



# Service Component Architecture Client and Implementation Model for C++ Specification Version 1.1

Committee Draft 04 / Public Review Draft 02

21 January 2010

## Specification URIs:

### This Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd04.html>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd04.doc>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd04.pdf> (Authoritative)

### Previous Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd03.html>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd03.doc>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd03.pdf> (Authoritative)

### Latest Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec.html>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec.doc>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec.pdf> (Authoritative)

## Technical Committee:

OASIS Service Component Architecture / C and C++ (SCA-C-C++) TC

## Chair:

Bryan Aupperle, IBM

## Editors:

Bryan Aupperle, IBM  
David Haney  
Pete Robbins, IBM

## Related work:

This specification replaces or supercedes:

- [OSOA SCA C++ Client and Implementation V1.00](#)

This specification is related to:

- [OASIS Service Component Architecture Assembly Model Version 1.1](#)
- [OASIS SCA Policy Framework Version 1.1](#)
- [OASIS Service Component Architecture Web Service Binding Specification Version 1.1](#)

## Declared XML Namespace(s):

<http://docs.oasis-open.org/ns/opencsa/sca/200903200912>  
<http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901>

## Abstract:

This document describes the SCA Client and Implementation Model for the C++ programming language.

The SCA C++ implementation model describes how to implement SCA components in C++. A component implementation itself can also be a client to other services provided by other components or external services. The document describes how a C++ implemented component gets access to services and calls their operations.

The document also explains how non-SCA C++ components can be clients to services provided by other components or external services. The document shows how those non-SCA C++ component implementations access services and call their operations.

**Status:**

This document was last revised or approved by the Service Component Architecture / C and C++ 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-c-cpp/>.

For information on whether any patents have been disclosed that may 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-c-cpp/ipr.php>).

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

---

## Notices

Copyright © OASIS® 2006, ~~2009~~2010. All Rights Reserved.

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

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

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

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

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

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

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

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

---

# Table of Contents

1	Introduction.....	8
1.1	Terminology.....	8
1.2	Normative References.....	8
1.3	Conventions.....	9
1.3.1	Naming Conventions.....	9
1.3.2	Typographic Conventions.....	9
2	Basic Component Implementation Model.....	10
2.1	Implementing a Service.....	10
2.1.1	Implementing a Remotable Service.....	12
2.1.2	AllowsPassByReference.....	12
2.1.3	Implementing a Local Service.....	13
2.2	Component Implementation Scopes.....	13
2.2.1	Stateless Scope.....	13
2.2.2	Composite Scope.....	14
2.3	Implementing a Configuration Property.....	14
2.4	Component Type and Component.....	14
2.4.1	Interface.cpp.....	16
2.4.2	Function and CallbackFunction.....	17
2.4.3	Implementation.cpp.....	18
2.4.4	Implementation Function.....	19
2.5	Instantiation.....	20
3	Basic Client Model.....	21
3.1	Accessing Services from Component Implementations.....	21
3.2	Interface Proxies.....	22
3.3	Accessing Services from non-SCA Component Implementations.....	24
3.4	Calling Service Operations.....	24
3.5	Long Running Request-Response Operations.....	25
3.5.1	Response Callback.....	26
3.5.2	Response Polling.....	27
3.5.3	Synchronous Response Access.....	28
3.5.4	Response Class.....	28
4	Asynchronous Programming.....	31
4.1	Non-blocking Calls.....	31
4.2	Callbacks.....	31
4.2.1	Using Callbacks.....	32
4.2.2	Callback Instance Management.....	34
4.2.3	Implementing Multiple Bidirectional Interfaces.....	34
5	Error Handling.....	35
6	C++ API.....	36
6.1	Reference Counting Pointers.....	36
6.1.1	operator*.....	36
6.1.2	operator->.....	37
6.1.3	operator void*.....	37

6.1.4	operator!	37
6.1.5	constCast	37
6.1.6	dynamicCast	38
6.1.7	reinterpretCast	38
6.1.8	staticCast	38
6.2	Component Context	38
6.2.1	getCurrent	39
6.2.2	getURI	39
6.2.3	getService	39
6.2.4	getServices	40
6.2.5	getServiceReference	40
6.2.6	getServiceReferences	40
6.2.7	getProperties	40
6.2.8	getDataFactory	41
6.2.9	getSelfReference	41
6.3	ServiceReference	41
	The detailed description of the usage of these member functions is described in	42
6.3.1	getService	42
6.3.2	getCallback	42
6.4	DomainContext	42
6.4.1	getService	43
6.5	SCAException	43
6.5.1	getEClassName	43
6.5.2	getMessageText	43
6.5.3	getFileName	44
6.5.4	getLineNumber	44
6.5.5	getFunctionName	44
6.6	SCANullPointerException	44
6.7	ServiceRuntimeException	45
6.8	ServiceUnavailableException	45
6.9	MultipleServicesException	45
7	C++ Contributions	46
7.1	Executable files	46
7.1.1	Executable in contribution	46
7.1.2	Executable shared with other contribution(s) (Export)	46
7.1.3	Executable outside of contribution (Import)	47
7.2	componentType files	48
7.3	C++ Contribution Extensions	49
7.3.1	Export.cpp	49
7.3.2	Import.cpp	49
8	C++ Interfaces	50
8.1	Types Supported in Service Interfaces	50
8.1.1	Local Service	50
8.1.2	Remotable Service	50
8.2	Header Files	51

9	WSDL to C++ and C++ to WSDL Mapping .....	53
9.1	Augmentations for WSDL to C++ Mapping .....	53
9.1.1	Mapping WSDL targetNamespace to a C++ namespace .....	54
9.1.2	Mapping WSDL Faults to C++ Exceptions .....	54
9.1.3	Mapping of in, out, in/out parts to C++ member function parameters .....	54
9.2	Augmentations for C++ to WSDL Mapping .....	55
9.2.1	Mapping C++ namespaces to WSDL namespaces .....	55
9.2.2	Parameter and return type classification .....	55
9.2.3	C++ to WSDL Type Conversion .....	55
9.2.4	Service-specific Exceptions .....	55
9.3	SDO Data Binding .....	55
9.3.1	Simple Content Binding .....	55
9.3.2	Complex Content Binding .....	58
10	Conformance .....	59
10.1	Conformance Targets .....	59
10.2	SCA Implementations .....	59
10.3	SCA Documents .....	60
10.4	C++ Files .....	60
10.5	WSDL Files .....	60
A	C++ SCA Annotations .....	61
A.1	Application of Annotations to C++ Program Elements .....	61
A.2	Interface Header Annotations .....	62
A.2.1	@Interface .....	62
A.2.2	@Remotable .....	62
A.2.3	@Callback .....	63
A.2.4	@OneWay .....	63
A.2.5	@Function .....	64
A.3	Implementation Header Annotations .....	65
A.3.1	@ComponentType .....	65
A.3.2	@Scope .....	66
A.3.3	@EagerInit .....	66
A.3.4	@AllowsPassByReference .....	67
A.3.5	@Property .....	67
A.3.6	@Reference .....	68
A.4	Base Annotation Grammar .....	69
B	C++ SCA Policy Annotations .....	70
B.1	General Intent Annotations .....	70
B.2	Specific Intent Annotations .....	72
B.2.1	Security Interaction .....	73
B.2.2	Security Implementation .....	73
B.2.3	Reliable Messaging .....	73
B.2.4	Transactions .....	74
B.2.5	Miscellaneous .....	74
B.3	Policy Set Annotations .....	76
B.4	Policy Annotation Grammar Additions .....	77

B.5	Annotation Constants .....	77
C	C++ WSDL Mapping Annotations .....	78
C.1	Interface Header Annotations .....	78
C.1.1	@WebService .....	78
C.1.2	@WebFunction .....	79
C.1.3	@OneWay .....	81
C.1.4	@WebParam .....	82
C.1.5	@WebResult .....	85
C.1.6	@SOAPBinding .....	87
C.1.7	@WebFault .....	88
C.1.8	@WebThrows .....	90
D	WSDL C++ Mapping Extensions .....	91
D.1	<cpp:bindings> .....	91
D.2	<cpp:class> .....	91
D.3	<cpp:enableWrapperStyle> .....	92
D.4	<cpp:namespace> .....	93
D.5	<cpp:memberFunction> .....	94
D.6	<cpp:parameter> .....	96
D.7	JAX-WS WSDL Extensions .....	98
D.8	sca-wsdlex-1.1.xsd .....	98
E	XML Schemas .....	100
E.1	sca-interface-cpp-1.1.xsd .....	100
E.2	sca-implementation-cpp-1.1.xsd .....	101
E.3	sca-contribution-cpp-1.1.xsd .....	102
F	Normative Statement Summary .....	104
F.1	Annotation Normative Statement Summary .....	108
F.2	WSDL Extension Normative Statement Summary .....	109
F.3	JAX-WS Normative Statements .....	109
F.3.1	Ignored Normative Statements .....	114
G	Migration .....	117
G.1	Method child elements of interface.cpp and implementation.cpp .....	117
H	Acknowledgements .....	118
I	Revision History .....	119

# 1 Introduction

This document describes the SCA Client and Implementation Model for the C++ programming language.

The SCA C++ implementation model describes how to implement SCA components in C++. A component implementation itself can also be a client to other services provided by other components or external services. The document describes how a C++ implemented component gets access to services and calls their operations.

The document also explains how non-SCA C++ components can be clients to services provided by other components or external services. The document shows how those non-SCA C++ component implementations access services and call their operations.

## 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]

This specification uses predefined namespace prefixes throughout; they are given in the following list. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

**Table 1-1 Prefixes and Namespaces used in this specification**

Prefix	Namespace	Notes
xs	"http://www.w3.org/2001/XMLSchema"	Defined by XML Schema 1.0 specification
sca	"http://docs.oasis-open.org/ns/opencsa/sca/200903-200912"	Defined by the SCA specifications
cpp	"http://docs.oasis-open.org/ns/opencsa/sca-cpp/cpp/200901"	

**Table 1-1: Prefixes and Namespaces used in this Specification**

## 1.2 Normative References

- [RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>
- [ASSEMBLY]** OASIS Committee Draft ~~0305~~, *Service Component Architecture Assembly Model Specification Version 1.1*, ~~March 2009~~ **January 2010**. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-ed03cd05.pdf>
- [POLICY]** OASIS Committee Draft 02, *SCA Policy Framework Version 1.1*, March 2009. <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf>
- [SDO21]** OSOA, *Service Data Objects For C++ Specification*, December 2006. <http://www.osoa.org/download/attachments/36/Cpp-SDO-Spec-v2.1.0-FINAL.pdf>
- [WSDL11]** World Wide Web Consortium, *Web Service Description Language (WSDL)*, March 2001. <http://www.w3.org/TR/wsdl>
- [XSD]** World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*, October 2004. <http://www.w3.org/TR/xmlschema-2/>



36 [JAXWS21] Doug. Kohlert and Arun Gupta, *The Java API for XML-Based Web Services*  
37 (*JAX-WS*) 2.1, JSR, JCP, May 2007.  
38 <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>

## 39 1.3 Conventions

### 40 1.3.1 Naming Conventions

41 This specification follows ~~some~~-naming conventions for artifacts defined by the specification, ~~as follows~~:

- 42 • For the names of elements and the names of attributes within XSD files, the names follow the  
43 CamelCase convention, with all names starting with a lower case letter.  
44 e.g. <element name="componentType" type="sca:ComponentType"/>
- 45 • For the names of types within XSD files, the names follow the CamelCase convention with all names  
46 starting with an upper case letter  
47 e.g. <complexType name="ComponentService">
- 48 • For the names of intents, the names follow the CamelCase convention, with all names starting with a  
49 lower case letter, EXCEPT for cases where the intent represents an established acronym, in which  
50 case the entire name is in upper case.  
51 An example of an intent which is an acronym is the "SOAP" intent.

### 52 1.3.2 Typographic Conventions

53 This specification follows ~~some~~-typographic conventions for ~~some~~-specific constructs:

- 54 • Conformance points are highlighted, [numbered] and cross-referenced to Normative Statement  
55 Summary
- 56 • XML attributes are identified in text as @attribute
- 57 • Language identifiers used in text are in `courier`
- 58 • Literals in text are in *italics*

---

## 2 Basic Component Implementation Model

60 This section describes how SCA components are implemented using the C++ programming language. It  
61 shows how a C++ implementation based component can implement a local or remotable service, and  
62 how the implementation can be made configurable through properties.

64 A component implementation can itself be a client of services. This aspect of a component  
65 implementation is described in the basic client model section.

### 2.1 Implementing a Service

67 A component implementation based on a C++ class (a **C++ implementation**) provides one or more  
68 services.

70 A service provided by a C++ implementation has an interface (a **service interface**) which is defined using  
71 one of:

- 72 • a C++ abstract base class
- 73 • a WSDL 1.1 portType [**WSDL11**]

74 An abstract base class is a class which has only pure virtual member functions. A C++ implementation  
75 **MUST implement all of the operation(s) of the service interface(s) of its componentType.** [**CPP20001**]

77 ~~The following snippets~~ [Snippet 2-1](#) – [Snippet 2-3](#) show ~~the~~ C++ service interface and the C++  
78 implementation class of a C++ implementation.

80 ~~Service interface.~~

```
82 // LoanService interface  
83 class LoanService {  
84 public:  
85     virtual bool approveLoan(unsigned long customerNumber,  
86                             unsigned long loanAmount) = 0;  
87 };
```

88 ~~Implementation declaration header file.~~

90 ~~Snippet 2-1: A C++ Service Interface~~

```
92 class LoanServiceImpl : public LoanService {  
93 public:  
94     LoanServiceImpl();  
95     virtual ~LoanServiceImpl();  
96  
97     virtual bool approveLoan(unsigned long customerNumber,  
98                             unsigned long loanAmount);  
99 };
```

102 ~~Snippet 2-2: C++ Service Implementation- Declaration~~

```

104 #include "LoanServiceImpl.h"
105
106 LoanServiceImpl::LoanServiceImpl()
107 {
108     ...
109 }
110
111 LoanServiceImpl::~LoanServiceImpl()
112 {
113     ...
114 }
115
116 bool LoanServiceImpl::approveLoan(unsigned long customerNumber,
117                                 unsigned long loanAmount)
118 {
119     ...
120 }

```

*Snippet 2-3: C++ Service Implementation*

The following snippet shows the component type for this component implementation.

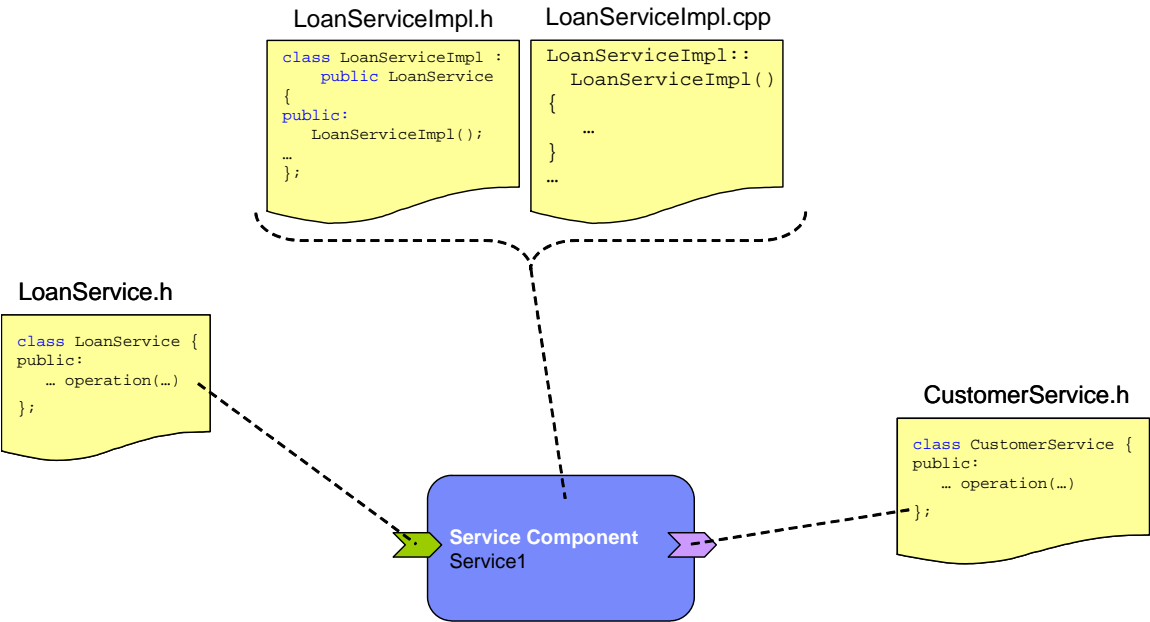
```

125 <?xml version="1.0" encoding="ASCII"?>
126 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912">
127   <service name="LoanService">
128     <interface.cpp header="LoanService.h"/>
129   </service>
130 </componentType>

```

*The following picture Snippet 2-4: Component Type for Service Implementation in Snippet 2-3*

Figure 2-1 shows the relationship between the C++ header files and implementation files for a component that has a single service and a single reference.



*Figure 2-1: Relationship of C++ Implementation Artifacts*

## 139 2.1.1 Implementing a Remotable Service

140 A `@remotable="true"` attribute on an `interface.cpp` element indicates that the interface is **remotable** as  
141 described in the Assembly Specification [ASSEMBLY]. [The following snippet Snippet 2\\_5](#) shows the  
142 component type for a remotable service:

143

```
144 <?xml version="1.0" encoding="ASCII"?>  
145 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912">  
146   <service name="LoanService">  
147     <interface.cpp header="LoanService.h" remotable="true"/>  
148   </service>  
149 </componentType>
```

150 [Snippet 2-5: ComponentType for a Remotable Service](#)

## 151 2.1.2 AllowsPassByReference

152 Calls to remotable services have by-value semantics. This means that input parameters passed to the  
153 service can be modified by the service without these modifications being visible to the client. Similarly, the  
154 return value or exception from the service can be modified by the client without these modifications being  
155 visible to the service implementation. For remote calls (either cross-machine or cross-process), these  
156 semantics are a consequence of marshalling input parameters, return values and exceptions “on the wire”  
157 and unmarshalling them “off the wire” which results in physical copies being made. For local calls within  
158 the same operating system address space, C++ calling semantics include by-reference and therefore do  
159 not provide the correct by-value semantics for SCA remotable interfaces. To compensate for this, the  
160 SCA runtime can intervene in these calls to provide by-value semantics by making copies of any by-  
161 reference values passed.

162

163 The cost of such copying can be very high relative to the cost of making a local call, especially if the data  
164 being passed is large. Also, in many cases this copying is not needed if the implementation observes  
165 certain conventions for how input parameters, return values and exceptions are used. An  
166 `@allowsPassByReference="true"` attribute allows implementations to indicate that they use input  
167 parameters, return values and exceptions in a manner that allows the SCA runtime to avoid the cost of  
168 copying by-reference values when a remotable service is called locally within the same operating system  
169 address space. See `Implementation.cpp` and `Implementation Function` for a description of the  
170 `@allowsPassByReference` attribute and how it is used.

### 171 2.1.2.1 Marking services and references as “allows pass by reference”

172 Marking a service member function implementation as “allows pass by reference” asserts that the  
173 member function implementation observes the following restrictions:

- 174 • Member function execution will not modify any input parameter before the member function returns.
- 175 • The service implementation will not retain a reference or pointer to any by-reference input parameter,  
176 return value or exception after the member function returns.
- 177 • The member function will observe “allows pass by value” client semantics [\(see below\), as defined in](#)  
178 [the next paragraph](#) for any callbacks that it makes.

179

180 Marking a client as “allows pass by reference” asserts that [for all reference’s member functions](#), the client  
181 [observeobserves](#) the [following](#) restrictions [for all references’ member functions](#):

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

### 187 2.1.2.2 Using “allows pass by reference” to optimize remotable calls

188 The SCA runtime MAY use by-reference semantics when passing input parameters, return values or  
189 exceptions on calls to remotable services within the same system address space if both the service  
190 member function implementation and the client are marked “allows pass by reference”. [CPP20014]

191  
192 The SCA runtime MUST use by-value semantics when passing input parameters, return values and  
193 exceptions on calls to remotable services within the same system address space if the service member  
194 function implementation is not marked “allows pass by reference” or the client is not marked “allows pass  
195 by reference”. [CPP20015]

### 196 2.1.3 Implementing a Local Service

197 A service interface not marked as remotable is **local**.

## 198 2.2 Component Implementation Scopes

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

203  
204 Scopes are specified using the `@scope` attribute of the `implementation.cpp` element.

205  
206 When a scope is not specified on an implementation class, the SCA runtime will interpret the  
207 implementation scope as **stateless**.

208  
209 An SCA runtime MUST support these scopes; **stateless** and **composite**. Additional scopes MAY be  
210 provided by SCA runtimes. [CPP20003]

211  
212 ~~The following snippet~~ Snippet 2-6 shows the component type for a composite scoped component:

```
213  
214 <component name="LoanService">  
215   <implementation.cpp library="loan" class="LoanServiceImpl"  
216     scope="composite" />  
217 </component>
```

### 218 ~~1.1.1 Stateless scope~~

219 ~~Snippet 2-6: Component Type for a Composite Scoped Component~~

220  
221 ~~Independent of scope, component implementations have to manage any state maintained in global~~  
222 ~~variables or static data members. A library can be loaded as early as when any component implemented~~  
223 ~~by the library enters the running state [ASSEMBLY] but no later than the first member function invocation~~  
224 ~~of a service provided by a component implemented by the library. Component implementations can not~~  
225 ~~make any assumptions about when a library might be unloaded. An SCA runtime MUST NOT perform~~  
226 ~~any synchronization of access to component implementations. [CPP20018]~~

### 227 2.2.1 Stateless Scope

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

230

231 The concurrency model for the stateless scope is single threaded. An SCA runtime MUST ensure that a  
232 stateless scoped implementation instance object is only ever dispatched on one thread at any one time.  
233 In addition, within the SCA lifecycle of an instance, an SCA runtime MUST only make a single invocation  
234 of one business member function. [CPP20012]

### 235 **2.2.12.2.2 Composite scopeScope**

236 All service requests are dispatched to the same implementation instance for the lifetime of the composite  
237 containing ~~composite~~the component. The lifetime of the containing composite is defined as the time it  
238 ~~becomes active~~placed in the runtimerunning state to the time it is ~~deactivated~~removed from the running  
239 state, either normally or abnormally.

240

241 A composite scoped implementation can also specify eager initialization using the `@eagerInit="true"`  
242 attribute on the `implementation.cpp` element of a component definition. When marked for eager  
243 initialization, the composite-scoped implementation instance will be created when ~~its containing the~~  
244 component is ~~started~~placed in running state, otherwise, initialization is lazy and the instance will be  
245 created when the first client request is received.

246

247 The concurrency model for the composite scope is multi-threaded. An SCA runtime MAY run multiple  
248 threads in a single composite scoped implementation instance object. [CPP20013]

## 249 **2.3 Implementing a Configuration Property**

250 Component implementations can be configured through properties. The properties and their types (not  
251 their values) are defined in the component type file. The C++ component can retrieve the properties using  
252 the `getProperties()` on the `ComponentContext` class.

