw an InvalidServiceException.
[JCA90028]	If an operation is called on a reference where the target of the reference has become unavailable for some reason, the SCA runtime SHOULD throw a ServiceUnavailableException.
[JCA90029]	If the target service of the reference is changed, the reference 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.

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

[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 0..n or multiplicity 1..n MUST NOT change its contents when changes occur to the reference wiring or to the targets of the wiring.
[JCA90038]	In cases where the contents of a reference array or collection change when the wiring changes or the targets change, then for references that use setter injection, the setter method MUST be called by the SCA runtime for any change to the contents.
[JCA90039]	A reinjected array or Collection for a reference MUST NOT be the same array or Collection object previously injected to the component.
[JCA90040]	A remotable service can be published externally as a service and MUST be translatable into a WSDL portType.
[JCA90041]	The @Scope annotation MUST only be used on a service's implementation class. It is an error to use this annotation on an interface.
[JCA90042]	An implementation class need not be declared as implementing all of the interfaces implied by the services declared in its @Service annotation, but all methods of all the declared service interfaces MUST be present.
[JCA90045]	If a component implementation has two services with the same Java simple name, the names attribute of the @Service annotation MUST be specified.
[JCA90046]	When used to annotate a method or a field of an implementation class for injection of a callback object, the@Callback annotation MUST NOT specify any attributes.
[JCA90047]	For a @Property annotation, if the type of the Java class field or the type of the input parameter of the setter method or constructor is defined as an array or as any type that extends or implements

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

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

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

[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.

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

[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

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

4154 # D. Acknowledgements

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

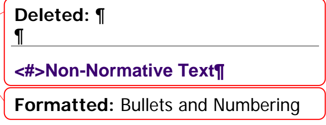
4157 **Participants:**

Participant Name	Affiliation
Bryan Aupperle	IBM
Ron Barack	SAP AG*
Michael Beisiegel	IBM
Henning Blohm	SAP AG*
David Booz	IBM
Martin Chapman	Oracle Corporation
Graham Charters	IBM
Shih-Chang Chen	Oracle Corporation
Chris Cheng	Primeton Technologies, Inc.
Vamsavardhana Reddy Chillakuru	IBM
Roberto Chinnici	Sun Microsystems
Pyounguk Cho	Oracle Corporation
Eric Clairambault	IBM
Mark Combellack	Avaya, Inc.
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
Raymond Feng	IBM
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Anish Karmarkar	Oracle Corporation
Michael Keith	Oracle Corporation
Rainer Kerth	SAP AG*
Meeraj Kunnumpurath	Individual
Simon Laws	IBM
Yang Lei	IBM
Mark Little	Red Hat
Ashok Malhotra	Oracle Corporation
Jim Marino	Individual
Jeff Mischkinsky	Oracle Corporation
Sriram Narasimhan	TIBCO Software Inc.
Simon Nash	Individual
Sanjay Patil	SAP AG*
Plamen Pavlov	SAP AG*
Peter Peshev	SAP AG*
Ramkumar Ramalingam	IBM
Luciano Resende	IBM
Michael Rowley	Active Endpoints, Inc.
Vladimir Savchenko	SAP AG*
Pradeep Simha	TIBCO Software Inc.
Raghav Srinivasan	Oracle Corporation

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

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

4158



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

4159

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

4160 # E. Revision History

4161 [optional; should not be included in OASIS Standards]

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.

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

			All comments removed.
cd02-rev1	2009-02-03	Mike Edwards	Issues 25+95 Issue 120
cd02-rev2	2009-02-08	Mike Edwards	Merge annotation definitions contained in section 10 into section 8 Move remaining parts of section 10 to section 7. Accept all changes.
cd02-rev3	2009-03-16	Mike Edwards	Issue 104 - RFC2119 work and formal marking of all normative statements - all sections - Completion of Appendix B (list of all normative statements) Accept all changes
cd02-rev4	2009-03-20	Mike Edwards	Editorially removed sentence about componentType side files in Section1 Editorially changed package name to org.oasisopen from org.osoa in lines 291, 292 Issue 6 - add Section 2.3, modify section 9.1 Issue 30 - Section 2.2.2 Issue 76 - Section 6.2.4 Issue 27 - Section 7.6.2, 7.6.2.1 Issue 77 - Section 1.2 Issue 102 - Section 9.21 Issue 123 - conersations removed Issue 65 - Added a new Section 4 ** Causes renumbering of later sections ** ** NB new numbering is used below ** Issue 119 - Added a new section 12 Issue 125 - Section 3.1 Issue 130 - (new number) Section 8.6.2.1 Issue 132 - Section 1 Issue 133 - Section 10.15, Section 10.17 Issue 134 - Section 10.3, Section 10.18 Issue 135 - Section 10.21 Issue 138 - Section 11 Issue 141 - Section 9.1 Issue 142 - Section 10.17.1
cd02-rev5	2009-04-20	Mike Edwards	Issue 154 - Appendix A Issue 129 - Section 8.3.1.1
cd02-rev6	2009-04-28	Mike Edwards	Issue 148 - Section 3 Issue 98 - Section 8
cd02-rev7	2009-04-30	Mike Edwards	Editorial cleanup throughout the spec

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

cd02-rev8	2009-05-01	Mike Edwards	Further extensive editorial cleanup throughout the spec Issue 160 - Section 8.6.2 & 8.6.2.1 removed
cd02-rev8a	2009-05-03	Simon Nash	Minor editorial cleanup
cd03	2009-05-04	Anish Karmarkar	Updated references and front page clean up
cd03-rev1	2009-09-15	David Booz	Applied Issues: 1,13,125,131,156,157,158,159,161,165,172,177
cd03-rev2	2010-01-19	David Booz	Updated to current Assembly namespace Applied issues: 127,155,168,181,184,185,187,189,190,194
cd03-rev3	2010-02-01	Mike Edwards	Applied issue 54. Editorial updates to code samples.
cd03-rev4	2010-02-05	Bryan Aupperle, Dave Booz	Editorial update for OASIS formatting
CD04	2010-02-06	Dave Booz	Editorial updates for Committee Draft 04 All changes accepted

4163

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

Page 107: [1] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

perform any constructor reference or property 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 reference or property injection when it calls the constructor of a component implementation.

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

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 model is used so that all injected 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 synchronization.

Page 108: [10] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

When the injection of properties and references 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 in the Injecting

Page 108: [11] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

If an exception is thrown whilst 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**

Initializing

Page 108: [12] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

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

Page 108: [13] 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 not yet been initialized, the SCA Runtime MUST throw a ServiceUnavailableException.

Page 108: [14] 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 not yet been initialized, the SCA Runtime MUST throw a ServiceUnavailableException.

Page 108: [15] 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 Initializing state.

Page 108: [16] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

Once the method annotated with @Init completes successfully

Page 108: [16] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

Destroying

Page 108: [17] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

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

Page 108: [18] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

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

Page 108: [19] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

the

Page 108: [19] Deleted **Mike Edwards** **2/16/2010 10:10:00 AM**

enters the Destroying state

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	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: [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

Page 108: [24] Deleted	Mike Edwards	2/16/2010 10:10:00 AM

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 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.