253

254 The following code extractSnippet 2-7 shows how to get the property values.

255

```
256 #include "ComponentContext.h"  
257 using namespace oasis::sca;  
258  
259 void clientFunction()  
260 {  
261     ...  
262  
263     ComponentContextComponentContextPtr context =  
264     ComponentContext::getCurrent();  
265  
266     DataObjectPtr properties = context-->getProperties();  
267  
268     long loanRating = properties->getInteger("maxLoanValue");  
269  
270     ...  
271 }
```

272 Snippet 2-7: Retrieving Property Values

## 273 **2.4 Component Type and Component**

274 For a C++ component implementation, a component type is specified in a side file. By default, the  
275 `componentType` side file is in the root directory of the composite containing the component or some  
276 subdirectory of the composite root directory with a name matching the implementation class of the  
277 component. The location can be modified as described below in `Implementation.cpp`.

278

279 This Client and Implementation Model for C++ extends the SCA Assembly model **[ASSEMBLY]** providing  
280 support for the C++ interface type system and support for the C++ implementation type.

281  
282 ~~The following snippets~~[Snippet 2-8 – Snippet 2-10](#) show ~~the~~ C++ service interface and the C++  
283 implementation class of a C++ service.

```
284  
285 // LoanService interface  
286 class LoanService {  
287 public:  
288     virtual bool approveLoan(unsigned long customerNumber,  
289                             unsigned long loanAmount) = 0;  
290 };
```

291  
292 ~~Implementation declaration header file.~~

293 ~~Snippet 2-8: A C++ Service Interface~~

```
294  
295 class LoanServiceImpl : public LoanService {  
296 public:  
297     LoanServiceImpl();  
298     virtual ~LoanServiceImpl();  
299  
300     virtual bool approveLoan(unsigned long customerNumber,  
301                             unsigned long loanAmount);  
302 };
```

303  
304 ~~Implementation.~~

305 ~~Snippet 2-9: C++ Service Implementation Declaration~~

```
306  
307 #include "LoanServiceImpl.h"  
308  
309 // Construction/Destruction  
310  
311 LoanServiceImpl::LoanServiceImpl()  
312 {  
313     ...  
314 }  
315 LoanServiceImpl::~LoanServiceImpl()  
316 {  
317     ...  
318 }  
319 // Implementation  
320  
321 bool LoanServiceImpl::approveLoan(unsigned long customerNumber,  
322                                  unsigned long loanAmount)  
323 {  
324     ...  
325 }
```

326  
327 ~~The following snippet~~[Snippet 2-10: C++ Service Implementation](#)

328  
329 [Snippet 2-11](#) shows the component type for this component implementation.

```
330  
331 <?xml version="1.0" encoding="ASCII"?>
```

```

332 | <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912" >
333 |   <service name="LoanService">
334 |     <interface.cpp header="LoanService.h" />
335 |   </service>
336 | </componentType>

```

337

338 | *The following snippet Snippet 2-11: Component Type for Service Implementation in Snippet 2-10*

339

340 | Snippet 2-12 shows ~~the~~a component using the implementation.

341

```

342 | <?xml version="1.0" encoding="ASCII"?>
343 | <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
344 |   name="LoanComposite" >
345 |   ...
346 |
347 |   <component name="LoanService">
348 |     <implementation.cpp library="loan" class="LoanServiceImpl" />
349 |   </component>
350 | </composite>

```

351 | *Snippet 2-12: Component Using Implementation in Snippet 2-10*

## 352 | 2.4.1 Interface.cpp

353 | *The following snippet Snippet 2-13 shows the pseudo-schema for the C++ interface element used to type*  
354 | *services and references of component types.*

355

```

356 | <?xml version="1.0" encoding="ASCII"?>
357 | <!-- interface.cpp schema snippet -->
358 | <interface.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
359 |   header="string" class="Name"? remotable="boolean"?
360 |   callbackHeader="string" callbackClass="Name"?
361 |   requires="listOfQNames"? policySets="listOfQNames"? >
362 |
363 |   <function ... />*
364 |   <callbackFunction ... />*
365 |   <requires/>*
366 |   <policySetAttachment/>*
367 |
368 | </interface.cpp>

```

369 | *Snippet 2-13: Pseudo-schema for C++ Interface Element*

370

371 | The **interface.cpp** element has the ~~following~~ **attributes**:

- 372 | • **header : string (1..1)** – full name of the header file that describes the interface, including relative path  
373 | from the composite root.
- 374 | • **class : Name (0..1)** – name of the class declaration for the interface in the header file, including any  
375 | namespace definition. If the header file identified by the @header attribute of an <interface.cpp/>  
376 | element contains more than one class, then the @class attribute MUST be specified for the  
377 | <interface.cpp/> element. [CPP20005]
- 378 | • **callbackHeader : string (0..1)** – full name of the header file that describes the callback interface,  
379 | including relative path from the composite root.
- 380 | • **callbackClass : Name (0..1)** – name of the class declaration for the callback interface in the callback  
381 | header file, including any namespace definition. If the header file identified by the @callbackHeader



382 attribute of an `<interface.cpp/>` element contains more than one class, then the `@callbackClass`  
383 attribute MUST be specified for the `<interface.cpp/>` element. [CPP20006]

384 • **remotable : boolean (0..1)** – indicates whether the service is remotable or local. The default is local.  
385 See Implementing a Remotable Service

386 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification  
387 [POLICY] for a description of this attribute. If intents are specified at both the class and member  
388 function level, the effective intents for the member function is determined by merging the combined  
389 intents from the member function with the combined intents for the class according to the Policy  
390 Framework rules for merging intents within a structural hierarchy, with the member function at the  
391 lower level and the class at the higher level.

392 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification  
393 [POLICY] for a description of this attribute.

394

395 The `interface.cpp` element has the following child elements:

396 • **function : CPPFunction (0..n)** – see Function and CallbackFunction

397 • **callbackFunction : CPPFunction (0..n)** – see Function and CallbackFunction

398 • **requires : requires (0..n)** - See the Policy Framework specification [POLICY] for a description of this  
399 element.

400 • **policySetAttachment : policySetAttachment (0..n)** - See the Policy Framework specification  
401 [POLICY] for a description of this element.

## 402 2.4.2 Function and CallbackFunction

403 Some member functions-function of an interface might have behavioral characteristics, which will be  
404 described later, that need to be identified. This is done using a `function` or `callbackFunction` child element  
405 of `interface.cpp`. These child elements are also used when not all functions in a class are part of the  
406 interface.

407

408 • The following snippet if the header file identified by the `@header` attribute of an `<interface.cpp/>`  
409 element contains function declarations that are not operations of the interface, then the functions that  
410 are not operations of the interface MUST be excluded using `<function/>` child elements of the  
411 `<interface.cpp/>` element with `@exclude="true"`. [CPP20016]

412 • If the header file identified by the `@callbackHeader` attribute of an `<interface.cpp/>` element contains  
413 function declarations that are not operations of the callback interface, then the functions that are not  
414 operations of the callback interface MUST be excluded using `<callbackFunction/>` child elements of  
415 the `<interface.cpp/>` element with `@exclude="true"`. [CPP20017]

416 Snippet 2-14 shows the `interface.cpp` pseudo-schema with the pseudo-schema for the `function` and  
417 `callbackFunction` child elements:

418

```
419 <?xml version="1.0" encoding="ASCII"?>
420 <!-- Function schema snippet -->
421 <interface.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
422 ... >
423
424   <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
425     oneWay="Boolean"? /*exclude="Boolean"? >
426     <requires/*>
427     <policySetAttachment/*>
428     </function> *
429
430   <callbackFunction name="NCName" requires="listOfQNames"?
431     policySets="listOfQNames"? oneWay="Boolean"? /*exclude="Boolean"? >
```

```

432     <requires/>*
433     <policySetAttachment/>*
434   </callbackFunction> *
435
436 </interface.cpp>

```

437 [Snippet 2-14: Pseudo-schema for Interface Function and CallbackFunction Sub-elements](#)

438

439 The **function** and **callbackFunction** elements have the following attributes:

440 • **name** : **NCName (1..1)** – name of the method being decorated. The @name attribute of a <function/>  
 441 child element of a <interface.cpp/> MUST be unique amongst the <function/> elements of that  
 442 <interface.cpp/>. [CPP20007]

443 The @name attribute of a <callbackFunction/> child element of a <interface.cpp/> MUST be unique  
 444 amongst the <callbackFunction/> elements of that <interface.cpp/>. [CPP20008]

445 • ~~requires~~ : **listOfQNames (0..1)** – a list of **policy** intents-. See the Policy Framework specification  
 446 [POLICY] ~~needed by this member function.~~

447 • ~~for a description of this attribute.~~

448 • **policySets** : **listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification  
 449 [POLICY] for a description of this attribute.

450 • **oneWay** : **boolean (0..1)** – see Non-blocking Calls

451 • ~~exclude~~ : **boolean (0..1)** – if true, the member function is excluded from the interface. The default is  
 452 ~~false.~~

453 The **function** and **callbackFunction** elements have the **child elements**:

454 • ~~requires~~ : **requires (0..n)** - See the Policy Framework specification [POLICY] for a description of this  
 455 element.

456 • ~~policySetAttachment~~ : **policySetAttachment (0..n)** - See the Policy Framework specification  
 457 [POLICY] for a description of this element.

## 458 2.4.3 Implementation.cpp

459 The following snippet Snippet 2-15 shows the **pseudo**-schema for the C++ implementation element used  
 460 to define the implementation of a component.

461

```

462 <?xml version="1.0" encoding="ASCII"?>
463 <!-- implementation.cpp schema snippet -->
464 <implementation.cpp xmlns="http://docs.oasis-
465 open.org/ns/opencsa/sca/200903200912"
466   library="NCName" path="string"? class="Name"
467   scope="scope"? componentType="string"? allowsPassByReference="Boolean"?
468   eagerInit="boolean"? >requires="listOfQNames"?
469
470 <method ...   policySets="listOfQNames"? >
471
472   <function ... />*
473   <requires/>*
474   <policySetAttachment/>*
475
476 </implementation.cpp>

```

477 [Snippet 2-15: Pseudo-schema for C++ Implementation Element](#)

478

479 The **implementation.cpp** element has the following attributes:

- 480 • **library : NCName (1..1)** – name of the dll or shared library that holds the factory for the service  
481 component. This is the root name of the library.
  - 482 • **path : string (0..1)** - path to the library which is either relative to the root of the contribution containing  
483 the composite or is prefixed with a contribution import name and is relative to the root of the import.  
484 See C++ Contributions.
  - 485 • **class : Name (1..1)** – name of the class declaration of the implementation, including any namespace  
486 definition. The name of the componentType file for a C++ implementation MUST match the class  
487 name (excluding any namespace definition) of the implementations as defined by the @class attribute  
488 of the <implementation.cpp/> element. [CPP2009] The SCA runtime will append .componentType to  
489 the class name to find the componentType file.
  - 490 • **scope : CPPImplementationScope (0..1)** – identifies the scope of the component implementation.  
491 The default is stateless. See Component Implementation Scopes
  - 492 • **componentType : string (0..1)** – path to the componentType file which is relative to the root of the  
493 contribution containing the composite or is prefixed with a contribution import name and is relative to  
494 the root of the import.
  - 495 • **allowsPassByReference : boolean (0..1)** – indicates the implementation allows pass by reference  
496 data exchange semantics on calls to it or from it. These semantics apply to all services provided by  
497 and references used by an implementation. See AllowsPassByReference
  - 498 • **eagerInit : boolean (0..1)** – indicates a composite scoped implementation is to be initialized when it  
499 is loaded. See Composite [Scope](#)
  - 500
  - 501 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification  
502 [POLICY] for a description of this attribute. If intents are specified at both the class and member  
503 function level, the effective intents for the member function is determined by merging the combined  
504 intents from the member function with the combined intents for the class according to the Policy  
505 Framework rules for merging intents within a structural hierarchy, with the member function at the  
506 lower level and the class at the higher level.
  - 507 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification  
508 [POLICY] for a description of this attribute.
- 509 The *implementation.cpp* element has the following ~~child elements~~:
- 510 • **function : CPPImplementationMethod (0..n)** – see Implementation Function
  - 511 • **requires : requires (0..n)** - See the Policy Framework specification [POLICY] for a description of this  
512 element.
  - 513 • **policySetAttachment : policySetAttachment (0..n)** - See the Policy Framework specification  
514 [POLICY] for a description of this element.

## 515 2.4.4 Implementation Function

516 **SomeA** member ~~functions-function~~ of an implementation **might** have operational characteristics that need  
517 to be identified. This is done using a *function* child element of *implementation.cpp*

518

519 **The following snippet** [Snippet 2-16](#) shows the *implementation.cpp* schema with the schema for a *method*  
520 child element:

```
521
522 <?xml version="1.0" encoding="ASCII"?>
523 <!-- ImplementationFunction schema snippet -->
524 <implementation.cpp xmlns="http://docs.oasis-
525 open.org/ns/opencsa/sca/200903200912" ... >
526
527     <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
528         allowsPassByReference="boolean"? <-->
```

```
529     <requires/>*
530     <policySetAttachment/>*
531     </function> *
532
533 </implementation.cpp>
```

534 *Snippet 2-16: Pseudo-schema for Implementation Function Sub-element*

535

536 The **function** element has the following attributes:

- 537 • **name : NCName (1..1)** – name of the method being decorated. The @name attribute of a  
538 <function/> child element of a <implementation.cpp/> MUST be unique amongst the <function/>  
539 elements of that <implementation.cpp/>. [CPP20010]
- 540 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification  
541 [POLICY] needed by for a description of this member function attribute.
- 542 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification  
543 [POLICY] for a description of this attribute.
- 544 • **allowsPassByReference : boolean (0..1)** – indicates the member function allows pass by reference  
545 data exchange semantics. See AllowsPassByReference

546 The **function** element has the child elements:

- 547 • **requires : requires (0..n)** - See the Policy Framework specification [POLICY] for a description of this  
548 element.
- 549 • **policySetAttachment : policySetAttachment (0..n)** - See the Policy Framework specification  
550 [POLICY] for a description of this element.

## 551 2.5 Instantiation

552 A C++ implementation class MUST be default constructable by the SCA runtime to instantiate the  
553 component. [CPP20011]

---

## 554 3 Basic Client Model

555 This section describes how to get access to SCA services from both SCA components and from non-SCA  
556 components. It also describes how to call methods of these services.

### 557 3.1 Accessing Services from Component Implementations

558 A component can get access to a service using a component context.

559 |  
560 | **The following snippet** [Snippet 3-1](#) shows the `ComponentContext` C++ class with its `getService()`  
561 | member function.

562

```
563 namespace oasis {  
564     namespace sca {  
565  
566         class ComponentContext {  
567             public:  
568                 static ComponentContextPtr getCurrent();  
569                 virtual ServiceProxyPtr getService(  
570                     const std::string& referenceName) const = 0;  
571                 ...  
572             }  
573         }  
574     }  
}
```

575 | [Snippet 3-1: Partial ComponentContext Class Definition](#)

576

577 The `getService()` member function takes as its input argument the name of the reference and returns  
578 a pointer to a proxy providing access to the service. The returned pointer is to a generic `ServiceProxy`  
579 and is assigned to a pointer to a proxy which is derived from `ServiceProxy` and implements the  
580 interface of the reference.

581

582 | **The following shows** [Snippet 3-2](#) a sample of how the `ComponentContext` is used in a C++ component  
583 | implementation. The `getService()` member function is called on the `ComponentContext` passing the  
584 | reference name as input. The return of the `getService()` member function is cast to the abstract base  
585 | class defined for the reference.

586

```
587 #include "ComponentContext.h"  
588 #include "CustomerServiceProxy.h"  
589  
590 using namespace oasis::sca;  
591  
592 void clientFunction()  
593 {  
594  
595     unsigned long customerNumber = 1234;  
596  
597     ComponentContextPtr context = ComponentContext::getCurrent();  
598  
599     ServiceProxyPtr service = context->getService("customerService");  
600     CustomerServiceProxyPtr customerService =  
601         dynamicCast<CustomerServiceProxy>(service);  
602  
603     if (customerService)
```

```
604     short rating = customerService->getCreditRating(customerNumber);
605
606 }
```

607 [Snippet 3-2: Using ComponentContext](#)

## 608 3.2 Interface Proxies

609 For each Reference used by a client, a proxy class is generated by an SCA implementation. The proxy  
610 class for a Reference is derived from both the base `ServiceProxy` and the interface of the Reference  
611 (details below) and implements the necessary functionality to inform the SCA runtime that an operation is  
612 being invoked and submit the request over the transport determined by the wiring.

613

614 The base `ServiceProxy` class definition (in the namespace `oasis::sca`) is:

615

```
616 class ServiceProxy {
617 public:
618     //Possible future extensions
619 };
```

620 [Snippet 3-3: ServiceProxy Class Definition](#)

621

622 A remotable interface is always mappable to WSDL, which can be mapped to C++ as described in  
623 WSDL to C++ and C++ to WSDL Mapping. The proxy class for a remotable interface is derived from  
624 `ServiceProxy` and contains the member functions mapped from the WSDL definition for the interface. If  
625 a remotable interface is defined with a C++ class, an SCA implementation SHOULD map the interface  
626 definition to WSDL before generating the proxy for the interface. [CPP30001]

627

628 For the interface definition:

629

```
630 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
631             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
632             xmlns:tns="http://www.example.org/"
633             xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
634             targetNamespace="http://www.example.org/">
635
636     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
637               xmlns:tns="http://www.example.org/"
638               attributeFormDefault="unqualified"
639               elementFormDefault="unqualified"
640               targetNamespace="http://www.example.org/">
641         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
642         <xs:element name="GetLastTradePriceResponse"
643                   type="tns:GetLastTradePriceResponse"/>
644         <xs:complexType name="GetLastTradePrice">
645             <xs:sequence>
646                 <xs:element name="tickerSymbol" type="xs:string"/>
647             </xs:sequence>
648         </xs:complexType>
649         <xs:complexType name="GetLastTradePriceResponse">
650             <xs:sequence>
651                 <xs:element name="return" type="xs:float"/>
652             </xs:sequence>
653         </xs:complexType>
654     </xs:schema>
655
656     <-message name="GetLastTradePrice">
657         <part name="parameters" element="tns:GetLastTradePrice">
```

```

658     </part>
659 </message>
660
661 <-message name="GetLastTradePriceResponse">
662     <part name="parameters" element="tns:GetLastTradePriceResponse">
663         </part>
664 </-message>
665
666 <portType name="StockQuote">
667     <cpp:bindings>
668         <cpp:class name="StockQuoteService"/>
669     </cpp:bindings>
670     <operation name="GetLastTradePrice">
671         <cpp:bindings>
672             <cpp:memberFunction name="getTradePrice"/>
673         </cpp:bindings>
674         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
675             </input>
676         <output name="GetLastTradePriceResponse"
677             message="tns:GetLastTradePriceResponse">
678             </output>
679         </operation>
680     </portType>
681 </definitions>

```

682 [\*Snippet 3-4: Sample WSDL Interface\*](#)

683

684 The resulting abstract proxy class is:

685

```

686 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
687 //     serviceName="StockQuoteService")
688 class StockQuoteServiceProxy: public ServiceProxy {
689
690     // @WebFunction(operationName="GetLastTradePrice",
691     //     action="urn:GetLastTradePrice")
692     float getTradePrice(const std::string& tickerSymbol);
693 };

```

694 [\*Snippet 3-5: Proxy Class for Interface in Snippet 3-4\*](#)

695

696 The proxy class for a local interface is derived from `ServiceProxy` and contains the member functions  
697 of the C++ class defining the interface. For the interface definition:

698

```

699 // LoanService interface
700 class LoanService {
701 public:
702     virtual bool approveLoan(unsigned long customerNumber,
703                             unsigned long loanAmount) = 0;
704 };

```

705 [\*Snippet 3-6: Sample C++ Interface\*](#)

706

707 The resulting proxy class is:

708

```

709 class LoanServiceProxy : public ServiceProxy {
710 public:
711     virtual bool approveLoan(unsigned long customerNumber,
712                             unsigned long loanAmount) = 0;
713 };

```



714 [Snippet 3-7: Proxy Class for Interface in Snippet 3-6](#)

715

716 For each reference of a component, an SCA implementation MUST generate a service proxy derived  
717 from `ServiceProxy` that contains the operations of the reference's interface definition. [CPP30002]

### 718 **3.3 Accessing Services from non-SCA component** 719 **implementationsComponent Implementations**

720 Non-SCA components can access component services by obtaining a `DomainContextPtr` from the  
721 SCA runtime and then following the same steps as a component implementation as described above.

722

723 [The following Snippet 3-8](#) shows a sample of how the `DomainContext` is used in non-SCA C++ code.

724

```
725 #include "DomainContext.h"  
726 #include "CustomerServiceProxy.h"  
727  
728 void externalFunction()  
729 {  
730  
731     unsigned long customerNumber = 1234;  
732  
733     DomainContextPtr context = myImplGetDomain("http://example.com/mydomain");  
734  
735     ServiceProxyPtr service = context->getService("customerService");  
736     CustomerServiceProxyPtr customerService =  
737         dynamicCast<CustomerServiceProxy>(service);  
738  
739     if (customerService)  
740         short rating = customerService->getCreditRating(customerNumber);  
741  
742 }
```

743 [Snippet 3-8: Using DomianContext](#)

744

745 No SCA metadata is specified for the client. E.g. no binding or policies are specified. Non-SCA clients  
746 cannot call services that use callbacks.

747

748 The SCA infrastructure decides which binding is used OR extended form of serviceURI is used:

- 749 • `componentName/serviceName/bindingName`

750

751 The function `myImplGetDomain()` in [Snippet 3-8](#) is an example of how a non-SCA client might get a  
752 `DomainContextPtr` and is not a function defined by this specification. The specific mechanism for how  
753 an SCA runtime implementation returns a `DomainContextPtr` is not defined by this specification.

### 754 **3.4 Calling Service Operations**

755 ~~The previous sections~~ Accessing Services from Component Implementations and Accessing Services  
756 from non-SCA Component Implementations show ~~the various options for getting~~ how to get access to a  
757 service. Once you have access to the service, calling an operation of the service is like calling a member  
758 function of a C++ class: via a ServiceProxy.

759



760 If you have access to a service whose interface is marked as remotable, then on calls to operations of  
761 that service you will experience remote semantics. Arguments and return are passed by-value and it is  
762 possible to get a `ServiceUnavailableException`, which is a `ServiceRuntimeException`.

### 763 3.5 Long Running Request-Response Operations

764 The Assembly Specification [ASSEMBLY] allows service interfaces or individual operations to be marked  
765 **long-running** using an `@requires="asyncInvocation"` intent, with the meaning that the operation(s) might  
766 not complete in any specified time interval, even when the operations are request-response operations.  
767 A client calling such an operation has to be prepared for any arbitrary delay between the time a request is  
768 made and the time the response is received. To support this kind of operation three invocation styles are  
769 available: asynchronous – the client provides a response handler, polling – the client will poll the SCA  
770 runtime to determine if a response is available, and synchronous – the SCA runtime handles suspension  
771 of the main thread, asynchronously receiving the response and resuming the main thread. The details of  
772 each of these styles are provided in the following sections.

773  
774 For a service operation with signature

```
775 <return type> <function name>(<parameters>);
```

776 the asynchronous invocation style includes a member function in the interface proxy class

```
777 <proxy_class>::<response_message_name>Response <function name>Async(  
778 <in_parameters>);
```

779 *[Snippet 3-9: AsynchronousInvocation Member Function Format](#)*

780  
781 where `<response_message name>Response` is the response class for the operation as defined by  
782 Response Class. The client uses this member function to issue a request through the SCA runtime. The  
783 response is returned immediately, and can be used to access the response when it becomes available.

784  
785 An SCA runtime MUST include an asynchronous invocation member function for every operation of a  
786 reference interface with a `@requires="asyncInvocation"` intent applied either to the operation or the  
787 reference as a whole. [CPP30003]

788  
789 The following [Snippet 3-10](#) shows a sample proxy class interface providing an asynchronous API.

```
790  
791 using namespace oasis::sca;  
792 using namespace commonj::sdo;  
793  
794  
795 class CustomerServiceProxy : public ServiceProxy {  
796 public:  
797  
798     // synchronous member function  
799     virtual short getCreditRating(unsigned long customerNumber) = 0;  
800  
801     // forward declare callback class  
802     class getCreditRatingCallback;  
803  
804     // asynchronous response object  
805     class getCreditRatingResponse {  
806     public:  
807         // IOU/Future member functions  
808         virtual void cancel() = 0;  
809         virtual bool isCancelled() = 0;  
810         virtual bool isReady() = 0;
```

```

811     virtual void setCallback(getCreditRatingCallbackPtr callback) = 0;
812
813     virtual short getReturn() = 0;
814 };
815
816     // asynchronous callback object
817     class getCreditRatingCallback {
818     public:
819         virtual void invoke(getCreditRatingResponsePtr) = 0;
820     };
821
822     // asynchronous member function
823     virtual getCreditRatingResponsePtr getCreditRatingAsync(unsigned long
824         customerNumber) = 0;
825 };
826

```

827 *[The following Snippet 3-10: Proxy with an Asynchronous API](#)*

828

829 [Snippet 3-11](#) shows a sample of how the asynchronous invocation style is used in a C++ component  
830 implementation.

831

```

832 #include "ComponentContext.h"
833 #include "CustomerServiceProxy.h"
834
835 using namespace oasis::sca;
836
837 void clientFunction()
838 {
839     ComponentContextPtr context = ComponentContext::getCurrent();
840
841     ServiceProxyPtr service = context->getService("customerService");
842     CustomerServiceProxyPtr customerService =
843         dynamicCast<CustomerServiceProxy>(service);
844
845     if (customerService) {
846         getCreditRatingResponsePtr response =
847             customerService->getCreditRatingAsync(1234);
848
849         // ...
850     }
851 }
852
853

```

854 *[Snippet 3-11: Using an Asynchronous API](#)*

855

856 Once a response object has been returned, the user can use the provided API in order to set a callback  
857 object to be invoked when the response is ready, to poll the variable waiting for the response to become  
858 available, or to block the current thread waiting for the response to become available.

### 859 **3.5.1 Response Callback**

860 If a callback is specified on a response object, that callback will be invoked when the response is received  
861 by the runtime. [The following example Snippet 3-12](#) demonstrates creating a response object and setting  
862 it on a response instance:

863

```

864 #include "ComponentContext.h"
865 #include "CustomerServiceProxy.h"
866

```

```

867 using namespace oasis::sca;
868
869 class CreditRatingCallback :
870     public CustomerServiceProxy::getCreditRatingCallback {
871
872     virtual void invoke(getCreditRatingResponsePtr response) {
873         try {
874             short rating = response->getReturn();
875         }
876         catch (...) {
877             // ...
878         }
879     }
880 };
881
882 void clientFunction()
883 {
884     ComponentContextPtr context = ComponentContext::getCurrent();
885
886     ServiceProxyPtr service = context->getService("customerService");
887     CustomerServiceProxyPtr customerService =
888         dynamicCast<CustomerServiceProxy>(service);
889
890     if (customerService) {
891         CustomerServiceProxy::getCreditRatingResponsePtr response =
892             customerService->getCreditRatingAsync(1234);
893
894         CustomerServiceProxy::getCreditRatingCallbackPtr callback =
895             new CreditRatingCallback();
896
897         response->setCallback(callback);
898
899         // ...
900     }
901 }

```

902 [\*Snippet 3-12: Using a Response Object\*](#)

### 903 **3.5.2 Response Polling**

904 A client can poll a response object in order to determine if a response has been received.

```

905
906 #include "ComponentContext.h"
907 #include "CustomerServiceProxy.h"
908
909 using namespace oasis::sca;
910
911 void clientFunction()
912 {
913     ComponentContextPtr context = ComponentContext::getCurrent();
914
915     ServiceProxyPtr service = context->getService("customerService");
916     CustomerServiceProxyPtr customerService =
917         dynamicCast<CustomerServiceProxy>(service);
918
919     if (customerService) {
920         CustomerServiceProxy::getCreditRatingResponsePtr response =
921             customerService->getCreditRatingAsync(1234);
922
923         while (!response->isReady()) {
924             // do something else
925         }
926
927         // The response is ready and can be accessed without blocking.

```

```

928     try {
929         short rating = response->getReturn();
930     }
931     catch (...) {
932         // ...
933     }
934 }
935 }

```

936 [Snippet 3-13: Polling a Response Object](#)

### 937 3.5.3 Synchronous Response Access

938 If a client chooses to block until a response becomes available, they can attempt to access a part of the  
939 response object. If the response has not been received, the call will block until the response is available.  
940 Once the response is received and the response object is populated, the call will return.

```

941
942 #include "ComponentContext.h"
943 #include "CustomerServiceProxy.h"
944
945 using namespace oasis::sca;
946
947 void clientFunction()
948 {
949     ComponentContextPtr context = ComponentContext::getCurrent();
950
951     ServiceProxyPtr service = context->getService("customerService");
952     CustomerServiceProxyPtr customerService =
953         dynamicCast<CustomerServiceProxy>(service);
954
955     if (customerService) {
956         CustomerServiceProxy::getCreditRatingResponsePtr response =
957             customerService->getCreditRatingAsync(1234);
958
959         // The response is ready and can be accessed without blocking.
960         try {
961             short rating = response->getReturn();
962         }
963         catch (...) {
964             // ...
965         }
966     }
967 }
968 }

```

969 [Snippet 3-14: Blocking on a Response Object](#)

### 970 3.5.4 Response Class

971 The proxy for an interface includes a response class for a response message type returned by an  
972 operation that can be invoked asynchronously. A response class presents the following interface to the  
973 client component.

```

974
975 class <response_message_name>Response {
976 public:
977     virtual <response_message_type> getReturn() const = 0;
978     virtual void setCallback(<response_message_name>CallbackPtr callback) = 0;
979     virtual boolean isReady() const = 0;
980     virtual void cancel() const = 0;
981     virtual boolean isCancelled() const = 0;
982 };

```

983 [Snippet 3-15: AsynchronousInvocation Response Class Definition](#)

984

985 An SCA runtime MUST include a response class for every response message of a reference interface  
986 that can be returned by an operation of the interface with a `@requires="asynchronous"` intent applied  
987 either to the operation of the reference as a whole. [CPP30004]

### 988 3.5.4.1 getReturn

989 A C++ component implementation uses `getReturn()` to retrieve the response data for an asynchronous  
990 invocation.

Precondition	C++ component instance is running and has an outstanding asynchronous call	
Input Parameter		
Return	Response data for the operation.	
Throws	Any exceptions defined for the operation.	
Post Condition	The response object is marked as done.	

991 [Table 3-1: AsynchronousInvocation Response::getReturn Details](#)

### 992 3.5.4.2 setCallback

993 A C++ component implementation uses `setCallback()` to set a callback object for an asynchronous  
994 invocation.

Precondition	C++ component instance is running and has an outstanding asynchronous call	
Input Parameter	<code>callback</code>	A pointer to the callback object for the response
Return		
Post Condition	The response object is marked as done.	

995 [Table 3-2: AsynchronousInvocation Response::setCallback Details](#)

### 996 3.5.4.3 isReady

997 A C++ component implementation uses `isReady()` to determine if the response data for an polling  
998 invocation is available.

Precondition	C++ component instance is running and has an outstanding polling call	
Input Parameter		
Return	True if the response is available	
Post Condition	No change	

999 [Table 3-3: AsynchronousInvocation Response::isReady Details](#)

### 1000 3.5.4.4 cancel

1001 A C++ component implementation uses `cancel()` to cancel an outstanding invocation.

Precondition	C++ component instance is running and has an outstanding asynchronous call	
Input Parameter		

Return	
Post Condition	If a response is subsequently received for the operation, it will be discarded.

1002 [\*Table 3-4: AsynchronousInvocation Response::cancel Details\*](#)

1003 **3.5.4.5 isCancelled**

1004 A C++ component implementation uses `isCancelled()` to determine if another thread has cancelled an  
 1005 outstanding invocation.

Precondition	C++ component instance is running and has an outstanding asynchronous call	
Input Parameter		
Return	True if the operation has been cancelled	
Post Condition	No change	

1006 [\*Table 3-5: AsynchronousInvocation Response::isCancelled Details\*](#)

1007

## 4 Asynchronous Programming

1008 Asynchronous programming of a service is where a client invokes a service and carries on executing  
1009 without waiting for the service to execute. Typically, the invoked service executes at some later time.  
1010 Output from the invoked service, if any, is fed back to the client through a separate mechanism, since no  
1011 output is available at the point where the service is invoked. This is in contrast to the call-and-return style  
1012 of synchronous programming, where the invoked service executes and returns any output to the client  
1013 before the client continues. The SCA asynchronous programming model consists of support for non-  
1014 blocking operation calls and callbacks. ~~Each of these topics is discussed in the following sections.~~

### 4.1 Non-blocking Calls

1016 Non-blocking calls represent the simplest form of asynchronous programming, where the client of the  
1017 service invokes the service and continues processing immediately, without waiting for the service to  
1018 execute.

1019

1020 Any member function that returns `void`, has only by-value parameters and has no declared exceptions  
1021 can be marked with the `@oneWay="true"` attribute in the interface definition of the service. An operation  
1022 marked as `oneWay` is considered non-blocking and the SCA runtime MAY use a binding that buffers the  
1023 requests to the member function and sends them at some time after they are made. [CPP40001]

1024

1025 The following snippet [Snippet 4-1](#) shows the component type for a service with the `reportEvent()`  
1026 member function declared as a one-way operation:

1027

```
1028 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912">  
1029   <service name="LoanService">  
1030     <interface.cpp header="LoanService.h">  
1031       <function name="reportEvent" oneWay="true" />  
1032     </interface.cpp>  
1033   </service>  
1034 </componentType>
```

1035 [Snippet 4-1: ComponentType with oneWay Member Function](#)

1036

1037 SCA does not currently define a mechanism for making non-blocking calls to methods that return values  
1038 or are declared to throw exceptions. It is considered to be a best practice that service designers define  
1039 one-way member function as often as possible, in order to give the greatest degree of binding flexibility to  
1040 deployers.

### 4.2 Callbacks

1042 Callback services are used by *bidirectional services* as defined in the Assembly Specification  
1043 [ASSEMBLY].

1044

1045 A callback interface is declared by the `@callbackHeader` and `@callbackClass` attributes in the interface  
1046 definition of the service. The following snippet [Snippet 4-2](#) shows the component type for a service  
1047 *MyService* with the interface defined in *MyService.h* and the interface for callbacks defined in  
1048 *MyServiceCallback.h*,

1049

```
1050 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912">  
1051 >  
1052   <service name="MyService">
```

```
1053     <interface.cpp header="MyService.h"
1054         callbackHeader="MyServiceCallback.h"/>
1055     </service>
1056 </componentType>
```

1057 [Snippet 4-2: ComponentType with a Callback Interface](#)

## 1058 4.2.1 Using Callbacks

1059 Bidirectional interfaces and callbacks are used when a simple request/response pattern isn't sufficient to  
1060 capture the business semantics of a service interaction. Callbacks are well suited for cases when a  
1061 service request can result in multiple responses or new requests from the service back to the client, or  
1062 where the service might respond to the client some time after the original request has completed.

1063  
1064 ~~The following example shows~~[Snippet 4-3](#)~~–~~[Snippet 4-5 show](#) a scenario in which bidirectional interfaces  
1065 and callbacks could be used. A client requests a quotation from a supplier. To process the enquiry and  
1066 return the quotation, some suppliers might need additional information from the client. The client does  
1067 not know which additional items of information will be needed by different suppliers. This interaction can  
1068 be modeled as a bidirectional interface with callback requests to obtain the additional information.

```
1069  
1070 class Quotation {
1071 public:
1072     virtual double requestQuotation(std::string productCode,
1073                                     unsigned int quantity) = 0;
1074 };
1075  
1076 class QuotationCallback {
1077 public:
1078     virtual std::string getState() = 0;
1079     virtual std::string getZipCode() = 0;
1080     virtual std::string getCreditRating() = 0;
1081 };
```

1082 [Snippet 4-3: C++ Interface with a Callback Interface](#)

1083  
1084 In ~~this example~~[Snippet 4-3](#), the `requestQuotation` operation requests a quotation to supply a given  
1085 quantity of a specified product. The `QuotationCallback` interface provides a number of operations that the  
1086 supplier can use to obtain additional information about the client making the request. For example, some  
1087 suppliers might quote different prices based on the state or the zip code to which the order will be  
1088 shipped, and some suppliers might quote a lower price if the ordering company has a good credit rating.  
1089 Other suppliers might quote a standard price without requesting any additional information from the client.

1090  
1091 ~~The following code snippet~~[Snippet 4-4](#) illustrates a possible implementation of the example service.

```
1092  
1093 #include "QuotationImpl.h"
1094 #include "QuotationCallbackProxy.h"
1095 #include "ComponentContext.h"
1096 using namespace oasis::sca;
1097  
1098 double QuotationImpl::requestQuotation(std::string productCode,
1099                                         unsigned int quantity) {
1100     double price = getPrice(productCode, quantity);
1101     double discount = 0;
1102  
1103     ComponentContextPtr context = ComponentContext::getCurrent();
1104     ServiceReferencePtr serviceRef = context->getSelfReference();
1105     ServiceProxyPtr callback = serviceRef->getCallback();
1106     QuotationCallbackQuotationCallbackProxyPtr quotationCallback =
```



```

1107     dynamicCast<QuotationCallbackProxy>(callback);
1108
1109     if (quotationCallback) {
1110         if (quantity > 1000 && callback->getState().compare("FL") == 0)
1111             discount = 0.05;
1112         if (quantity > 10000 && callback->getCreditRating().data() == 'A')
1113             discount += 0.05;
1114     }
1115     return price * (1-discount);
1116 }

```

1117 [Snippet 4-4: Implementation of Forward Service with Interface in Snippet 4-3](#)

1118

1119 ~~The code snippet below~~ [Snippet 4-5](#) is taken from the client of this example service. The client's service  
1120 implementation class implements the member functions of the QuotationCallback interface as well as  
1121 those of its own service interface ClientService.

1122

```

1123 #include "QuotationCallback.h"
1124 #include "QuotationServiceProxy.h"
1125 #include "ComponentContext.h"
1126 using namespace oasis::sca;
1127
1128 void ClientImpl:: aClientFunction() {
1129     ComponentContextPtr context = ComponentContext::getCurrent();
1130
1131     ServiceProxyPtr service = context->getService("quotationService");
1132     QuotationServiceProxyPtr quotationService =
1133         dynamicCast<QuotationServiceProxy>(service);
1134
1135     if (quotationService)
1136         quotationService->requestQuotation("AB123", 2000);
1137 }
1138
1139 std::string QuotationCallbackImpl::getState() {
1140     return "TX";
1141 }
1142 std::string QuotationCallbackImpl::getZipCode() {
1143     return "78746";
1144 }
1145 std::string QuotationCallbackImpl::getCreditRating() {
1146     return "AA";
1147 }

```

1148 [Snippet 4-5: Implementation of Callback Interface in Snippet 4-3](#)

1149

1150 For each service of a component that includes a bidirectional interface, an SCA implementation MUST  
1151 generate a service proxy derived from `ServiceProxy` that contains the operations of the reference's  
1152 callback interface definition. [\[CPP40002\]](#)

1153

1154 If a service of a component that has a callback interface contains operations with a  
1155 `@requires="asyncInvocation"` intent applied either to the operation of the reference as a whole, an SCA  
1156 implementation MUST include asynchronous invocation member functions and response classes as  
1157 described in Long Running Request-Response Operations. [\[CPP40003\]](#)

1158

1159 In the example the callback is **stateless**, i.e., the callback requests do not need any information relating  
1160 to the original service request. For a callback that needs information relating to the original service  
1161 request (a **stateful** callback), this information can be passed to the client by the service provider as  
1162 parameters on the callback request.

## 1163 4.2.2 Callback Instance Management

1164 Instance management for callback requests received by the client of the bidirectional service is handled in  
1165 the same way as instance management for regular service requests. If the client implementation has  
1166 **STATELESS***stateless* scope, the callback is dispatched using a newly initialized instance. If the client  
1167 implementation has **COMPOSITE***composite* scope, the callback is dispatched using the same shared  
1168 instance that is used to dispatch regular service requests.

1169 | As described [in](#) Using Callbacks, a stateful callback can obtain information relating to the original service  
1170 request from parameters on the callback request. Alternatively, a composite-scoped client could store  
1171 information relating to the original request as instance data and retrieve it when the callback request is  
1172 received. These approaches could be combined by using a key passed on the callback request (e.g., an  
1173 order ID) to retrieve information that was stored in a composite-scoped instance by the client code that  
1174 made the original request.

## 1175 4.2.3 Implementing Multiple Bidirectional Interfaces

1176 Since it is possible for a single class to implement multiple services, it is also possible for callbacks to be  
1177 defined for each of the services that it implements. To access the callbacks the  
1178 `ServiceReference::getCallback(serviceName)` member function is used, passing in the name  
1179 of the service for which the callback is to be obtained.

---

## 5 Error Handling

1180

1181 Clients calling service operations will experience business exceptions, and SCA runtime exceptions.

1182

1183 Business exceptions are raised by the implementation of the called service operation. It is expected that  
1184 these will be caught by client invoking the operation on the service.

1185

1186 SCA runtime exceptions are raised by the SCA runtime and signal problems in the management of the  
1187 execution of components, and in the interaction with remote services. Currently the ~~following~~ SCA runtime  
1188 exceptions ~~are~~ defined are:

- 1189 • `SCAException` – defines a root exception type from which all SCA defined exceptions derive.
  - 1190 – `SCANullPointerException` – signals that code attempted to dereference a null pointer from a  
1191 `RefCountingPointer` object.
  - 1192 – `ServiceRuntimeException` - signals problems in the management of the execution of SCA  
1193 components.
    - 1194 • `ServiceUnavailableException` – signals problems in the interaction with remote  
1195 services. This extends `ServiceRuntimeException`. These are exceptions that could be  
1196 transient, so retrying is appropriate. Any exception that is a `ServiceRuntimeException`  
1197 that is not a `ServiceUnavailableException` is unlikely to be resolved by retrying the  
1198 operation, since it most likely requires human intervention.
    - 1199 • `MultipleServicesException` – signals that a member function expecting identification of  
1200 a single service is called where there are multiple services defined. Thrown by  
1201 `ComponentContext::getService()`, `ComponentContext::getSelfReference()`  
1202 and `ComponentContext::getServiceReference()`.

## 6 C++ API

1203

1204 All the C++ interfaces are found in the namespace `oasis::sca`, which has been omitted from the  
1205 [following descriptions/definitions](#) for clarity.

### 6.1 Reference Counting Pointers

1206

1207 These are a derived version of the familiar smart-pointer. The pointer class holds a real (dumb) pointer to  
1208 the object. If the reference counting pointer is copied, then a duplicate pointer is returned with the same  
1209 real pointer. A reference count within the object is incremented for each copy of the pointer, so only when  
1210 all pointers go out of scope will the object be freed.

1211

1212 Reference counting pointers in SCA have the same name as the type they are pointing to, with a suffix of  
1213 Ptr. (E.g. `ComponentContextPtr`, `ServiceReferencePtr`).

1214

1215 `RefCountingPointer` defines member functions with raw pointer like semantics. This includes  
1216 defining operators for dereferencing the pointer (`*`, `->`), as well as operators for determining the validity of  
1217 the pointer.

1218

```
1219 template <typename T>  
1220 class RefCountingPointer {  
1221 public:  
1222     T& operator* () const;  
1223     T* operator-> () const;  
1224     operator void* () const;  
1225     bool operator! () const;  
1226 };  
1227  
1228 template <typename T, typename U>  
1229 RefCountingPointer<T> constCast(RefCountingPointer<U> other);  
1230  
1231 template <typename T, typename U>  
1232 RefCountingPointer<T> dynamicCast(RefCountingPointer<U> other);  
1233  
1234 template <typename T, typename U>  
1235 RefCountingPointer<T> reinterpretCast(RefCountingPointer<U> other);  
1236  
1237 template <typename T, typename U>  
1238 RefCountingPointer<T> staticCast(RefCountingPointer<U> other);
```

1239

1240 ~~The **RefCountingPointer** class has the following member functions:~~

1241 [Snippet 6-1: RefCountingPointer Class Definition](#)

#### 6.1.1 operator\*

1242

1243 A C++ component implementation uses the `*` operator to dereferences the underlying pointer of a  
1244 reference counting pointer. This is equivalent to calling `*p` where `p` is the underlying pointer.

Precondition	C++ component instance is running and has a reference counting pointer
Return	A reference to the value of the pointer
Throws	<code>SCANullPointerException</code> if the pointer is <code>NULL</code>

Post Condition	No change
----------------	-----------

1245 [Table 6-1: RefCountingPointer operator\\* Details](#)

### 1246 6.1.2 operator->

1247 A C++ component implementation uses the `->` operator to invoke member functions on the underlying  
 1248 pointer of a reference counting pointer. This is equivalent to invoking `p->func()` where `func()` is a  
 1249 member function defined on the underlying pointer type.

Precondition	C++ component instance is running and has a reference counting pointer
Return	
Throws	SCANullPointerException if the pointer is NULL
Post Condition	The underlying member functions has been processed.

1250 [Table 6-2: RefCountingPointer operator-> Details](#)

### 1251 6.1.3 operator void\*

1252 A C++ component implementation uses the `void*` operator to determine if the underlying pointer of a  
 1253 reference counting pointer is set, i.e. `if (p) { /* do something */ }`.

Precondition	C++ component instance is running and has a reference counting pointer
Return	Zero if the underlying pointer is null, otherwise a non-zero value
Post Condition	No change

1254 [Table 6-3: RefCountingPointer operator void\\* Details](#)

### 1255 6.1.4 operator!

1256 A C++ component implementation uses the `!` operator to determine if the underlying pointer of a  
 1257 reference counting pointer is not set, i.e. `if (!p) { /* do something */ }`.

Precondition	C++ component instance is running and has a reference counting pointer
Return	A non-zero value if the underlying pointer is null, otherwise zero
Post Condition	No change

1258 [Table 6-4: RefCountingPointer operator! Details](#)

### 1259 6.1.5 constCast

1260 The `constCast` global function provides the semantic behavior of the `const_cast` operator for use with  
 1261 RefCountingPointers.

Precondition	C++ component instance is running and has a reference counting pointer
Return	A RefCountingPointer instance templated on the target type.
Post Condition	No change

1262 [Table 6-5: constCast for RefCountingPointers Details](#)

## 1263 6.1.6 dynamicCast

1264 The dynamicCast global function provides the semantic behavior of the dynamic\_cast operator for use  
1265 with RefCountingPointers.

Precondition	C++ component instance is running and has a reference counting pointer
Return	A RefCountingPointer instance templated on the target type. This can return an invalid RefCountingPointer instance if the cast fails.
Post Condition	No change

1266 [Table 6-6: dynamicCast for RefCountingPointers Details](#)

## 1267 6.1.7 reinterpretCast

1268 The reinterpretCast global function provides the semantic behavior of the reinterpret\_cast operator for  
1269 use with RefCountingPointers.

Precondition	C++ component instance is running and has a reference counting pointer
Return	A RefCountingPointer instance templated on the target type.
Post Condition	No change

1270 [Table 6-7: reinterpretCast for RefCountingPointers Details](#)

## 1271 6.1.8 staticCast

1272 The staticCast global function provides the semantic behavior of the static\_cast operator for use with  
1273 RefCountingPointers.

Precondition	C++ component instance is running and has a reference counting pointer
Return	A RefCountingPointer instance templated on the target type.
Post Condition	No change

1274 [Table 6-8: staticCast for RefCountingPointers Details](#)

## 1275 6.2 Component Context

1276 The ~~following shows the complete~~ ComponentContext interface-definition is:  
1277

```
1278 class ComponentContext {  
1279 public:  
1280     static ComponentContextPtr getCurrent();  
1281  
1282     virtual std::string getURI() const = 0;  
1283  
1284     virtual ServiceProxyPtr getService(  
1285         const std::string& referenceName) const = 0;  
1286     virtual std::listvector<ServiceProxyPtr> getServices(  
1287         const std::string& referenceName) const = 0;  
1288  
1289     virtual ServiceReferencePtr getServiceReference(  
1290         const std::string& referenceName) const = 0;  
1291     virtual std::listvector<ServiceReferencePtr> getServiceReferences(  
1292         const std::string& referenceName) const = 0;  
1293  
1294
```

```

1295     virtual DataObjectPtr getProperties() const = 0;
1296     virtual DataFactoryPtr getDataFactory() const = 0;
1297
1298     virtual ServiceReferencePtr getSelfReference() const = 0;
1299     virtual ServiceReferencePtr getSelfReference(
1300         const std::string& serviceName) const = 0;
1301 };

```

1302  
1303 **The `ComponentContext` C++ interface has the following member functions:**

1304 [Snippet 6-2: `ComponentContext` Class Definition](#)

### 1305 6.2.1 `getCurrent`

1306 A C++ component implementation uses `ComponentContext::getCurrent()` to get a  
1307 `ComponentContext` object for itself.

Precondition	C++ component instance is running	
Input Parameter		
Return	<code>ComponentContext</code> for the current component	
Post Condition	The component instance has a valid context object to use for subsequent runtime calls.	

1308 [Table 6-9: `ComponentContext::getCurrent` Details](#)

### 1309 6.2.2 `getURI`

1310 A C++ component implementation uses `getURI()` to get an absolute URI for itself.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>	
Input Parameter		
Return	Absolute URI for the current component	
Post Condition	No change	

1311 [Table 6-10: `ComponentContext::getURI` Details](#)

### 1312 6.2.3 `getService`

1313 A C++ component implementation uses `getService()` to get a service proxy implementing the interface  
1314 defined for a Reference.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>	
Input Parameter	<code>referenceName</code>	Name of the Reference to get an interface object for
Return	Pointer to a <code>ServiceProxy</code> implementing the interface of the Reference. This will be NULL if <code>referenceName</code> is not defined for the component.	
Throws	<code>MultipleServicesException</code> if the reference resolves to more than one service	
Post Condition	A <code>ServiceProxy</code> object for the Reference is constructed. This <code>ServiceProxy</code> object is independent of any <code>ServiceReference</code> that are obtained for the Reference.	

1315 [Table 6-11: `ComponentContext::getService` Details](#)

## 1316 6.2.4 getServices

1317 A C++ component implementation uses `getServices()` to get a [listvector](#) of service proxies  
 1318 implementing the interface defined for a Reference.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>	
Input Parameter	<code>referenceName</code>	Name of the Reference to get an interface object for
Return	<a href="#">ListVector</a> of pointers to <code>ServiceProxy</code> objects implementing the interface of the Reference. This <a href="#">listvector</a> will be empty if <code>referenceName</code> is not defined for the component. Operations need to be invoked on each object in the <a href="#">listvector</a> .	
Post Condition	<code>ServiceProxy</code> objects for the Reference are constructed. These <code>ServiceProxy</code> objects are independent of any <code>ServiceReferences</code> that are obtained for the Reference.	

1319 [Table 6-12: ComponentContext::getServices Details](#)

## 1320 6.2.5 getServiceReference

1321 A C++ component implementation uses `getServiceReference()` to get a `ServiceReference` for a  
 1322 Reference.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>	
Input Parameter	<code>referenceName</code>	Name of the Reference to get a <code>ServiceReference</code> for
Return	<code>ServiceReference</code> for the Reference. This will be NULL if <code>referenceName</code> is not defined for the component.	
Throws	<code>MultipleServicesException</code> if the reference resolves to more than one service	
Post Condition	A <code>ServiceReference</code> for the Reference is constructed.	

1323 [Table 6-13: ComponentContext::getServiceReference Details](#)

## 1324 6.2.6 getServiceReferences

1325 A C++ component implementation uses `getServiceReferences()` to get a [listvector](#) of  
 1326 `ServiceReference` for a Reference.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>	
Input Parameter	<code>referenceName</code>	Name of the Reference to get a list of <code>ServiceReferences</code> for
Return	<a href="#">ListVector</a> of <code>ServiceReferences</code> for the Reference. This <a href="#">vector</a> will be empty if <code>referenceName</code> is not defined for the component.	
Post Condition	<code>ServiceReferences</code> for the Reference are constructed.	

1327 [Table 6-14: ComponentContext::getServiceReferences Details](#)

## 1328 6.2.7 getProperties

1329 A C++ component implementation uses `getProperties()` to get its configured property values.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>	
--------------	---	--



Input Parameter	
Return	An SDO <b>[SDO21]</b> from which all the properties defined in the componentType file can be retrieved.
Post Condition	An SDO with the property values for the component instance is constructed.

1330 [Table 6-15: ComponentContext::getProperties Details](#)

## 1331 6.2.8 getDataFactory

1332 A C++ component implementation uses `getDataFactory()` to get its an SDO `DataFactory` which  
 1333 can be used to create `DataObjects` for complex data types used by this component.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>
Input Parameter	
Return	An SDO <code>DataFactory</code> which has definitions for all complex data types used by a component.
Post Condition	An SDO <code>DataFactory</code> is constructed

1334 [Table 6-16: ComponentContext::getDataFactory Details](#)

## 1335 6.2.9 getSelfReference

1336 A C++ component implementation uses `getSelfReference()` to get a `ServiceReference` for use  
 1337 with some callback APIs.

1338

1339 There are two variations of this API.

Precondition	C++ component instance is running and has a <code>ComponentContext</code>
Input Parameter	
Return	A <code>ServiceReference</code> for the service provided by this component.
Throws	<code>MultipleServicesException</code> if the component implements more than one <code>Service</code>
Post Condition	A <code>ServiceReference</code> object is constructed

1340 and

Precondition	C++ component instance is running and has a <code>ComponentContext</code>
Input Parameter	<code>serviceName</code> Name of the Service to get a <code>ServiceReference</code> for
Return	A <code>ServiceReference</code> for the service provided by this component.
Post Condition	A <code>ServiceReference</code> object is constructed

1341 [Table 6-17: ComponentContext::getSelfReference Details](#)

## 1342 6.3 ServiceReference

1343 The following shows the `ServiceReference` interface- definition is:

1344

```

1345 class ServiceReference {
1346 public:
1347     virtual ServiceProxyPtr getService() const = 0;
1348
1349     virtual ServiceProxyPtr getCallback() const = 0;
1350 };

```

1351  
1352 *The [Snippet 6-3: ServiceReference](#) interface has the following member functions (the [Class Definition](#)*

1353  
1354 *The* detailed description of the usage of these member functions is described in Asynchronous  
1355 Programming.

1356 *);*

### 1357 6.3.1 getService

1358 A C++ component implementation uses `getService()` to get a service proxy implementing the interface  
1359 defined for a `ServiceReference`.

Precondition	C++ component instance is running and has a <code>ServiceReference</code>
Input Parameter	
Return	Pointer to a <code>ServiceProxy</code> implementing the interface of the <code>ServiceReference</code> .
Post Condition	A <code>ServiceProxy</code> object for the <code>ServiceReference</code> is constructed.

1360 *[Table 6-18: ServiceReference::getService Details](#)*

### 1361 6.3.2 getCallback

1362 A C++ component implementation uses `getCallback()` to get a service proxy implementing the  
1363 callback interface defined for a `ServiceReference`.

Precondition	C++ component instance is running and has a <code>ServiceReference</code>
Input Parameter	
Return	Pointer to a <code>ServiceProxy</code> implementing the callback interface of the <code>ServiceReference</code> . This will be NULL if no callback interface is defined.
Post Condition	A <code>ServiceProxy</code> object for the callback interface of the <code>ServiceReference</code> is constructed.

1364 *[Table 6-19: ServiceReference::getCallback Details](#)*

## 1365 6.4 DomainContext

1366 The ~~following shows the~~ `DomainContext` interface ~~definition is:~~

```

1367 class DomainContext {
1368 public:
1369     virtual ServiceProxyPtr getService(
1370                                     const std::string& serviceURI) const = 0;
1371 };

```

1372 *[Snippet 6-4: DomainContext Class Definition](#)*

1373 **6.4.1 getService**

1374 Non-SCA C++ code uses `getService()` to get a service proxy implementing the interface of a service  
 1375 in an SCA domain.

Precondition	None	
Input Parameter	<code>serviceURI</code>	URI of the Service to get an interface object for
Return	Pointer to a <code>ServiceProxy</code> object implementing the interface of the Service. This will be NULL if <code>serviceURI</code> is not defined in the domain.	
Post Condition	A <code>ServiceProxy</code> object for the Service is constructed.	

1376 *Table 6-20: `DomainContext::getService` Details*

1377 **6.5 SCAException**

1378 The ~~following shows the~~ `SCAException` interface ~~definition is:~~

1379

```

1380 class SCAException : public std::exception {
1381 public:
1382     const char* getEClassName() const;
1383     const char* getMessageText() const;
1384     const char* getFileName() const;
1385     unsigned long getLineNumber() const;
1386     const char* getFunctionName() const;
1387 };
  
```

1388

1389 *The Snippet 6-5: `SCAException` C++ interface has the following member functions (the lass Definition*

1390

1391 ~~The~~ details concerning this class and its derived types are described in ~~the section~~ Error Handling).

1392 **6.5.1 getEClassName**

1393 A C++ component implementation uses `getEClassName()` to get the name of the exception type.

Precondition	C++ component instance is running and has caught an SCA Exception	
Input Parameter		
Return	The type of the exception as a string. e.g. "ServiceUnavailableException"	
Post Condition	No change	

1394 *Table 6-21: `SCAException::getEClassName` Details*

1395 **6.5.2 getMessageText**

1396 A C++ component implementation uses `getMessageText()` to get any message included with the  
 1397 exception.

Precondition	C++ component instance is running and has caught an SCA Exception	
Input Parameter		
Return	The message which the SCA runtime attached to the exception	

Post Condition	No change
----------------	-----------

1398 [Table 6-22: SCAException::getMessageText Details](#)

### 1399 6.5.3 getFileName

1400 A C++ component implementation uses `getFileName()` to get the filename containing the function  
 1401 where the exception occurred.

Precondition	C++ component instance is running and has caught an SCA Exception	
Input Parameter		
Return	The filename within which the exception occurred – Will be an empty string if the filename is not known	
Post Condition	No change	

1402 [Table 6-23: SCAException::getFileName Details](#)

### 1403 6.5.4 getLineNumber

1404 A C++ component implementation uses `getLineNumber()` to get the line number in the source file  
 1405 where the exception occurred.

Precondition	C++ component instance is running and has caught an SCA Exception	
Input Parameter		
Return	The line number at which the exception occurred – Will 0 if the line number is not known	
Post Condition	No change	

1406 [Table 6-24: SCAException::getLineNumber Details](#)

### 1407 6.5.5 getFunctionName

1408 A C++ component implementation uses `getFunctionName()` to get the function name where the  
 1409 exception occurred.

Precondition	C++ component instance is running and has caught an SCA Exception	
Input Parameter		
Return	The function name in which the exception occurred – Will be an empty string if the function name is not known	
Post Condition	No change	

1410 [Table 6-25: SCAException::getFunctionName Details](#)

## 1411 6.6 SCANullPointerException

1412 The ~~following shows the~~ `SCANullPointerException` interface- definition is:

1413

```
1414 class SCANullPointerException : public SCAException {
1415     };
```

1416

1417 | [Snippet 6-6: SCANullPointerException Class Definition](#)

## 1418 | **6.7 ServiceRuntimeException**

1419 | The ~~following shows the~~ ServiceRuntimeException interface- [definition is:](#)

1420

```
1421 | class ServiceRuntimeException : public SCAException {  
1422 |     };
```

1423 | [Snippet 6-7: ServiceRuntimeException Class Definition](#)

## 1424 | **6.8 ServiceUnavailableException**

1425 | The ~~following shows the~~ ServiceUnavailableException interface- [definition is:](#)

1426

```
1427 | class ServiceUnavailablException : public ServiceRuntimeException {  
1428 |     };
```

1429 | [Snippet 6-8: ServiceUnavailableException Class Definition](#)

## 1430 | **6.9 MultipleServicesException**

1431 | The ~~following shows the~~ MultipleServicesException interface- [definition is:](#)

1432

```
1433 | class MultipleServicesException : public ServiceRuntimeException {  
1434 |     };
```

1435 | [Snippet 6-9: MultipleServicesException Class Definition](#)

## 1436 7 C++ Contributions

1437 Contributions are defined in the Assembly specification [ASSEMBLY]. C++ contributions are typically, but  
1438 not necessarily contained in .zip files. In addition to SCDL and potentially WSDL artifacts, C++  
1439 contributions include binary executable files, componentType files and potentially C++ interface headers.  
1440 No additional discussion is needed for header files, but there are **some** additional considerations for  
1441 executable and componentType files ~~discussed in the following sections~~.

### 1442 7.1 Executable files

1443 Executable files containing the C++ implementations for a contribution can be contained in the  
1444 contribution, contained in another contribution or external to any contribution. In some cases, it could be  
1445 desirable to have contributions share an executable. In other cases, an implementation deployment  
1446 policy might dictate that executables are placed in specific directories in a file system.

#### 1447 7.1.1 Executable in contribution

1448 When the executable file containing a C++ implementation is in the same contribution, the *@path*  
1449 attribute of the *implementation.cpp* element is used to specify the location of the executable. The specific  
1450 location of an executable within a contribution is not defined by this specification.

1451  
1452 **The following Snippet 7-1** shows a contribution containing a DLL.

```
1453  
1454 META-INF/  
1455   sca-contribution.xml  
1456 bin/  
1457   autoinsurance.dll  
1458 AutoInsurance/  
1459   AutoInsurance.composite  
1460   AutoInsuranceService/  
1461     AutoInsurance.h  
1462     AutoInsuranceImpl.componentType  
1463 include/  
1464   Customers.h  
1465   Underwriting.h  
1466   RateUtils.h
```

1467 **Snippet 7-1: Contribution Containing a DLL**

1468  
1469 The SCDL for the AutoInsuranceService component **of Snippet 7-1** is:

```
1470  
1471 <component name="AutoInsuranceService">  
1472   <implementation.cpp library="autoinsurance" path="bin/"  
1473     class="AutoInsuranceImpl" />  
1474 </component>
```

1475 **Snippet 7-2: Component Definition Using Implementation in a Common DLL**

#### 1476 7.1.2 Executable shared with other contribution(s) (Export)

1477 If a contribution contains an executable that also implements C++ components found in other  
1478 contributions, the contribution has to export the executable. An executable in a contribution is made  
1479 visible to other contributions by adding an **export.cpp** element to the contribution definition as shown in  
1480 **the following snippet Snippet 7-3**.

1481

```
1482 <contribution>
1483   <deployable composite="myNS:RateUtilities"
1484   <export.cpp name="contribNS:rates" >
1485 </contribution>
```

1486 [Snippet 7-3: Exporting a Contribution](#)

1487

1488 It is also possible to export only a subtree of a contribution. ~~If~~For a contribution ~~contains the following~~:

1489

```
1490 META-INF/
1491   sca-contribution.xml
1492 bin/
1493   rates.dll
1494 RateUtilities/
1495   RateUtilities.composite
1496   RateUtilitiesService/
1497     RateUtils.h
1498     RateUtilsImpl.componentType
```

1499 [Snippet 7-4: Contribution with a Subdirectory to be Shared](#)

1500

1501 An export of the form:

1502

```
1503 <contribution>
1504   <deployable composite="myNS:RateUtilities"
1505   <export.cpp name="contribNS:ratesbin" path="bin/" >
1506 </contribution>
```

1507 [Snippet 7-5: Exporting a Subdirectory of a Contribution](#)

1508

1509 only makes the contents of the bin directory visible to other contributions. By placing all of the executable  
1510 files of a contribution in a single directory and exporting only that directory, the amount of information  
1511 available to a contribution that uses the exported executable files is limited. This is considered a best  
1512 practice.

### 1513 7.1.3 Executable outside of contribution (Import)

1514 When the executable that implements a C++ component is located outside of a contribution, the  
1515 contribution MUST has to import the executable. If the executable is located in another contribution, the  
1516 **import.cpp** element of the contribution definition uses a *@location* attribute that identifies the name of the  
1517 export as defined in the contribution that defined the export as shown in ~~the following snippet~~ [Snippet 7-6](#).

1518

```
1519 <contribution>
1520   <deployable composite="myNS:Underwriting"
1521   <import.cpp name="rates" location="contribNS:rates">
1522 </contribution>
```

1523 [Snippet 7-6: Contribution with an Import](#)

1524

1525 The SCDL for the UnderwritingService component of [Snippet 7-6](#) is:

1526

```
1527 <component name="UnderwritingService">
1528   <implementation.cpp library="rates" path="rates:bin/"
1529   class="UnderwritingImpl" />
1530 </component>
```

1531 [Snippet 7-7: Component Definition Using Implementation in an External DLL](#)

1532

1533 If the executable is located in the file system, the `@location` attribute identifies the location in the files  
1534 system used as the root of the import as shown in [this snippet Snippet 7-8](#).

1535

```
1536 <contribution>  
1537   <deployable composite="myNS:CustomerUtilities"  
1538     <import.cpp name="usr-bin" location="/usr/bin/" >  
1539 </contribution>
```

1540 [Snippet 7-8: Component Definition Using Implementation in a File System](#)

## 1541 7.2 componentType files

1542 As stated in [section 2-5](#) Component Type and Component, each component implemented in C++ has a  
1543 corresponding componentType file. This componentType file is, by default, located in the root directory of  
1544 the composite containing the component or a subdirectory of the composite root with the name  
1545 `<implementation class>.componentType`, as shown in [the following example Snippet 7-9](#).

1546

```
1547 META-INF/  
1548   sca-contribution.xml  
1549 bin/  
1550   autoinsurance.dll  
1551 AutoInsurance/  
1552   AutoInsurance.composite  
1553   AutoInsuranceService/  
1554     AutoInsurance.h  
1555     AutoInsuranceImpl.componentType
```

1556 [Snippet 7-9: Contribution with ComponentType](#)

1557

1558 The SCDL for the AutoInsuranceService component [of Snippet 7-9](#) is:

1559

```
1560 <component name="AutoInsuranceService">  
1561   <implementation.cpp library="autoinsurance" path="bin/"  
1562     class="AutoInsuranceImpl" />  
1563 </component>
```

1564 [Snippet 7-10: Component Definition with Local ComponentType](#)

1565

1566 Since there is a one-to-one correspondence between implementations and componentTypes, when an  
1567 implementation is shared between contributions, it is desirable to also share the componentType file.  
1568 ComponentType files can be exported and imported in the same manner as executable files. The  
1569 location of a `.componentType` file can be specified using the `@componentType` attribute of the  
1570 `implementation.cpp` element.

1571

```
1572 <component name="UnderwritingService">  
1573   <implementation.cpp library="rates" path="rates:bin/"  
1574     class="UnderwritingImpl" componentType="rates:types/UnderwritingImpl"  
1575   />  
1576 </component>
```

1577 [Snippet 7-11: Component Definition with Imported ComponentType](#)



## 1578 7.3 C++ Contribution Extensions

### 1579 7.3.1 Export.cpp

1580 | The following snippet Snippet 7-12 shows the pseudo-schema for the C++ export element used to make  
1581 an executable or componentType file visible outside of a contribution.

```
1582  
1583 | <?xml version="1.0" encoding="ASCII"?>  
1584 | <!-- export.cpp schema snippet -->  
1585 | <export.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"  
1586 |     name="QName" path="string"? >
```

1587 | [Snippet 7-12: Pseudo-schema for C++ Export Element](#)

1588

1589 The **export.cpp** element has the following **attributes**:

- 1590 • **name : QName (1..1)** – name of the export. The @name attribute of a <export.cpp/> element MUST  
1591 be unique amongst the <export.cpp/> elements in a domain. [CPP70001]
- 1592 • **path : string (0..1)** – path of the exported executable relative to the root of the contribution. If not  
1593 present, the entire contribution is exported.

### 1594 7.3.2 Import.cpp

1595 | The following snippet Snippet 7-13 shows the pseudo-schema for the C++ import element used to  
1596 reference an executable or componentType file that is outside of a contribution.

```
1597  
1598 | <?xml version="1.0" encoding="ASCII"?>  
1599 | <!-- import.cpp schema snippet -->  
1600 | <import.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"  
1601 |     name="QName" location="string" >
```

1602 | [Snippet 7-13: Pseudo-schema for C++ Import Element](#)

1603

1604 The **import.cpp** element has the following **attributes**:

- 1605 • **name : QName (1..1)** – name of the import. The @name attribute of a <import.cpp/> child element of  
1606 a <contribution/> MUST be unique amongst the <import.cpp/> elements in of that contribution.  
1607 [CPP70002]
- 1608 • **location : string (1..1)** – either the QName of an export or a file system location. If the value does not  
1609 match an export name it is taken as an absolute file system path.

---

## 8 C++ Interfaces

A service interface can be defined by a C++ class which has only pure virtual public member functions. The class might additionally have private or protected member functions, but these are not part of the service interface.

When mapping a C++ interface to WSDL or when comparing two C++ interfaces for compatibility, as defined by the Assembly specification [ASSEMBLY], it is necessary for an SCA implementation to determine the signature (return type, name, and the names and types of the parameters) of every member function of the class defining the service interface. An SCA implementation MUST translate a class to tokens as part of conversion to WSDL or compatibility testing. [CPP80001] Macros and typedefs in member function declarations might lead to portability problems. Complete member function declarations within a macro are discouraged. The processing of typedefs needs to be aware of the types that impact mapping to WSDL (see Table 9-1 and Table 9-2)

### 7.48.1 Types Supported in Service Interfaces

A service interface can support a restricted set of the types available to a C++ programmer. This section summarizes the valid types that can be used.

#### 1.2 Local service

The return type and types of the parameters of a member function of a local service interface MUST be one of:

- Any of the C++ primitive types (for example, `int`, `short`, `char`). In this case the data will be passed by value as is normal for C++.
- Pointers to any of the C++ primitive types (for example, `int *`, `short *`, `char *`).
- The `const` keyword can be used for any pointer to a C++ primitive type (for example `const char *`). If this is used on a parameter then the destination can not change the value.
- C++ class. The class will be passed by value as is normal for C++.
- Pointer to a C++ class. A pointer will be passed to the destination which can then modify the original contents.
- `DataObjectPtr`. An SDO pointer. This will be passed by reference.
- References to C++ classes (passed by reference). [CPP80001]

#### 1.3 Remotable service

Not all service interfaces support the complete set of the types available in C++.

#### 8.1.1 Local Service

Any fundamental or compound type defined by C++ can be used in the interface of a local service.

#### 8.1.2 Remotable Service

For a remotable service being called by another service the data exchange semantics is by-value. The return type and types of the parameters of a member function of a remotable service interface MUST be one of:

- Any of the C++ types specified in Simple Content Binding. These types may be passed by-value, by-reference, or by-pointer. Unless the member function and client indicate that they allow by-reference semantics (see `AllowsPassByReference`), a copy will be explicitly created by the runtime for any parameters passed by-reference or by-pointer.

- 1650 • An SDO `DataObjectPtr` instance. This type may be passed by-value, by-reference, or by-pointer.  
1651 Unless the member function and client indicate that they allow by-reference semantics (see  
1652 `AllowsPassByReference`), a deep-copy of the `DataObjectPtr` will be created by the runtime for any  
1653 parameters passed by-value, by-reference, or by-pointer. When by-reference semantics are allowed,  
1654 the `DataObjectPtr` itself will be passed. [CPP80002]

1655 **7.58.2 Unless the interface is marked as allowing pass by reference**  
1656 **semantics, the behavior of the following are not defined: Header**  
1657 **Files**

- 1658 • ~~Pointers.~~  
1659 • ~~References.~~

---

1660 **2 ~~Restrictions on C++ header files~~**

1661 ~~A C++ header file that is used to describe an interface has some restrictions.~~

1662 • A C++ header file used to define an interface MUST declare at least one class with:

1663 ~~•~~ At least one public member function.

1664 ~~•~~ All public member functions are pure virtual. [CPP90004]

1665

1666 ~~A C++ header file used to define an interface MUST NOT use the following constructs:~~

1667 ~~•~~ Macros

1668 ~~•~~ Inline member functions

1669 ~~•~~ Friend classes [CPP90002CPP80003]

## 89 WSDL to C++ and C++ to WSDL Mapping

1670

1671 The SCA Client and Implementation Model for C++ applies the WSDL to Java and Java to WSDL  
1672 mapping rules (augmented for C++) as defined by the JAX-WS specification [JAXWS21] for generating  
1673 remotable C++ interfaces from WSDL portTypes and vice versa. Use of the JAX-WS specification as a  
1674 guideline for WSDL to C++ and C++ to WSDL mappings does not imply that any support for the Java  
1675 language is mandated by this specification.

1676

1677 For the purposes of the Java-C++ to-WSDL mapping algorithm, the interface is treated as if it had a  
1678 @WebService annotation on the class, even if it doesn't. For the WSDL-to-Java-C++ mapping, the  
1679 generated @WebService annotation implies that the interface is @Remotable.

1680

1681 For the mapping from C++ types to XML schema types SCA supports the SDO 2.1 [SDO21][SDO21]  
1682 mapping. A detailed mapping of C++ to WSDL types and WSDL to C++ types is covered in section SDO  
1683 Data Binding.

1684

1685 ~~The following limitations apply to items in the JAX-WS WSDL to Java and Java to WSDL mapping that are~~  
1686 ~~not supported:~~

- 1687 • JAX-WS style external binding ~~files are not supported~~. (See JAX-WS Sec. 2)
- 1688 • MIME binding ~~is not supported~~. (See JAX-WS Sec. 2.1.1)
- 1689 • Holder classes ~~are not supported~~. (See JAX-WS Sec. 2.3.3)
- 1690 • Asynchronous mapping ~~is not supported~~. (See JAX-WS Sec. 2.3.4)
- 1691 • Generation of Service classes from WSDL ~~is not supported~~. (See JAX-WS Sec. 2.7)
- 1692 • Generation of WSDL from Service implementation classes ~~is not supported~~. (See JAX-WS Sec. 3.3)
- 1693 • Templates ~~are not supported~~ when converting from C++ to WSDL (See JAX-WS Sec. 3.9)

1694

1695 ~~The following general~~ General rules apply to for the application of JAX-WS to C++.

- 1696 • References to Java are considered references to C++.
- 1697 • References to Java classes are considered references to C++ classes.
- 1698 • References to Java methods are considered references to C++ member functions.
- 1699 • References to Java interfaces are considered references to C++ classes which only define pure  
1700 virtual member functions.
- 1701 ~~• For the purposes of the C++ to WSDL mapping algorithm, a C++ class with only pure virtual functions~~  
1702 ~~and no state is treated as if it had a @WebService annotation on the class. All default values are~~  
1703 ~~assumed for the @WebService annotation.~~

1704

1705 Major divergences from JAX-WS:

- 1706 • Algorithms for converting WSDL namespaces to C++ namespaces (and vice-versa).
- 1707 • Mapping of WSDL faults to C++ exceptions and vice-versa.
- 1708 • Managing of data bindings.

### 8.19.1 Augmentations for WSDL to C++ Mapping

1709

1710 An SCA implementation MUST map a WSDL portType to a remotable C++ interface definition.  
1711 [CPP100009]

### 1712 **8.1.19.1.1 Mapping WSDL targetNamespace to a C++ namespace**

1713 Since C++ does not define a standard convention for the use of namespaces, the SCA specification does  
1714 not define an implicit mapping of WSDL targetNamespaces to C++ namespaces. A WSDL file might  
1715 define a namespace using the <sca:namespace> WSDL extension, otherwise all C++ classes MUST be  
1716 placed in a default namespace as determined by the implementation. Implementations SHOULD provide  
1717 a mechanism for overriding the default namespace. [CPP100001]

### 1718 **8.1.29.1.2 Mapping WSDL Faults to C++ Exceptions**

1719 WSDL operations that specify one or more <wsdl:fault> elements will produce a C++ member function  
1720 that is annotated with an @WebThrows annotation listing a C++ exception class associated with each  
1721 <wsdl:fault>.

1722  
1723 The C++ exception class associated with a fault will be generated based on the message that is  
1724 associated with the <wsdl:fault> element, and in particular with the global element that the  
1725 wsdl:fault/wsdl:message/@part indicates.

```
1727 <FaultException>(const char* message, const <FaultInfo>& faultInfo);  
1728 <FaultInfo> getFaultInfo() const;
```

1729 *Snippet 9-1: Fault Exception Class Member Functions*

1730  
1731 Where <FaultException> is the name of the generated exception class, and where <FaultInfo> is  
1732 the name of the C++ type representing the fault's global element type.

### 1733 **8.1.2.19.1.2.1 Multiple Fault References**

1734 If multiple operations within the same portType indicate that they throw faults that reference the same  
1735 global element, an SCA implementation MUST generate a single C++ exception class with each C++  
1736 member function referencing this class in its @WebThrows annotation. [CPP100002]

### 1737 **8.1.39.1.3 Mapping of in, out, in/out parts to C++ member function 1738 parameters**

1739 C++ diverges from the JAX-WS specification in it's handling of some parameter types, especially around  
1740 how passing of out and in/out parameters are handled in the context of C++'s various pass-by styles.  
1741 The following outlines an updated mapping for use with C++.

- 1742 • For unwrapped messages, an SCA implementation MUST map:
- 1743 – **in** - the message part to a member function parameter, passed by const-reference.
  - 1744 – **out** - the message part to a member function parameter, passed by reference, or to the member  
1745 function return type, returned by-value.
  - 1746 – **in/out** - the message part to a member function parameter, passed by reference. [CPP100003]
- 1747
- 1748 • For wrapped messages, an SCA implementation MUST map:
- 1749 – **in** - the wrapper child to a member function parameter, passed by const-reference.
  - 1750 – **out** - the wrapper child to a member function parameter, passed by reference, or to the member  
1751 function return type, returned by-value.
  - 1752 – **in/out** - the wrapper child to a member function parameter, passed by reference. [CPP100004]
- 1753

## 1754 **8.29.2 Augmentations for C++ to WSDL Mapping**

1755 Where annotations are discussed as a means for an application to control the mapping to WSDL, an  
1756 implementation-specific means of controlling the mapping can be used instead.

1757 An SCA implementation MUST map a C++ interface definition to WSDL as if it has a @WebService  
1758 annotation with all default values on the class. [CPP100010]

1759 An application can customize the name of the portType and port using the @WebService annotation.

### 1760 **8.2.19.2.1 Mapping C++ namespaces to WSDL namespaces**

1761 Since C++ does not define a standard convention for the use of namespaces, the SCA specification does  
1762 not define an implicit mapping of C++ namespaces to WSDL namespace URIs. The default  
1763 targetNamespace is defined by the implementation. An SCA implementation SHOULD provide a  
1764 mechanism for overriding the default targetNamespace. [CPP100005]

### 1765 **8.2.29.2.2 Parameter and return type classification**

1766 The classification of parameters and return types in C++ are determined based on how the value is  
1767 passed into the function.

1768  
1769 An SCA implementation MUST map a method's return type as an out parameter, a parameter passed by-  
1770 reference or by-pointer as an in/out parameter, and all other parameters, including those passed by-  
1771 const-reference as in parameters. [CPP100006]

1772  
1773 An application can customize parameter classification using the @WebParam annotation.

### 1774 **8.2.39.2.3 C++ to WSDL Type Conversion**

1775 C++ types are mapped to WSDL and schema types based on the mapping described in Section Simple  
1776 Content Binding.

### 1777 **8.2.49.2.4 Service-specific Exceptions**

1778 C++ classes that define a web service interface can indicate which faults they might throw using the  
1779 @WebThrows annotation. @WebThrows lists the names of each C++ class that might be thrown as a  
1780 fault from a particular member function. An SCA implementation MUST ensure each class that is  
1781 referenced from an @WebThrows annotation MUST itself have a @WebFault annotation that associates  
1782 the fault with a particular global element that will be associated with the fault message. [CPP100007]  
1783 By default, no exceptions are mapped to operation faults.

## 1784 **8.39.3 SDO Data Binding**

### 1785 **8.3.19.3.1 Simple Content Binding**

1786 The translation of XSD simple content types to C++ types follows the convention defined in the SDO  
1787 specification. The following table Table 9-1 summarizes that mapping as it applies to SCA services.

1788

<i>XSD Schema Type</i> →	<i>C++ Type</i>	→ <i>XSD Schema Type</i>
anySimpleType	std::string	string
anyType	commonj::sdo::DataObject	anyType
anyURI	std::string	string

base64Binary	char*	string
boolean	bool	boolean
byte	int8_t	byte
date	std::string	string
dateTime	std::string	string
decimal	std::string	string
double	double	double
duration	std::string	string
ENTITIES	std::list<std::string>	IDREFS
ENTITY	std::string	string
float	float	float
gDay	std::string	string
gMonth	std::string	string
gMonthDay	std::string	string
gYear	std::string	string
gYearMonth	std::string	string
hexBinary	char*	string
ID	std::string	string
IDREF	std::string	string
IDREFS	std::list<std::string>	IDREFS
int	int32_t	int
integer	std::string	string
language	std::string	string
long	int64_t	long
Name	std::string	string
NCName	std::string	string
negativeInteger	std::string	string
NMTOKEN	std::string	string
NMTOKENS	std::list<std::string>	IDREFS
nonNegativeInteger	std::string	string
nonPositiveInteger	std::string	string
normalizedString	std::string	string
NOTATION	std::string	string



positiveInteger	std::string	string
QName	std::string	string
short	int16_t	short
string	std::string	string
time	std::string	string
token	std::string	string
unsignedByte	uint8_t	unsignedByte
unsignedInt	uint32_t	unsignedInt
unsignedLong	uint64_t	unsignedLong
unsignedShort	uint16_t	unsignedShort

1789 *Table 49-1: XSD simple type to C++ type mapping*

1790

1791 [Table 9-2 defines the mapping of C++ types to XSD schema types that are not covered in Table 9-1.](#)

1792

<b>C++ Type →</b>	<b>XSD Schema Type</b>
char	string
wchar_t	string
signed char	byte
unsigned char	unsignedByte
short	short
unsigned short	unsignedShort
int	int
unsigned int	unsignedInt
long	long
unsigned long	unsignedLong
long long	long
unsigned long long	unsignedLong
wchar_t *	string
long double	decimal
time_t	dateTime
struct tm	dateTime

1793 *Table 29-2: C++ type to XSD type mapping*

1794

1795 The C++ standard does not define value ranges for integer types so it is possible that on a platform  
1796 parameters or return values could have values that are out of range for the default XSD schema type. In  
1797 these circumstances, the mapping would need to be customized, using @WebParam or @WebResult if  
1798 supported, or some other implementation-specific mechanism.

1799

1800 An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.  
1801 [CPP100008]

### 1802 **8.3.29.3.2 Complex Content Binding**

1803 Any XSD complex types are mapped to an instance of an SDO DataObject.

---

## 910 Conformance

The XML schema pointed to by the RDDDL document at the SCA namespace URI, defined by the Assembly specification [ASSEMBLY]{CPP110004}

and extended by this specification, are considered to be authoritative and take precedence over the XML schema in this document.

The XML schema pointed to by the RDDDL document at the SCA C++ namespace URI, defined by this specification, is considered to be authoritative and takes precedence over the XML schema in this document.

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

An SCA implementation MUST reject a composite file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-cpp-1.1.xsd> or <http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-cpp-1.1.xsd>. [CPP110002]

[CPP110001]

An SCA implementation MUST reject a componentType file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-cpp-1.1.xsd>. [CPP110003]

[CPP110002]

An SCA implementation MUST reject a contribution file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200912/sca-contribution-cpp-1.1.xsd>. [CPP110003]

An SCA implementation MUST reject a WSDL file that does not conform to <http://docs.oasis-open.org/opencsa/sca-c-cpp/cpp/200901/sca-wsdl-ext-cpp-1.1.xsd>. [CPP110004]

### 9.110.1 Conformance Targets

The conformance targets of this specification are:

- **SCA implementations**, which provide a **runtime** for SCA components and potentially **tools** for authoring SCA artifacts, component descriptions and/or runtime operations.
- **SCA documents**, which describe SCA artifacts, and specific **elements** within these documents.
- **C++ component implementations**, which execute under the control of an SCA runtime.
- **C++ files**, which define SCA service interfaces and implementations.
- **WSDL files**, which define SCA service interfaces.

### 9.210.2 SCA Implementations

An implementation conforms to this specification if it meets ~~the following~~these conditions:

1. It MUST conform to the SCA Assembly Model Specification [ASSEMBLY] and the SCA Policy Framework [POLICY].
2. It MUST comply with all statements in [Table F-1](#) and [Table F-4](#) related to an SCA implementation, notably all mandatory statements have to be implemented.
3. It MUST implement the SCA C++ API defined in section C++ API.
4. It MUST implement the mapping between C++ and WSDL 1.1 [WSDL11] defined in WSDL to C++ and C++ to WSDL Mapping.

- 1845 5. It MUST support <interface.cpp/> and <implementation.cpp/> elements as defined in Component  
1846 Type and Component in composite-, ~~and~~ componentType ~~and constrainingType~~ documents.
- 1847 6. It MUST support <export.cpp/> and <import.cpp/> elements as defined in C++ Contributions in  
1848 contribution documents.
- 1849 7. It MAY support source file annotations as defined in C++ SCA Annotations, C++ SCA Policy  
1850 Annotations and C++ WSDL Mapping Annotations. If source file annotations are supported, the  
1851 implementation MUST comply with all statements in Table F-2 related to an SCA implementation,  
1852 notably all mandatory statements in that section have to be implemented.
- 1853 8. It MAY support WSDL extensions as defined in WSDL C++ Mapping Extensions. If WSDL  
1854 extensions are supported, the implementation MUST comply with all statements in Table F-3 related  
1855 to an SCA implementation, notably all mandatory statements in that section have to be implemented.

### 1856 **9.310.3 SCA Documents**

1857 An SCA document conforms to this specification if it meets ~~the following conditions~~ these condition:

1858 ~~9.1.~~ It MUST conform to the SCA Assembly Model Specification [**ASSEMBLY**] and, if appropriate, the  
1859 SCA Policy Framework [**POLICY**].

1860 ~~10.2.~~ If it is a composite document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd)  
1861 [open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd) and [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-implementation-cpp-1.1.xsd)  
1862 [open.org/opencsa/sca/200903200912/sca-implementation-cpp-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-implementation-cpp-1.1.xsd) schema and MUST comply  
1863 with the additional constraints on the document contents as defined in [Table F-1](#).

1864

1865 If it is a componentType ~~or constrainingType~~ document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd)  
1866 [open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd) schema and MUST comply with the  
1867 additional constraints on the document contents as defined in [Table F-1](#).

1868

1869 If it is a contribution document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contribution-cpp-1.1.xsd)  
1870 [open.org/opencsa/sca/200903200912/sca-contribution-cpp-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contribution-cpp-1.1.xsd) schema and MUST comply with  
1871 the additional constraints on the document contents as defined in [Table F-1](#).

### 1872 **9.410.4 C++ Files**

1873 A C++ files conforms to this specification if it meets the ~~following conditions~~ condition:

- 1874 1. It MUST comply with all statements in [Table F-1](#), [Table F-2](#) ~~and~~ [Table F-4](#) related to C++ contents  
1875 and annotations-, notably all mandatory statements have to be satisfied.

### 1876 **9.510.5 WSDL Files**

1877 A WSDL file conforms to this specification if it meets ~~the following~~ these conditions:

- 1878 1. It is a valid WSDL 1.1 [**WSDL11**] document.
- 1879 2. It MUST comply with all statements in [Table F-1](#), [Table F-3](#) ~~and~~ [Table F-4](#) related to WSDL contents  
1880 and extensions, notably all mandatory statements have to be satisfied.

1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921

## A C++ SCA Annotations

To allow developers to define SCA related information directly in source files, without having to separately author SCDL files, a set of annotations is defined. If annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as described. The SCA runtime MUST only process the SCDL files and not the annotations. [CPPA0001]

The annotations are defined as C++ comments in interface and implementation header files, for example:

```
// @Scope("stateless")
```

*Snippet A-1: Example Annotation*

### A.1 Application of Annotations to C++ Program Elements

In general an annotation immediately precedes the program element it applies to. If multiple annotations apply to a program element, all of the annotations SHOULD be in the same comment block. [CPPA0002]

- Class

The annotation immediately precedes the class.

Example:

```
// @Scope("composite")  
class LoanServiceImpl : public LoanService {  
    ...  
};
```

*Snippet A-2: Example Class Annotation*

- Member function

The annotation immediately precedes the member function.

Example:

```
class LoanService  
{  
public:  
    // @OneWay  
    virtual void reportEvent(int eventId) = 0;  
    ...  
};
```

*Snippet A-3: Example Member Function Annotation*

- Data Member

The annotation immediately precedes the data member.

Example:

```
// @Property(name="loanType", type="xsd:int")  
long loanType;
```

*Snippet A-4: Example Data Member Annotation*

Annotations follow normal inheritance rules. An annotation on a base class or any element of a base class applies to any classes derived from the base class.

## 1922 **A.1A.2 Interface Header Annotations**

1923 This section lists the annotations that can be used in the header file that defines a service interface.

### 1924 **A.1.1A.2.1 @Interface**

1925 Annotation used to indicate a class defines an interface when multiple classes exist in a header file. ~~An~~  
1926 ~~SCA implementation MUST treat a class with an @WebService annotation specified as if an @Interface~~  
1927 ~~annotation was specified. [CPPA0003]~~

1928

1929

1930 **Corresponds to:** @class attribute of an *interface.cpp* element.

1931

1932 **Format:**

```
1933 // @Interface
```

1934

1935 *Snippet A-5: @Interface Annotation Format*

1936 **Applies to:** Class

1937

1938 **Example:**

1939 Interface header:

```
1940 // @Interface  
1941 class LoanService {  
1942     ...  
1943 };
```

1944

1945 Service definition:

```
1946 <service name="LoanService">  
1947     <interface.cpp header="LoanService.h" class="LoanService" />  
1948 </service>
```

1949 *Snippet A-6: Example of @Interface Annotation*

### 1950 **A.1.2A.2.2 @Remotable**

1951 Annotation on service interface class to indicate that a service is remotable. ~~and implies an @Interface~~  
1952 ~~annotation applies as well.~~ An SCA implementation MUST treat a class with an @WebService annotation  
1953 specified as if a @Remotable annotation was specified. [CPPA0003]

1954

1955 **Corresponds to:** @remotable="true" attribute of an *interface.cpp* element.

1956

1957 **Format:**

```
1958 // @Remotable
```

1959 *Snippet A-7: @Remotable Annotation Format*

1960 The default is **false** (not remotable).

1961

1962 **Applies to:** Class

1963

1964 Example:

1965 Interface header:

```
1966 // @Remotable
1967 class LoanService {
1968     ...
1969 };
```

1970

1971 Service definition:

```
1972 <service name="LoanService">
1973     <interface.cpp header="LoanService.h" remotable="true" />
1974 </service>
```

1975 [Snippet A-8: Example of @Remotable Annotation](#)

### 1976 **A-1.3A.2.3 @Callback**

1977 Annotation on a service interface class to specify the callback interface.

1978

1979 **Corresponds to:** `@callbackHeader` and `@callbackClass` attributes of an `interface.cpp` element.

1980

1981 **Format:**

```
1982 // @Callback(header="headerName", class="className")
```

1983 [Snippet A-9: @Callback Annotation Format](#)

1984 where

- 1985 • **headerName : (1..1)** – is the name of the header defining the callback service interface.
- 1986 • **className : (0..1)** – is the name of the class for the callback interface.

1987

1988 **Applies to:** Class

1989

1990 Example:

1991 Interface header:

```
1992 // @Callback(header="MyServiceCallback.h", class="MyServiceCallback")
1993 class MyService {
1994     public:
1995         virtual void someFunction( unsigned int arg ) = 0;
1996 };
```

1997

1998 Service definition:

```
1999 <service name="MyService">
2000     <interface.cpp header="MyService.h"
2001         callbackHeader="MyServiceCallback.h"
2002         callbackClass="MyServiceCallback" />
2003 </service>
```

2004 [Snippet A-10: Example of @Callback Annotation](#)

### 2005 **A-1.4A.2.4 @OneWay**

2006 Annotation on a [servicean](#) interface member function to indicate the member function is one way. The  
2007 `@OneWay` annotation also affects the representation of a service in WSDL, see `@OneWay`.

2008

2009 | **Corresponds to:** `@oneWay="true"` attribute of `function` element of an `interface.cpp` element.

2010

2011 | **Format:**

```
2012 | // @OneWay
```

2013 | [Snippet A-11: @OneWay Annotation Format](#)

2014 | The default is **false** (not OneWay).

2015

2016 | **Applies to:** Member function

2017

2018 | **Example:**

2019 | **Interface header:**

```
2020 | class LoanService
2021 | {
2022 | public:
2023 | // @OneWay
2024 |     virtual void reportEvent(int eventId) = 0;
2025 | ...
2026 | };
```

2027

2028 | **Service definition:**

```
2029 | <service name="LoanService">
2030 |     <interface.cpp header="LoanService.h">
2031 |         <function name="reportEvent" oneWay="true" />
2032 |     </interface.cpp>
2033 | </service>
```

2034 | [Snippet A-12: Example of @OneWay Annotation](#)

## 2035 | **A.2.5 @Function**

2036 | Annotation on a interface member function to modify its representation in an SCA interface. An SCA  
2037 | implementation MUST treat a member function with a @WebFunction annotation specified as if  
2038 | @Function was specified with the operationName value of the @WebFunction annotation used as the  
2039 | name value of the @Function annotation and the exclude value of the @WebFunction annotation used as  
2040 | the exclude valued of the @Function annotation. [CPPA0004]

2041 | **Corresponds to:** `function` or `callbackFunction` element of an `interface.cpp` element. If the class the  
2042 | function is a member of is being processed because it was identified via either a combination of  
2043 | `interface.cpp/@callbackHeader` and `interface.cpp/@callbackClass` or a `@Callback` annotation, then the  
2044 | `@Function` annotation corresponds to a `callbackFunction` element, otherwise it corresponds to a `function`  
2045 | element.

2046 | **Format:**

```
2047 | // @Function(name="operationName", exclude="true")
```

2048 | [Snippet A-13: @Function Annotation Format](#)

2049 | **where**

- 2050 | • **name : NCName (0..1)** – specifies the name of the operation. The default operation name is the  
2051 | function name.
- 2052 | • **exclude : boolean (0..1)** – specifies whether this member function is to be excluded from the SCA  
2053 | interface. Default is **false**.

2054 | **Applies to:** Member function

2055 | **Example:**



2056 **Interface header:**  
2057 `class LoanService`  
2058 `{`  
2059 `public:`  
2060 `...`  
2061 `// @Function(exclude="true")`  
2062 `virtual void reportEvent(int eventId) = 0;`  
2063 `...`  
2064 `};`

2065  
2066 **Service definition:**  
2067 `<service name="LoanService">`  
2068 `<interface.cpp header="LoanService.h">`  
2069 `<function name="reportEvent" exclude="true" />`  
2070 `</interface.cpp>`  
2071 `</service>`

2072 *Snippet A-14: Example of @Function Annotation*

## 2073 **A.2A.3 Implementation Header Annotations**

2074 This section lists the annotations that can be used in the header file that defines a service  
2075 implementation.

### 2076 **A.2.1A.3.1 @ComponentType**

2077 Annotation used to indicate which class implements a componentType when multiple classes exist in an  
2078 implementation file.

2079  
2080 **Corresponds to:** @class attribute of an *implementation.cpp* element.

2081  
2082 **Format:**

2083 `// @ComponentType`

2084  
2085 *Snippet A-15: @ComponentType Annotation Format*

2086 **Applies to:** Class

2087  
2088 **Example:**

2089 **Implementation header:**  
2090 `// @ComponentType`  
2091 `class LoanServiceImpl : public LoanService {`  
2092 `...`  
2093 `};`

2094  
2095 **Component definition:**  
2096 `<component name="LoanService">`  
2097 `<implementation.cpp library="loan" class="LoanServiceImpl"`  
2098 `class="LoanServiceImpl" />`  
2099 `</component>`

2100 *Snippet A-16: Example of @ComponentType Annotation*

## 2101 **A-2.2A.3.2 @Scope**

2102 Annotation on a service implementation class to indicate the scope of the service.

2103

2104 **Corresponds to:** `@scope` attribute of an *implementation.cpp* element.

2105

2106 **Format:**

```
2107 // @Scope("value")
```

2108 [Snippet A-17: @Scope Annotation Format](#)

2109 where

- 2110 • **value** : [ **stateless** | **composite** ] (1..1) – specifies the scope of the implementation. The default value  
2111 is **stateless**.

2112

2113 **Applies to:** Class

2114

2115 Example:

2116 Implementation header:

```
2117 // @Scope("composite")  
2118 class LoanServiceImpl : public LoanService {  
2119     ...  
2120 };
```

2121

2122 Component definition:

```
2123 <component name="LoanService">  
2124     <implementation.cpp library="loan" class="LoanServiceImpl"  
2125         scope="composite" />  
2126 </component>
```

2127 [Snippet A-18: Example of @Scope Annotation](#)

## 2128 **A-2.3A.3.3 @EagerInit**

2129 Annotation on a service implementation class to indicate the implantation is to be instantiated when its  
2130 containing component is started.

2131

2132 **Corresponds to:** `@eagerInit="true"` attribute of an *implementation.cpp* element.

2133

2134 **Format:**

```
2135 // @EagerInit
```

2136 [Snippet A-19: @EagerInit Annotation Format](#)

2137 The default is **false** (the service is initialized lazily).

2138

2139 **Applies to:** Class

2140

2141 Example:

2142 Implementation header:

```
2143 // @EagerInit
```

```
2144 class LoanServiceImpl : public LoanService {
2145     ...
2146 };
```

2147

2148 Component definition:

```
2149 <component name="LoanService">
2150     <implementation.cpp library="loan" class="LoanServiceImpl"
2151         eagerInit="true" />
2152 </component>
```

2153 [Snippet A-20: Example of @EagerInit Annotation](#)

### 2154 **A-2.4A.3.4 @AllowsPassByReference**

2155 Annotation on service implementation class or member function to indicate that a service or member  
2156 function allows pass by reference semantics.

2157

2158 **Corresponds to:** `@allowsPassByReference="true"` attribute of an `implementation.cpp` element or a  
2159 `function` child element of an `implementation.cpp` element.

2160

2161 **Format:**

```
2162 // @AllowsPassByReference
```

2163 [Snippet A-21: @AllowsPassByReference Annotation Format](#)

2164 The default is **false** (the service does not allow by reference parameters).

2165

2166 **Applies to:** Class or Member function

2167

2168 Example:

Implementation header:

```
2169 // @AllowsPassByReference
2170 class LoanService {
2171     ...
2172 };
2173
```

2174

2175 Component definition:

```
2176 <component name="LoanService">
2177     <implementation.cpp library="loan" class="LoanServiceImpl"
2178         allowsPassByReference="true" />
2179 </component>
```

2180 [Snippet A-22: Example of @AllowsPassByReference Annotation](#)

### 2181 **A-2.5A.3.5 @Property**

2182 Annotation on a service implementation class data member to define a property of the service.

2183

2184 **Corresponds to:** `property` element of a `componentType` element.

2185

2186 **Format:**

```
2187 // @Property(name="propertyName", type="typeName"
2188 //           default="defaultValue", required="true")
```

2189 [Snippet A-23: @Property Annotation Format](#)

2190 where

- 2191 • **name** : **NCName (0..1)** - specifies the name of the property. If name is not specified the property  
2192 name is taken from the name of the following data member.
- 2193 • **type** : **QName (0..1)** - specifies the type of the property. If not specified the type of the property is  
2194 based on the C++ mapping of the type of the following data member to an xsd type as defined in  
2195 SDO Data Binding. If the data member is an array, then the property is many-valued.
- 2196 • **required** : **boolean (0..1)** - specifies whether a value has to be set in the component definition for this  
2197 property. Default is *false*
- 2198 • **default** : **<type> (0..1)** - specifies a default value and is only needed if *required* is *false*,

2199

2200 **Applies to:** DataMember

2201

2202 Example:

2203 Implementation:

```
2204 // @Property(name="loanType", type="xsd:int")  
2205 long loanType;
```

2206

2207 Component Type definition:

```
2208 <componentType ... >  
2209   <service ... />  
2210   <property name="loanType" type="xsd:int" />  
2211 </componentType>
```

2212 [Snippet A-24: Example of @Property Annotation](#)

## 2213 **A-2.6A.3.6 @Reference**

2214 Annotation on a service implementation class data member to define a reference of the service.

2215

2216 **Corresponds to:** *reference* element of a *componentType* element.

2217

2218 **Format:**

```
2219 // @Reference(name="referenceName", interfaceHeader="LoanService.h",  
2220 //           interfaceClass="LoanService", required="true")
```

2221 [Snippet A-25: @Reference Annotation Format](#)

2222 where

- 2223 • **name** : **NCName (0..1)** - specifies the name of the reference. If name is not specified the reference  
2224 name is taken from the name of the following data member.
- 2225 • **interfaceHeader** : **Name (1..1)** - specifies the C++ header defining the interface for the reference.
- 2226 • **interfaceClass** : **Name (0..1)** - specifies the C++ class defining the interface for the reference. If not  
2227 specified the class is derived from the type of the annotated data member.
- 2228 • **required** : **boolean (0..1)** - specifies whether a value has to be set for this reference. Default is *true*.

2229

2230 If the annotated data member is a `std::listvector` then the implied component type has a reference with a  
2231 multiplicity of either 0..n or 1..n depending on the value of the `@Reference required` attribute – 1..n  
2232 applies if `required=true`. Otherwise a multiplicity of 0..1 or 1..1 is implied.

2233

2234 **Applies to:** Data Member

2235

2236 **Example:**

2237 **Implementation:**

```
2238 // @Reference(interfaceHeader="LoanService.h" required="true")
2239 LoanService*ProxyPtr loanService;
2240
2241 // @Reference(interfaceHeader="LoanService.h" required="false")
2242 std::list<LoanService*vector<LoanServiceProxyPtr> loanServices;
```

2243

2244 **Component Type definition:**

```
2245 <componentType ... >
2246   <service ... />
2247   <reference name="loanService" multiplicity="1..1">
2248     <interface.cpp header="LoanService.h" class="LoanService" />
2249   </reference>
2250   <reference name="loanServices" multiplicity="0..n">
2251     <interface.cpp header="LoanService.h" class="LoanService" />
2252   </reference>
2253 </componentType>
```

2254 *Snippet A-26: Example of @Reference Annotation*

## 2255 **A.3A.4 Base Annotation Grammar**

```
2256 <annotation> ::= // @<baseAnnotation>
2257
2258 <baseAnnotation> ::= <name> [(<params>)]
2259
2260 <params> ::= <paramNameValue>[, <paramNameValue>]* |
2261           <paramValue>[, <paramValue>]*
2262
2263 <paramNameValue> ::= <name>="<value>"
2264
2265 <paramValue> ::= "<value>"
2266
2267 <name> ::= NCName
2268
2269 <value> ::= string
```

2270 *Snippet A-27: Base Annotation Grammar*

- 2271 • Adjacent string constants are concatenated
- 2272 • NCName is as defined by XML schema **[XSD]**
- 2273 • Whitespace including newlines between tokens is ignored.
- 2274 • Annotations with parameters can span multiple lines within a comment, and are considered complete
- 2275 when the terminating ")" is reached.

2276  
2277  
2278  
2279  
2280  
2281  
2282  
2283  
2284  
2285  
2286  
2287  
2288  
2289  
2290  
2291  
2292  
2293  
2294  
2295  
2296  
2297  
2298  
2299  
  
2300  
2301  
2302  
2303  
2304  
2305  
2306  
2307  
2308  
2309  
2310  
2311  
2312  
2313  
2314  
2315  
2316  
2317  
2318

## B C++ SCA Policy Annotations

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 **[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.

The SCA C++ policy annotations provide the capability for the developer to attach policy information to C++ implementation code. The annotations ~~concerned first~~ provide both general facilities for attaching SCA Intents and Policy Sets to C++ code. ~~Secondly, there are further specific and~~ annotations that deal with particular-specific policy intents. Policy annotation can be used in header files for certain policy domains such as Securityservice interfaces or implementations.

### B.1 General Intent Annotations

SCA provides the annotation **@Requires** for the attachment of any intent to a C++ class, to a C++ interface or to elements within classes and interfaces such as member functions and data members.

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 is ~~as follows~~:

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

*Snippet B-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 constants will be defined for the namespace part of this string, as well as for each intent that is used by C++ code. SCA defines constants for intents such as the following:

```

2319 // @Define SCA_PREFIX "{http://docs.oasis-
2320 open.org/ns/opencsa/sca/200903200912}"
2321 // @Define CONFIDENTIALITY SCA_PREFIX ## "confidentiality"
2322 // @Define CONFIDENTIALITY_MESSAGE CONFIDENTIALITY ## ".message"

```

2323 [Snippet B-2: Example Intent Constants](#)

2324

2325 Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,  
 2326 separated by an underscore. These intent constants are defined in the file that defines an annotation for  
 2327 the intent (annotations for intents, and the formal definition of these constants, are covered in a following  
 2328 section).

2329

2330 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.

2331

2332 **Corresponds to:** `@requires` attribute of [a-service, reference, operationan interface.cpp,](#)  
 2333 [inplemenation.cpp, function](#) or [propertycallbackFunction](#) element.

2334

2335 **Format:**

```

2336 // @Requires("qualifiedIntent" | {"qualifiedIntent" [, "qualifiedIntent"]})

```

2337 where

```

2338 qualifiedIntent ::= QName | QName.qualifier | QName.qualifier1.qualifier2

```

2339

2340 [Snippet B-3: @Requires Annotation Format](#)

2341 **Applies to:** Class, Member Function

2342

2343 Examples:

2344 Attaching the intents "confidentiality.message" and "integrity.message".

```

2345 // @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})

```

2346 [Snippet B-4: Example @Requires Annotation](#)

2347

2348 A reference requiring support for confidentiality:

```

2349 class Foo {
2350     ...
2351     // @Requires(CONFIDENTIALITY)
2352     // @Reference(interfaceHeader="SetBar.h")
2353     void setBar(Bar* bar);
2354     ...
2355 }

```

2356 [Snippet B-5: @Requires Annotation applied with an @Reference Annotation](#)

2357

2358 Users can also choose to only use constants for the namespace part of the QName, so that they can add  
 2359 new intents without having to define new constants. In that case, this definition would instead look like  
 2360 this:

2361

```

2362 class Foo {
2363     ...
2364     // @Requires(SCA_PREFIX "confidentiality ")
2365     // @Reference(interfaceHeader="SetBar.h")

```

```
2366     void setBar(Bar* bar);
2367     ...
2368 }
```

2369 [Snippet B-6: @Requires Annotation Using Mixed Constants and Literals](#)

## 2370 B.2 Specific Intent Annotations

2371 In addition to the general intent annotation supplied by the @Requires annotation described above, there  
2372 are C++ annotations that correspond to ~~some~~-specific policy intents.

2373

2374 The general form of these specific intent annotations is an annotation with a name derived from the name  
2375 of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation  
2376 in the form of a string or an array of strings.

2377

2378 For example, the SCA confidentiality intent described in General Intent Annotations using the  
2379 @Requires(CONFIDENTIALITY) intent can also be specified with the specific @Confidentiality intent  
2380 annotation. The specific intent annotation for the "integrity" security intent is:

```
2381 // @Integrity
2382
```

2383 [Snippet B-7: Example Specific Intent Annotation](#)

2384

2385 **Corresponds to:** @requires="*Intent*" attribute of ~~a service, reference, operationan interface.cpp,~~  
2386 ~~implementation, cpp, function~~ or ~~propertycallbackFunction~~ element.

2387

2388 **Format:**

```
2389 // @<Intent>[(qualifiers)]
```

2390 where Intent is an NCName that denotes a particular type of intent.

```
2391 Intent ::= NCName
2392 qualifiers ::= "qualifier " | {"qualifier" [, "qualifier" ] }
2393 qualifier ::= NCName | NCName/qualifier
```

2394

2395 [Snippet B-8: @<Intent> Annotation Format](#)

2396 **Applies to:** Class, Member Function – but see specific intents for restrictions

2397

2398 **Example:**

```
2399 // @ClientAuthentication( { "message", "transport" } )
```

2400 [Snippet B-9: Example @<Intent> Annotation](#)

2401

2402 This annotation attaches the pair of qualified intents: *authentication.message* and *authentication.transport*  
2403 (the sca: namespace is assumed in both of these cases – "http://docs.oasis-  
2404 open.org/opencsa/ns/sca/~~200903~~-200912").

2405

2406 The Policy Framework **[POLICY]** defines a number of intents and qualifiers. ~~The following~~  
2407 ~~sections~~ Security Interaction – Miscellaneous define the annotations for those intents.



2408 **B.2.1 Security Interaction**

Intent	Annotation
<u>authentication</u> clientAuthentication	@ClientAuthentication
serverAuthentication	@ServerAuthentication
mutualAuthentication	@MutualAuthentication
confidentiality	@Confidentiality
integrity	@Integrity

2409 *Table B-1: Security Interaction Intent Annotations*

- 2410
- 2411 These three intents can be qualified with
- 2412 • transport
  - 2413 • message

2414 **B.2.2 Security Implementation**

Intent	Annotation
runAs	@RunAs(role"role")
Allow	@Allow(roles="<comma-separated list of roles>")
permitAll	@PermitAll
denyAll	@DenyAll

2415

2416 In addition to allow roles to defined, an SCA runtime MAY use the following annotation

2417 @DeclareRoles(<comma-separated list of roles>")

Intent	Annotation	Qualifiers
authorization	@Authorization	fine_grain

2418 *Table B-2: Security Implementation Intent Annotations*

2419 **B.2.3 Reliable Messaging**

Intent	Annotation
atLeastOnce	@AtLeastOnce
atMostOnce	@AtMostOnce
<del>Ordered</del> <u>ordered</u>	@Ordered
exactlyOnce	@ExactlyOnce

2420

2421 *Table B-3: Reliable Messaging Intent Annotations*

2422 **B.2.4 Transactions**

Intent	Annotation	Qualifiers
managedTransaction	@ManagedTransaction	<del>Local</del> local global
<del>noManagedTransaction</del>	<del>@NoManagedTransaction</del>	
transactedOneWay	@TransactedOneWay	
immediateOneWay	@ImmediateOneWay	
propagates Transaction	@PropagatesTransaction	
suspendsTransaction	@SuspendsTransaction	

2423  
2424 *Table B-4: Transaction Intent Annotations*

2425 **B.2.5 Miscellaneous**

Intent	Annotation	Qualifiers
SOAP	@SOAP	<del>v1_1_4</del> v1_2
JMS	@JMS	

2426 **A.2 Application of Intent Annotations**

2427 ~~Where multiple intent annotations (general or specific) are applied to the same C++ element, they are~~  
2428 ~~additive in effect. An example of multiple policy annotations being used together follows:~~

```
2429  
2430 // @Authentication  
2431 // @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})
```

2432  
2433 ~~In this case, the effective intents are authentication, confidentiality.message and integrity.message.~~

2434  
2435 ~~If an annotation is specified at both the class/interface level and the member function or data member~~  
2436 ~~level, then the member function or data member level annotation completely overrides the class level~~  
2437 ~~annotation of the same type.~~

2438  
2439 ~~The intent annotation can be applied either to classes or to class member functions when adding~~  
2440 ~~annotated policy on SCA services.~~

2441 **A.3 Inheritance and Intent Annotations**

2442 ~~The following example shows the inheritance relations of intents on classes, operations, and super~~  
2443 ~~classes.~~

```
2444  
2445 // @Remotable  
2446 // @Integrity("transport")
```

```

2447 // @Authentication
2448 class HelloService{
2449 public:
2450 // @Integrity
2451 // @Authentication("message")
2452 ~wchar_t* hello(wchar_t* message){...}
2453
2454 // @Integrity
2455 // @Authentication("transport")
2456 ~wchar_t* helloThere(){...}
2457 }
2458
2459 // @Remotable
2460 // @Confidentiality("message")
2461 class HelloChildService : public HelloService{
2462 public:
2463 // @Confidentiality("transport")
2464 ~wchar_t* hello(wchar_t* message){...}
2465 // @Authentication
2466 ~wchar_t* helloWorld(){...}
2467 }

```

2468 Example 1a. Usage example of annotated policy and inheritance.

- 2469
- 2470 • The effective intent annotation on the helloWorld member function is @Integrity("transport"),
  - 2471 @Authentication, and @Confidentiality("message").
  - 2472 • The effective intent annotation on the hello member function of the HelloChildService is
  - 2473 @Integrity("transport"), @Authentication, and @Confidentiality("transport").
  - 2474 • The effective intent annotation on the helloThere member function of the HelloChildService is
  - 2475 @Integrity and @Authentication("transport"), the same as in HelloService class.
  - 2476 • The effective intent annotation on the hello member function of the HelloService is @Integrity and
  - 2477 @Authentication("message")

2478

2479 The listing below contains the equivalent declarative security interaction policy of the HelloService and

2480 HelloChildService implementation corresponding to the C++ classes shown in Example 1a.

```

2481
2482 <?xml version="1.0" encoding="ASCII"?>
2483
2484 <composite xmlns="http://docs.oasis-open.org/ns/opensa/sea/200903"
2485 ~~~~~
2486 ~~~~~
2487 <component name="HelloServiceComponent">
2488 <service name="HelloService" requires="integrity/transport
2489 ~~~~~
2490 ~~~~~
2491 ~~~~~
2492 </service>
2493 <implementation.epp library="HelloService.dll"
2494 ~~~~~
2495 ~~~~~
2496 <function name="hello" requires="integrity
2497 ~~~~~
2498 ~~~~~
2499 </implementation.epp>
2500 </component>
2501 <component name="HelloChildServiceComponent">
2502 <service name="HelloChildService" requires="integrity/transport
2503 ~~~~~
2504 ~~~~~
2505 ~~~~~

```

```

2505 </service>
2506 <implementation.cpp library="HelloChildService.dll"
2507     class="HelloChildServiceImpl">
2508     <function name="hello" requires="confidentiality/transport"/>
2509     <function name="helloThere" requires="integrity/transport
2510         authentication"/>
2511     <function name="helloWorld" requires="authentication"/>
2512 </implementation.cpp>
2513 </component>
2514
2515 ...
2516
2517 </composite>

```

2518 **Example 1b.** Declarative intents equivalent to annotated intents in Example 1a:

## 2519 **A.4 Relationship of Declarative and Annotated Intents**

2520 ~~Annotated intents on a C++ class cannot be overridden by declarative intents either in a composite~~  
2521 ~~document which uses the class as an implementation or by statements in a componentType document~~  
2522 ~~associated with the class. This rule follows the general rule for intents that they represent fundamental~~  
2523 ~~requirements of an implementation.~~

2524  
2525 ~~An unqualified version of an intent expressed through an annotation in the C function or function~~  
2526 ~~declaration can be qualified by a declarative intent in a using composite document.~~

2527 [Table B-5: Miscellaneous Intent Annotations](#)

## 2528 **B.3 Policy Set Annotations**

2529 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies (for example,  
2530 a concrete policy is the specific encryption algorithm to use when encrypting messages when using a  
2531 specific communication protocol to link a reference to a service).

2532  
2533 Policy Sets can be applied directly to C++ implementations using the **@PolicySets** annotation. The  
2534 PolicySets annotation either takes the QName of a single policy set as a string or the name of two or  
2535 more policy sets as an array of strings.

2536  
2537 **Corresponds to:** ~~@policySets~~ attribute of ~~a service, reference, operation or interface.cpp,~~  
2538 ~~implementation.cpp, function~~ or ~~propertycallbackFunction~~ element.

2539  
2540 **Format:**

```

2541 // @PolicySets("<policy set QName>" |
2542 //           { "<policy set QName>" [, "<policy set QName>"] })

```

2543 [Snippet B-10: @PolicySets Annotation Format](#)

2544 As for intents, PolicySet names are QNames – in the form of “{Namespace-URI}localPart”.

2545  
2546 **Applies to:** Class, Member Function,

2547  
2548 **Example:**

```

2549 // @Reference(name="helloService", interfaceHeader="helloService.h",
2550 //           required="true")
2551 // @PolicySets({ MY_NS "WS_Encryption_Policy",
2552 //             MY_NS "WS_Authentication_Policy"})
2553 HelloService* helloService;

```

2554 ...

2555 [Snippet B-11: Example @PolicySets Annotation](#)

2556

2557 In this case, the Policy Sets WS\_Encryption\_Policy and WS\_Authentication\_Policy are applied, both  
2558 using the namespace defined for the constant MY\_NS.

2559

2560 PolicySets satisfy intents expressed for the implementation when both are present, according to the rules  
2561 defined in **[POLICY]**.

## 2562 B.4 Policy Annotation Grammar Additions

```
2563 <annotation> ::= // @<baseAnnotation> | @<requiresAnnotation> |  
2564                 @<intentAnnotation> | @<policySetAnnotation>  
2565  
2566 <requiresAnnotation> ::= Requires(<intents>)  
2567  
2568 <intents> ::= "<qualifiedIntent>" |  
2569             {"<qualifiedIntent>"[, "<qualifiedIntent>"]*})  
2570  
2571 <qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |  
2572                   <intentName>.<qualifier>.<qualifier>  
2573  
2574 <intentName> ::= {aAnyURI}NCName  
2575  
2576 <intentAnnotation> ::= <intent>[(<qualifiers>)]  
2577  
2578 <intent> ::= NCName [(param)]  
2579  
2580 <qualifiers> ::= "<qualifier>" | {"<qualifier>"[, "<qualifier>"]*}  
2581  
2582 <qualifier> ::= NCName | NCName/<qualifier>  
2583  
2584 <policySetAnnotation> ::= policySets(<policysets>)  
2585  
2586 <policysets> ::= "<policySetName>" | {"<policySetName>"[, "<policySetName>"]*}  
2587  
2588 <policySetName> ::= {aAnyURI}NCName
```

2589 [Snippet B-12: Annotation Grammar Additions for Policy Annotations](#)

2590 • anyURI is as defined by XML schema **[XSD]**

## 2591 B.5 Annotation Constants

```
2592 <annotationConstant> ::= // @Define <identifier> <token string>  
2593  
2594 <identifier> ::= token  
2595  
2596 <token string> ::= "string" | "string"[ ## <token string>]
```

2597 [Snippet B-13: Annotation Constants Grammar](#)

2598 • Constants are immediately expanded

2599

## C C++ WSDL Mapping Annotations

2600 To allow developers to control the mapping of C++ to WSDL, a set of annotations is defined. If WSDL  
2601 mapping annotations are supported by an implementation, the annotations defined here MUST be  
2602 supported and MUST be mapped to WSDL as described. [CPPC0001]

### 2603 C.1 Interface Header Annotations

#### 2604 C.1.1 @WebService

2605 Annotation on a C++ class indicating that it represents a web service. An SCA implementation MUST  
2606 treat any instance of a @Remotable annotation and without an explicit @WebService annotation as if a  
2607 @WebService annotation with no parameters was specified. An SCA implementation MUST treat any  
2608 instance of a @Interface annotation and without an explicit @WebService annotation as if a  
2609 @WebService annotation with no parameters was specified. An SCA implementation MUST treat any  
2610 instance of a @Interface annotation and without an explicit @WebService annotation as if a  
2611 @WebService annotation with no parameters was specified. [CPPC0002]

2612

2613 [CPPC0002]

2614 **Corresponds to:** javax.jws.WebService annotation in the JAX-WS specification (7.11.1)

2615

2616 **Format:**

```
2617 // @WebService(name="portTypeName", targetNamespace="namespaceURI",  
2618 //           serviceName="WSDLServiceName", portName="WSDLPortName")
```

2619 *Snippet C-1: @WebService Annotation Format*

2620 where:

- 2621 • **name : NCName (0..1)** – specifies the name of the web service portType. The default is the name of  
2622 the C++ class the annotation is applied to. The name of the associated binding is also determined by  
2623 the portType. The binding name is the name of the portType suffixed with "Binding".
- 2624 • **targetNamespace : anyURI (0..1)** – specifies the target namespace for the web service. The default  
2625 namespace is determined by the implementation.
- 2626 • **serviceName : NCName (0..1)** – specifies the target name for the associated service. The default  
2627 service name is the name of the C++ class suffixed with "Service". ~~The~~
- 2628 • **portName : NCName (0..1)** – specifies the name to be used for the associated WSDL port for the  
2629 service. If portName is not specified, the name of the associated binding is also determined by the  
2630 serviceName. In the case of a SOAP binding, the binding name WSDL port is the name of the service  
2631 suffixed portType suffixed with "SoapBinding".Port". See [CPPF0042]
- 2632 • ~~**portName : NCName (0..1)** – specifies the name to be used for the associated WSDL port for the~~  
2633 ~~service. If a @WebService does not have a portName element, an SCA implementation MUST use~~  
2634 ~~the value associated with the name element, suffixed with "Port". [CPPC0003]~~

2635

2636 **Applies to:** Class

2637

2638 **Example:**

2639 Input C++ source file:

```
2640 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
2641 //           serviceName="StockQuoteService")
```

```
2642 class StockQuoteService {
2643     };
```

2644

2645 Generated WSDL file:

```
2646 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2647     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2648     xmlns:tns="http://www.example.org/"
2649     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2650     targetNamespace="http://www.example.org/">
2651
2652     <portType name="StockQuote">
2653         <cpp:bindings>
2654             <cpp:class name="StockQuoteService"/>
2655         </cpp:bindings>
2656     </portType>
2657
2658     <binding name="StockQuoteServiceSoapBinding">
2659         <soap:binding style="document"
2660             transport="http://schemas.xmlsoap.org/soap/http"/>
2661     </binding>
2662
2663     <service name="StockQuoteService">
2664         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2665             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2666         </port>
2667     </service>
2668 </definitions>
```

2669 [Snippet C-2: Example @WebService Annotation](#)

## 2670 C.1.2 @WebFunction

2671 Annotation on a C++ member function indicating that it represents a web service operation. **An SCA**  
2672 **implementation MUST treat a member function annotated with an @Function annotation and without an**  
2673 **explicit @WebFunction annotation as if a @WebFunction annotation with with an operationName value**  
2674 **equal to the name value of the @Function annotation, an exclude value equal to the exclude value of the**  
2675 **@Operation annotation and no other parameters was specified. [CPPC0009]**

2676

2677 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2678

2679 **Format:**

```
2680 // @WebFunction(operationName="operation", action="SOAPAction",
2681 //     exclude="false")
```

2682 [Snippet C-3: @WebFunction Annotation Format](#)

2683 where:

- 2684 • **operationName : NCName (0..1)** – specifies the name of the WSDL operation to associate with this  
2685 function. The default is the name of the C++ member function the annotation is applied to.
- 2686 • **action : string (0..1)** – specifies the value associated with the soap:operation/@soapAction attribute  
2687 in the resulting code. The default value is an empty string.
- 2688 • **exclude : boolean (0..1)** – specifies whether this member function is included in the web service  
2689 interface. The default value is “false”.

2690

2691 **Applies to:** Member function.

2692

2693 Example:

2694 Input C++ source file:

```
2695 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
2696 //           serviceName="StockQuoteService")
2697 class StockQuoteService {
2698
2699     // @WebFunction(operationName="GetLastTradePrice",
2700     //           action="urn:GetLastTradePrice")
2701     float getLastTradePrice(const std::string& tickerSymbol);
2702
2703     // @WebFunction(exclude=true)
2704     void setLastTradePrice(const std::string& tickerSymbol, float value);
2705 };
```

2706

2707 Generated WSDL file:

```
2708 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2709             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2710             xmlns:tns="http://www.example.org/"
2711             xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2712             targetNamespace="http://www.example.org/">
2713
2714     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2715             xmlns:tns="http://www.example.org/"
2716             attributeFormDefault="unqualified"
2717             elementFormDefault="unqualified"
2718             targetNamespace="http://www.example.org/">
2719         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2720         <xs:element name="GetLastTradePriceResponse"
2721             type="tns:GetLastTradePriceResponse"/>
2722         <xs:complexType name="GetLastTradePrice">
2723             <xs:sequence>
2724                 <xs:element name="tickerSymbol" type="xs:string"/>
2725             </xs:sequence>
2726         </xs:complexType>
2727         <xs:complexType name="GetLastTradePriceResponse">
2728             <xs:sequence>
2729                 <xs:element name="return" type="xs:float"/>
2730             </xs:sequence>
2731         </xs:complexType>
2732     </xs:schema>
2733
2734     <-message name="GetLastTradePrice">
2735         <part name="parameters" element="tns:GetLastTradePrice">
2736             </part>
2737     </message>
2738
2739     <-message name="GetLastTradePriceResponse">
2740         <part name="parameters" element="tns:GetLastTradePriceResponse">
2741             </part>
2742     </-message>
2743
2744     <portType name="StockQuote">
2745         <cpp:bindings>
2746             <cpp:class name="StockQuoteService"/>
2747         </cpp:bindings>
2748         <operation name="GetLastTradePrice">
2749             <cpp:bindings>
2750                 <cpp:memberFunction name="getLastTradePrice"/>
2751             </cpp:bindings>
2752             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2753                 </input>
2754             <output name="GetLastTradePriceResponse"
```



```

2755         message="tns:GetLastTradePriceResponse">
2756     </output>
2757 </operation>
2758 </portType>
2759
2760 <binding name="StockQuoteServiceSoapBinding">
2761     <soap:binding style="document"
2762         transport="http://schemas.xmlsoap.org/soap/http"/>
2763     <wSDL:operation name="GetLastTradePrice">
2764         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2765         <wSDL:input name="GetLastTradePrice">
2766             <soap:body use="literal"/>
2767         </wSDL:input>
2768         <wSDL:output name="GetLastTradePriceResponse">
2769             <soap:body use="literal"/>
2770         </wSDL:output>
2771     </wSDL:operation>
2772 </binding>
2773
2774 <service name="StockQuoteService">
2775     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2776         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2777     </port>
2778 </service>
2779 </definitions>

```

2780 [Snippet C-4: Example @WebFunction Annotation](#)

### 2781 C.1.3 @OneWay

2782 Annotation on a C++ member function indicating that it represents a one-way request. The @OneWay  
2783 annotation also affects the service interface, see @OneWay.

2784

2785 **Corresponds to:** javax.jws.OneWay annotation in the JAX-WS specification (7.11.3)

2786

2787 **Format:**

```
2788 // @OneWay
```

2789

2790 [Snippet C-5: @OneWay Annotation Format](#)

2791 **Applies to:** Member function.

2792

2793 **Example:**

2794 **Input C++ source file:**

```

2795 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
2796 //     serviceName="StockQuoteService")
2797 class StockQuoteService {
2798
2799     // @WebFunction(operationName="SetTradePrice",
2800     //     action="urn:SetTradePrice")
2801     // @OneWay
2802     void setTradePrice(const std::string& tickerSymbol, float price);
2803 };

```

2804

2805 **Generated WSDL file:**

```

2806 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2807     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

```

```

2808     xmlns:tns="http://www.example.org/"
2809     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2810     targetNamespace="http://www.example.org/"
2811
2812     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2813               xmlns:tns="http://www.example.org/"
2814               attributeFormDefault="unqualified"
2815               elementFormDefault="unqualified"
2816               targetNamespace="http://www.example.org/">
2817       <xs:element name="SetTradePrice" type="tns:SetTradePrice"/>
2818       <xs:complexType name="SetTradePrice">
2819         <xs:sequence>
2820           <xs:element name="tickerSymbol" type="xs:string"/>
2821           <xs:element name="price" type="xs:float"/>
2822         </xs:sequence>
2823       </xs:complexType>
2824     </xs:schema>
2825
2826     <-message name="SetTradePrice">
2827       <part name="parameters" element="tns:SetTradePrice">
2828         </part>
2829     </message>
2830
2831     <portType name="StockQuote">
2832       <cpp:bindings>
2833         <cpp:class name="StockQuoteService"/>
2834       </cpp:bindings>
2835       <operation name="SetTradePrice">
2836         <cpp:bindings>
2837           <cpp:memberFunction name="setTradePrice"/>
2838         </cpp:bindings>
2839         <input name="SetTradePrice" message="tns:SetTradePrice">
2840           </input>
2841         </operation>
2842     </portType>
2843
2844     <binding name="StockQuoteServiceSoapBinding">
2845       <soap:binding style="document"
2846                   transport="http://schemas.xmlsoap.org/soap/http"/>
2847       <wSDL:operation name="SetTradePrice">
2848         <soap:operation soapAction="urn:SetTradePrice" style="document"/>
2849         <wSDL:input name="SetTradePrice">
2850           <soap:body use="literal"/>
2851         </wSDL:input>
2852       </wSDL:operation>
2853     </binding>
2854
2855     <service name="StockQuoteService">
2856       <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2857         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2858       </port>
2859     </service>
2860 </definitions>

```

2861 [\*Snippet C-6: Example @OneWay Annotation\*](#)

## 2862 **C.1.4 @WebParam**

2863 Annotation on a C++ member function indicating the mapping of a parameter to the associated input and  
2864 output WSDL messages.

2865

2866 **Corresponds to:** javax.jws.WebParam annotation in the JAX-WS specification (7.11.4)

2867

2868 **Format:**

```
2869 // @WebParam(paramName=<="parameter", name="WSDLElement",
2870 //           targetNamespace="namespaceURI", mode="IN"|"OUT"|"INOUT",
2871 //           header="false", partName="WSDLPart", type="xsdType")
```

2872 *Snippet C-7: @WebParam Annotation Format*

2873 where:

- 2874 • **paramName : NCName (1..1)** – specifies the name of the parameter that this annotation applies to.  
2875 **The value of the paramName of a @WebParam annotation MUST be the name of a parameter of the**  
2876 **member function the annotation is applied to. [CPPC0004]**
- 2877 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default  
2878 value is the name of the parameter. If an @WebParam annotation is not present, and the parameter  
2879 is unnamed, then a name of “argN”, where N is an incrementing value from 1 indicating the position of  
2880 he parameter in the argument list, will be used.
- 2881 • **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default  
2882 namespace is the namespace of the associated @WebService. The targetNamespace attribute is  
2883 ignored unless the binding style is document, and the binding parameterStyle is bare. See  
2884 @SOAPBinding.
- 2885 • **mode : token (0..1)** – specifies whether the parameter is associated with the input message, output  
2886 message, or both. The default value is determined by the passing mechanism for the parameter, see  
2887 Parameter and return type classification.
- 2888 • **header : boolean (0..1)** – specifies whether this parameter is associated with a SOAP header  
2889 element. The default value is “false”.
- 2890 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The  
2891 default value is the value of name.
- 2892 • **type : NCName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with  
2893 this parameter. **The value of the type property of a @WebParam annotation MUST be one of the**  
2894 **simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema. [CPPC0005]** The default  
2895 type is determined by the mapping defined in Simple Content Binding.

2896

2897 **Applies to:** Member function parameter.

2898

2899 **Example:**

2900 Input C++ source file:

```
2901 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
2902 //           serviceName="StockQuoteService")
2903 class StockQuoteService {
2904
2905     // @WebFunction(operationName="GetLastTradePrice",
2906     //           action="urn:GetLastTradePrice")
2907     // @WebParam(paramName="tickerSymbol", name="symbol")
2908     float getLastTradePrice(const std::string& tickerSymbol);
2909 };
```

2910

2911 **Generated WSDL file:**

```
2912 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2913             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2914             xmlns:tns="http://www.example.org/"
2915             xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2916             targetNamespace="http://www.example.org/">
2917
2918     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```

2919     xmlns:tns="http://www.example.org/"
2920     attributeFormDefault="unqualified"
2921     elementFormDefault="unqualified"
2922     targetNamespace="http://www.example.org/">
2923 <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice" />
2924 <xs:element name="GetLastTradePriceResponse"
2925     type="tns:GetLastTradePriceResponse" />
2926 <xs:complexType name="GetLastTradePrice">
2927     <xs:sequence>
2928         <xs:element name="symbol" type="xs:string" />
2929     </xs:sequence>
2930 </xs:complexType>
2931 <xs:complexType name="GetLastTradePriceResponse">
2932     <xs:sequence>
2933         <xs:element name="return" type="xs:float" />
2934     </xs:sequence>
2935 </xs:complexType>
2936 </xs:schema>
2937
2938 <-message name="GetLastTradePrice">
2939     <part name="parameters" element="tns:GetLastTradePrice">
2940     </part>
2941 </message>
2942
2943 <-message name="GetLastTradePriceResponse">
2944     <part name="parameters" element="tns:GetLastTradePriceResponse">
2945     </part>
2946 </-message>
2947
2948 <portType name="StockQuote">
2949     <cpp:bindings>
2950         <cpp:class name="StockQuoteService" />
2951     </cpp:bindings>
2952     <operation name="GetLastTradePrice">
2953         <cpp:bindings>
2954             <cpp:memberFunction name="getLastTradePrice" />
2955             <cpp:parameter name="tickerSymbol"
2956                 part="tns:GetLastTradePrice/parameter"
2957                 childElementName="symbol" />
2958         </cpp:bindings>
2959         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2960         </input>
2961         <output name="GetLastTradePriceResponse"
2962             message="tns:GetLastTradePriceResponse">
2963         </output>
2964     </operation>
2965 </portType>
2966
2967 <binding name="StockQuoteServiceSoapBinding">
2968     <soap:binding style="document"
2969         transport="http://schemas.xmlsoap.org/soap/http" />
2970     <wSDL:operation name="GetLastTradePrice">
2971         <soap:operation soapAction="urn:GetLastTradePrice" style="document" />
2972         <wSDL:input name="GetLastTradePrice">
2973             <soap:body use="literal" />
2974         </wSDL:input>
2975         <wSDL:output name="GetLastTradePriceResponse">
2976             <soap:body use="literal" />
2977         </wSDL:output>
2978     </wSDL:operation>
2979 </binding>
2980
2981 <service name="StockQuoteService">
2982     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">

```

```
2983     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2984     </port>
2985   </service>
2986 </definitions>
```

2987 [Snippet C-8: Example @WebParam Annotation](#)

## 2988 C.1.5 @WebResult

2989 Annotation on a C++ member function indicating the mapping of the member function's return type to the  
2990 associated output WSDL message.

2991

2992 **Corresponds to:** javax.jws.WebResult annotation in the JAX-WS specification (7.11.5)

2993

2994 **Format:**

```
2995 // @WebResult(name=<="WSDLElement", targetNamespace="namespaceURI",
2996 //           header="false", partName="WSDLPart", type="xsdType")
```

2997 [Snippet C-9: @WebResult Annotation Format](#)

2998 where:

- 2999 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default  
3000 value is “return”.
- 3001 • **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default  
3002 namespace is the namespace of the associated @WebService. The targetNamespace attribute is  
3003 ignored unless the binding style is document, and the binding parameterStyle is bare. See  
3004 @SOAPBinding.
- 3005 • **header : boolean (0..1)** – specifies whether the result is associated with a SOAP header element.  
3006 The default value is “false”.
- 3007 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The  
3008 default value is the value of name.
- 3009 • **type : NCName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with  
3010 this parameter. The value of the type property of a @WebResult annotation MUST be one of the  
3011 simpleTypes defined in namespace <http://www.w3.org/2001/XMLSchema>. [CPPC0006] The default  
3012 type is determined by the mapping defined in Simple Content Binding.

3013

3014 **Applies to:** Member function return value.

3015

3016 **Example:**

3017 Input C++ source file:

```
3018 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3019 //           serviceName="StockQuoteService")
3020 class StockQuoteService {
3021
3022     // @WebFunction(operationName="GetLastTradePrice",
3023     //           action="urn:GetLastTradePrice")
3024     // @WebResult(name="price")
3025     float getLastTradePrice(const std::string& tickerSymbol);
3026 };
```

3027

3028 Generated WSDL file:

```
3029 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3030             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```

3031     xmlns:tns="http://www.example.org/"
3032     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3033     targetNamespace="http://www.example.org/"
3034
3035     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3036         xmlns:tns="http://www.example.org/"
3037         attributeFormDefault="unqualified"
3038         elementFormDefault="unqualified"
3039         targetNamespace="http://www.example.org/">
3040     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3041     <xs:element name="GetLastTradePriceResponse"
3042         type="tns:GetLastTradePriceResponse"/>
3043     <xs:complexType name="GetLastTradePrice">
3044         <xs:sequence>
3045             <xs:element name="tickerSymbol" type="xs:string"/>
3046         </xs:sequence>
3047     </xs:complexType>
3048     <xs:complexType name="GetLastTradePriceResponse">
3049         <xs:sequence>
3050             <xs:element name="price" type="xs:float"/>
3051         </xs:sequence>
3052     </xs:complexType>
3053 </xs:schema>
3054
3055 <-message name="GetLastTradePrice">
3056     <part name="parameters" element="tns:GetLastTradePrice">
3057         </part>
3058 </message>
3059
3060 <-message name="GetLastTradePriceResponse">
3061     <part name="parameters" element="tns:GetLastTradePriceResponse">
3062         </part>
3063 </-message>
3064
3065 <portType name="StockQuote">
3066     <cpp:bindings>
3067         <cpp:class name="StockQuoteService"/>
3068     </cpp:bindings>
3069     <operation name="GetLastTradePrice">
3070         <cpp:bindings>
3071             <cpp:memberFunction name="getLastTradePrice"/>
3072         </cpp:bindings>
3073         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3074             </input>
3075         <output name="GetLastTradePriceResponse"
3076             message="tns:GetLastTradePriceResponse">
3077             </output>
3078         </operation>
3079 </portType>
3080
3081 <binding name="StockQuoteServiceSoapBinding">
3082     <soap:binding style="document"
3083         transport="http://schemas.xmlsoap.org/soap/http"/>
3084     <wSDL:operation name="GetLastTradePrice">
3085         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3086         <wSDL:input name="GetLastTradePrice">
3087             <soap:body use="literal"/>
3088         </wSDL:input>
3089         <wSDL:output name="GetLastTradePriceResponse">
3090             <soap:body use="literal"/>
3091         </wSDL:output>
3092     </wSDL:operation>
3093 </binding>
3094

```

```
3095     <service name="StockQuoteService">
3096         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3097             <soap:address location="REPLACE_WITH_ACTUAL_URL" />
3098         </port>
3099     </service>
3100 </definitions>
```

3101 [Snippet C-10: Example @WebResult Annotation](#)

## 3102 C.1.6 @SOAPBinding

3103 Annotation on a C++ member function indicating that it represents a web service operation.

3104

3105 **Corresponds to:** javax.jws.SOAPBinding annotation in the JAX-WS specification (7.11.6)

3106

3107 **Format:**

```
3108 // @SOAPBinding(style="DOCUMENT" | "RPC", use="LITERAL" | "ENCODED",
3109 //     parameterStyle="BARE" | "WRAPPED")
```

3110 [Snippet C-11: @SOAPBinding Annotation Format](#)

3111 where:

- 3112 • **style : token (0..1)** – specifies the WSDL binding style. The default value is “DOCUMENT”.
- 3113 • **use : token (0..1)** – specifies the WSDL binding use. The default value is “LITERAL”.
- 3114 • **parameterStyle : token (0..1)** – specifies the WSDL parameter style. The default value is  
3115 “WRAPPED”.

3116

3117 **Applies to:** Class, Member function.

3118

3119 **Example:**

3120 Input C++ source file:

```
3121 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3122 //     serviceName="StockQuoteService")
3123 // @SOAPBinding(style="RPC")
3124 class StockQuoteService {
3125 };
```

3126

3127 **Generated WSDL file:**

```
3128 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3129     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3130     xmlns:tns="http://www.example.org/"
3131     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3132     targetNamespace="http://www.example.org/">
3133
3134     <portType name="StockQuote">
3135         <cpp:bindings>
3136             <cpp:class name="StockQuoteService" />
3137         </cpp:bindings>
3138     </portType>
3139
3140     <binding name="StockQuoteServiceSoapBinding">
3141         <soap:binding style="document"
3142             transport="http://schemas.xmlsoap.org/soap/http" />
3143     </binding>
3144
3145     <service name="StockQuoteService">
```



```
3146     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3147         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3148     </port>
3149 </service>
3150 </definitions>
```

3151 [Snippet C-12: Example @SOAPBinding Annotation](#)

## 3152 C.1.7 @WebFault

3153 Annotation on a C++ exception class indicating that it might be thrown as a fault by a web service  
3154 function. A C++ class with a @WebFault annotation MUST provide a constructor that takes two  
3155 parameters, a std::string and a type representing the fault information. Additionally, the class MUST  
3156 provide a const member function "getFaultInfo" that takes no parameters, and returns the same type as  
3157 defined in the constructor. [CPPC0007]

3158

3159 **Corresponds to:** javax.xml.ws.WebFault annotation in the JAX-WS specification (7.2)

3160

3161 **Format:**

```
3162 // @WebFault(name="WSDL_Element", targetNamespace="namespaceURI")
```

3163 [Snippet C-13: @WebFault Annotation Format](#)

3164 where:

- 3165 • **name : NCName (1..1)** – specifies local name of the global element mapped to this fault.
- 3166 • **targetNamespace : string (0..1)** – specifies the namespace of the global element mapped to this  
3167 fault. The default namespace is determined by the implementation.

3168

3169 **Applies to:** Class.

3170

3171 **Example:**

3172 Input C++ source file:

```
3173 // @WebFault(name="UnknownSymbolFault",
3174 //     targetNamespace="http://www.example.org/")
3175 class UnknownSymbol {
3176     UnknownSymbol(const char* message,
3177         const std::string& faultInfo);
3178
3179     std::string getFaultInfo() const;
3180 };
3181
3182 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3183 //     serviceName="StockQuoteService")
3184 class StockQuoteService {
3185
3186     // @WebFunction(operationName="GetLastTradePrice",
3187     //     action="urn:GetLastTradePrice")
3188     // @WebThrows(faults="UnknownSymbol")
3189     float getLastTradePrice(const std::string& tickerSymbol);
3190 };
```

3191

3192 **Generated WSDL file:**

```
3193 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3194     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3195     xmlns:tns="http://www.example.org/"
3196     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
```



```

3197     targetNamespace="http://www.example.org/">
3198
3199     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3200       xmlns:tns="http://www.example.org/"
3201       attributeFormDefault="unqualified"
3202       elementFormDefault="unqualified"
3203       targetNamespace="http://www.example.org/">
3204     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3205     <xs:element name="GetLastTradePriceResponse"
3206       type="tns:GetLastTradePriceResponse"/>
3207     <xs:complexType name="GetLastTradePrice">
3208       <xs:sequence>
3209         <xs:element name="tickerSymbol" type="xs:string"/>
3210       </xs:sequence>
3211     </xs:complexType>
3212     <xs:complexType name="GetLastTradePriceResponse">
3213       <xs:sequence>
3214         <xs:element name="return" type="xs:float"/>
3215       </xs:sequence>
3216     </xs:complexType>
3217     <xs:element name="UnknownSymbolFault" type="xs:string"/>
3218   </xs:schema>
3219
3220   <message name="GetLastTradePrice">
3221     <part name="parameters" element="tns:GetLastTradePrice">
3222     </part>
3223   </message>
3224
3225   <message name="GetLastTradePriceResponse">
3226     <part name="parameters" element="tns:GetLastTradePriceResponse">
3227     </part>
3228   </message>
3229
3230   <message name="UnknownSymbol">
3231     <part name="parameters" element="tns:UnknownSymbolFault">
3232     </part>
3233   </message>
3234
3235   <portType name="StockQuote">
3236     <cpp:bindings>
3237       <cpp:class name="StockQuoteService"/>
3238     </cpp:bindings>
3239     <operation name="GetLastTradePrice">
3240       <cpp:bindings>
3241         <cpp:memberFunction name="getLastTradePrice"/>
3242       </cpp:bindings>
3243       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3244       </input>
3245       <output name="GetLastTradePriceResponse"
3246         message="tns:GetLastTradePriceResponse">
3247       </output>
3248       <fault name="UnknownSymbol" message="tns:UnknownSymbol">
3249       </fault>
3250     </operation>
3251   </portType>
3252
3253   <binding name="StockQuoteServiceSoapBinding">
3254     <soap:binding style="document"
3255       transport="http://schemas.xmlsoap.org/soap/http"/>
3256     <wSDL:operation name="GetLastTradePrice">
3257       <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3258     <wSDL:input name="GetLastTradePrice">
3259       <soap:body use="literal"/>
3260     </wSDL:input>

```

```

3261 |     <wddl:output name="GetLastTradePriceResponse">
3262 |         <soap:body use="literal"/>
3263 |     </wddl:output>
3264 |     <wddl:fault>
3265 |         <soap:fault name="UnknownSymbol" use="literal"/>
3266 |     </wddl:fault>
3267 | </wddl:operation>
3268 | </binding>
3269 |
3270 |     <service name="StockQuoteService">
3271 |         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3272 |             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3273 |         </port>
3274 |     </service>
3275 | </definitions>

```

3276 | [Snippet C-14: Example @WebFault Annotation](#)

## 3277 | C.1.8 @WebThrows

3278 | Annotation on a C++ class indicating which faults might be thrown by this class.

3279 |

3280 | **Corresponds to:** No equivalent in JAX-WS.

3281 |

3282 | **Format:**

```

3283 | // @WebThrows(faults="faultMsg1" [, "faultMsgn"]*)

```

3284 | [Snippet C-15: @WebThrows Annotation Format](#)

3285 | where:

- 3286 | • **faults : NMTOKEN (1..n)** – specifies the names of all faults that might be thrown by this member  
3287 | function. The name of the fault is the name of its associated C++ class name. A C++ class that is  
3288 | listed in a @WebThrows annotation MUST itself have a @WebFault annotation. [CPPC0008]

3289 |

3290 | **Applies to:** Member function.

3291 |

3292 | Example:

3293 | See [@WebFault](#)@WebFault.

3294

## D WSDL C++ Mapping Extensions

3295 | The following A set of WSDL extensions are used to augment the conversion process from WSDL to  
3296 C++. All of these extensions are defined in the namespace `http://docs.oasis-open.org/ns/opencsa/sca-c-`  
3297 `cpp/cpp/200901`. For brevity, all definitions of these extensions will be fully qualified, and all references to  
3298 the "cpp" prefix are associated with the namespace above. If WSDL extensions are supported by an  
3299 implementation, all the extensions defined here MUST be supported and MUST be mapped to C++ as  
3300 described. [CPPD0001]

### 3301 D.1 <cpp:bindings>

3302 <cpp:bindings> is a container type which can be used as a WSDL extension. All other SCA wsdl  
3303 extensions will be specified as children of a <cpp:bindings> element. A <cpp:bindings> element can be  
3304 used as an extension to any WSDL type that accepts extensions.

### 3305 D.2 <cpp:class>

3306 <cpp:class> provides a mechanism for defining an alternate C++ class name for a WSDL construct.

3307

3308 **Format:**

```
3309 <cpp:class name="xsd:string"/>
```

3310 *Snippet D-1: <cpp:class> Element Format*

3311 where:

- 3312 • **class/@name : NCName (1..1)** – specifies the name of the C++ class associated with this WSDL  
3313 element.

3314

3315 **Applicable WSDL element(s):**

- 3316 • wsd:portType
- 3317 • wsd:fault

3318

3319 A <cpp:bindings/> element MUST NOT have more than one <cpp:class/> child element. [CPPD0002]

3320

3321 **Example:**

3322 **Input WSDL file:**

```
3323 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3324     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3325     xmlns:tns="http://www.example.org/"  
3326     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"  
3327     targetNamespace="http://www.example.org/">  
3328  
3329     <portType name="StockQuote">  
3330         <cpp:bindings>  
3331             <cpp:class name="StockQuoteService"/>  
3332         </cpp:bindings>  
3333     </portType>  
3334 </definitions>
```

3335

3336 **Generated C++ file:**

```
3337 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3338 //     serviceName="StockQuoteService")
3339 class StockQuoteService {
3340 };
```

3341 [Snippet D-2: Example <cpp:class> Element](#)

## 3342 D.3 <cpp:enableWrapperStyle>

3343 <cpp:enableWrapperStyle> indicates whether or not the wrapper style for messages is applied, when  
3344 otherwise applicable. If false, the wrapper style will never be applied.

3345  
3346 **Format:**

```
3347 <cpp:enableWrapperStyle>value</cpp:enableWrapperStyle>
```

3348 [Snippet D-3: <cpp:enableWrapperStyle> Element Format](#)

3349 where:

- 3350 • **enableWrapperStyle/text() : boolean (1..1)** – specifies whether wrapper style is enabled or disabled  
3351 for this element and any of it's children. The default value is "true".

3352  
3353 **Applicable WSDL element(s):**

- 3354 • wsdl:definitions
- 3355 • wsdl:portType – overrides a binding applied to wsdl:definitions
- 3356 • wsdl:portType/wsdl:operation – overrides a binding applied to wsdl:definitions or the enclosing  
3357 wsdl:portType

3358  
3359 **A** <cpp:bindings/> element **MUST NOT** have more than one <cpp:enableWrapperStyle/> child element.  
3360 [CPPD0003]

3361  
3362 **Example:**

3363 Input WSDL file:

```
3364 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3365     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3366     xmlns:tns="http://www.example.org/"
3367     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3368     targetNamespace="http://www.example.org/">
3369
3370     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3371         xmlns:tns="http://www.example.org/"
3372         attributeFormDefault="unqualified"
3373         elementFormDefault="unqualified"
3374         targetNamespace="http://www.example.org/">
3375         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3376         <xs:element name="GetLastTradePriceResponse"
3377             type="tns:GetLastTradePriceResponse"/>
3378         <xs:complexType name="GetLastTradePrice">
3379             <xs:sequence>
3380                 <xs:element name="tickerSymbol" type="xs:string"/>
3381             </xs:sequence>
3382         </xs:complexType>
3383         <xs:complexType name="GetLastTradePriceResponse">
3384             <xs:sequence>
3385                 <xs:element name="return" type="xs:float"/>
3386             </xs:sequence>
3387         </xs:complexType>
```

```

3388 </xs:schema>
3389
3390 <-message name="GetLastTradePrice">
3391   <part name="parameters" element="tns:GetLastTradePrice">
3392     </part>
3393 </message>
3394
3395 <-message name="GetLastTradePriceResponse">
3396   <part name="parameters" element="tns:GetLastTradePriceResponse">
3397     </part>
3398 </-message>
3399
3400 <portType name="StockQuote">
3401   <cpp:bindings>
3402     <cpp:class name="StockQuoteService" />
3403     <cpp:enableWrapperStyle>false</cpp:enableWrapperStyle>
3404   <cpp:bindings>
3405     <operation name="GetLastTradePrice">
3406       <cpp:bindings>
3407         <cpp:memberFunction name="getLastTradePrice" />
3408       </cpp:bindings>
3409       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3410         </input>
3411       <output name="GetLastTradePriceResponse"
3412         message="tns:GetLastTradePriceResponse">
3413         </output>
3414     </operation>
3415   </portType>
3416 </definitions>

```

3417

3418 **Generated C++ file:**

```

3419 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3420 //   serviceName="StockQuoteService")
3421 class StockQuoteService {
3422
3423   // @WebFunction(operationName="GetLastTradePrice",
3424   //   action="urn:GetLastTradePrice")
3425   commonj::sdo::DataObjectPtr
3426   getLastTradePrice(commonj::sdo::DataObjectPtr parameters);
3427 };

```

3428 [Snippet D-4: Example <cpp:enableWrapperStyle> Element](#)

## 3429 **D.4 <cpp:namespace>**

3430 <cpp:namespace> specifies the name of the C++ namespace that the associated WSDL element (and  
3431 any of it's children) are created in.

3432

3433 **Format:**

```

3434 <cpp:namespace name="namespaceURI" />

```

3435 [Snippet D-5: <cpp:namespace> Element Format](#)

3436 where:

- 3437 • **namespace/@name : anyURI (1..1)** – specifies the name of the C++ namespace associated with  
3438 this WSDL element.

3439

3440 **Applicable WSDL element(s):**

- 3441 • wsdl:definitions

3442  
3443 A `<cpp:bindings/>` element MUST NOT have more than one `<cpp:namespace/>` child element.  
3444 [CPPD0004]

3445  
3446 Example:

3447 Input WSDL file:

```
3448 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3449             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3450             xmlns:tns="http://www.example.org/"
3451             xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3452             targetNamespace="http://www.example.org/">
3453   <cpp:bindings>
3454     <cpp:namespace name="stock"/>
3455   </cpp:bindings>
3456
3457   <portType name="StockQuote">
3458     <cpp:bindings>
3459       <cpp:class name="StockQuoteService"/>
3460     </cpp:bindings>
3461   </portType>
3462 </definitions>
```

3463  
3464 Generated C++ file:

```
3465 namespace stock
3466 {
3467   // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3468   //           serviceName="StockQuoteService")
3469   // @WebService(name="StockQuote",
3470   class StockQuoteService {
3471   };
3472 }
```

3473 [Snippet D-6: Example <cpp:namespace> Element](#)

## 3474 D.5 <cpp:memberFunction>

3475 <cpp:memberFunction> specifies the name of the C++ member function that the associated WSDL  
3476 operation is associated with.

3477  
3478 **Format:**

```
3479 <cpp:memberFunction name="myFunction"/>
```

3480 [Snippet D-7: <cpp:memberFunction> Element Format](#)

3481 where:

- 3482 • **memberFunction/@name : NCName (1..1)** – specifies the name of the C++ member function  
3483 associated with this WSDL operation.

3484  
3485 **Applicable WSDL element(s):**

- 3486 • wsd:portType/wsd:operation

3487  
3488 A `<cpp:bindings/>` element MUST NOT have more than one `<cpp:memberFunction/>` child element.  
3489 [CPPD0005]

3490

3491 Example:

3492 Input WSDL file:

```
3493 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3494             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3495             xmlns:tns="http://www.example.org/"
3496             xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3497             targetNamespace="http://www.example.org/"
3498
3499             <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3500                       xmlns:tns="http://www.example.org/"
3501                       attributeFormDefault="unqualified"
3502                       elementFormDefault="unqualified"
3503                       targetNamespace="http://www.example.org/"
3504                       <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice" />
3505                       <xs:element name="GetLastTradePriceResponse"
3506                                   type="tns:GetLastTradePriceResponse" />
3507                       <xs:complexType name="GetLastTradePrice">
3508                         <xs:sequence>
3509                           <xs:element name="tickerSymbol" type="xs:string" />
3510                         </xs:sequence>
3511                       </xs:complexType>
3512                       <xs:complexType name="GetLastTradePriceResponse">
3513                         <xs:sequence>
3514                           <xs:element name="return" type="xs:float" />
3515                         </xs:sequence>
3516                       </xs:complexType>
3517                     </xs:schema>
3518
3519             <-message name="GetLastTradePrice">
3520               <part name="parameters" element="tns:GetLastTradePrice">
3521                 </part>
3522             </message>
3523
3524             <-message name="GetLastTradePriceResponse">
3525               <part name="parameters" element="tns:GetLastTradePriceResponse">
3526                 </part>
3527             </-message>
3528
3529             <portType name="StockQuote">
3530               <cpp:bindings>
3531                 <cpp:class name="StockQuoteService" />
3532               </cpp:bindings>
3533               <operation name="GetLastTradePrice">
3534                 <cpp:bindings>
3535                   <cpp:memberFunction name="getTradePrice" />
3536                 </cpp:bindings>
3537                 <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3538                   </input>
3539                 <output name="GetLastTradePriceResponse"
3540                         message="tns:GetLastTradePriceResponse">
3541                   </output>
3542               </operation>
3543             </portType>
3544           </definitions>
```

3545

3546 Generated C++ file:

```
3547 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3548 //             serviceName="StockQuoteService")
3549 class StockQuoteService {
3550
3551     // @WebFunction(operationName="GetLastTradePrice",
3552     //               action="urn:GetLastTradePrice")
```

```
3553     float getTradePrice(const std::string& tickerSymbol);
3554     };
```

3555 [Snippet D-8: Example <cpp:memberFunction> Element](#)

## 3556 **A.5D.6 <cpp:parameter>**

3557 <cpp:parameter> specifies the name of the C++ member function parameter associated with a specific  
3558 WSDL message part or wrapper child element.

3559  
3560 **Format:**

```
3561 <cpp:parameter name="CPPParameter" part="WSDLPart"  
3562     childElementName="WSDLElement" type="CPPTyp" />
```

3563 [Snippet D-9: <cpp:parameter> Element Format](#)

3564 where:

- 3565 • **parameter/@name : NCName (1..1)** – specifies the name of the C++ member function parameter  
3566 associated with this WSDL operation. “return” is used to denote the return value.
- 3567 • **parameter/@part : string (1..1)** - an XPath expression identifying the wsdl:part of a wsdl:message.
- 3568 • **parameter/@childElementName : QName (1..1)** – specifies the qualified name of a child element of  
3569 the global element identified by parameter/@part.
- 3570 • **parameter/@type : NCNamestring (0..1)** – specifies the type of the parameter or struct member or  
3571 return type. The @type attribute of a <cpp:parameter/> element MUST be a C++ type specified in  
3572 Simple Content Binding. [CPPD0006] The default type is determined by the mapping defined in  
3573 Simple Content Binding.

3574  
3575 **Applicable WSDL element(s):**

- 3576 • wsdl:portType/wsdl:operation

3577  
3578 **Example:**

3579 Input WSDL file:

```
3580 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3581     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3582     xmlns:tns="http://www.example.org/"  
3583     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"  
3584     targetNamespace="http://www.example.org/">  
3585  
3586     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
3587         xmlns:tns="http://www.example.org/"  
3588         attributeFormDefault="unqualified"  
3589         elementFormDefault="unqualified"  
3590         targetNamespace="http://www.example.org/">  
3591         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice" />  
3592         <xs:element name="GetLastTradePriceResponse"  
3593             type="tns:GetLastTradePriceResponse" />  
3594         <xs:complexType name="GetLastTradePrice">  
3595             <xs:sequence>  
3596                 <xs:element name="symbol" type="xs:string" />  
3597             </xs:sequence>  
3598         </xs:complexType>  
3599         <xs:complexType name="GetLastTradePriceResponse">  
3600             <xs:sequence>  
3601                 <xs:element name="return" type="xs:float" />  
3602             </xs:sequence>  
3603         </xs:complexType>
```



```

3604 </xs:schema>
3605
3606 <-message name="GetLastTradePrice">
3607   <part name="parameters" element="tns:GetLastTradePrice">
3608     </part>
3609 </message>
3610
3611 <-message name="GetLastTradePriceResponse">
3612   <part name="parameters" element="tns:GetLastTradePriceResponse">
3613     </part>
3614 </-message>
3615
3616 <portType name="StockQuote">
3617   <cpp:bindings>
3618     <cpp:class name="StockQuoteService" />
3619   </cpp:bindings>
3620   <operation name="GetLastTradePrice">
3621     <cpp:bindings>
3622       <cpp:memberFunction name="getLastTradePrice" />
3623       <cpp:parameter name="tickerSymbol"
3624         part="tns:GetLastTradePrice/parameter"
3625         childElementName="symbol" />
3626     </cpp:bindings>
3627     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3628       </input>
3629     <output name="GetLastTradePriceResponse"
3630       message="tns:GetLastTradePriceResponse">
3631       </output>
3632     </operation>
3633 </portType>
3634
3635 <binding name="StockQuoteServiceSoapBinding">
3636   <soap:binding style="document"
3637     transport="http://schemas.xmlsoap.org/soap/http" />
3638   <wSDL:operation name="GetLastTradePrice">
3639     <soap:operation soapAction="urn:GetLastTradePrice" style="document" />
3640     <wSDL:input name="GetLastTradePrice">
3641       <soap:body use="literal" />
3642     </wSDL:input>
3643     <wSDL:output name="GetLastTradePriceResponse">
3644       <soap:body use="literal" />
3645     </wSDL:output>
3646   </wSDL:operation>
3647 </binding>
3648
3649 <service name="StockQuoteService">
3650   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3651     <soap:address location="REPLACE_WITH_ACTUAL_URL" />
3652   </port>
3653 </service>
3654 </definitions>

```

3655

#### 3656 Generated C++ file:

```

3657 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3658 //   serviceName="StockQuoteService")
3659 class StockQuoteService {
3660
3661   // @WebFunction(operationName="GetLastTradePrice",
3662   //   action="urn:GetLastTradePrice")
3663   // @WebParam(paramName="tickerSymbol", name="symbol")
3664   float getLastTradePrice(const std::string& tickerSymbol);
3665 };

```

3666 [Snippet D-10: Example <cpp:parameter> Element](#)

## 3667 **D.6D.7 JAX-WS WSDL Extensions**

3668 An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL  
3669 extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.  
3670 [Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted and their corresponding SCA](#)  
3671 [WSDL extensions.](#)

3672 [\[CPPD0007\]](#) The following is a list of JAX-WS WSDL extensions that MAY be recognized, and their  
3673 corresponding SCA WSDL extension.

3674

JAX-WS Extension	SCA Extension
jaxws:bindings	cpp:bindings
jaxws:class	cpp:class
jaxws:method	cpp:memberFunction
jaxws:parameter	cpp:parameter
jaxws:enableWrapperStyle	cpp:enableWrapperStyle

## 3675 **A.6 WSDL Extensions Schema**

3676 The XML schema pointed to by the RDDL document at the SCA C++ namespace URI, defined by this  
3677 specification, is considered to be authoritative and takes precedence over the XML schema in this  
3678 appendix.

3679

3680 [Table D-1: Allowed JAX-WS Extensions](#)

## 3681 **D.8 sca-wsdlext-cpp-1.1.xsd**

```
3682 <?xml version="1.0" encoding="UTF-8"?>
3683 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3684         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-
3685 cpp/cpp/200901"
3686         xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3687         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3688         elementFormDefault="qualified">
3689
3690     <element name="bindings" type="cpp:BindingsType" />
3691     <complexType name="BindingsType">
3692         <choice minOccurs="0" maxOccurs="unbounded">
3693             <element ref="cpp:namespace" />
3694             <element ref="cpp:class" />
3695             <element ref="cpp:enableWrapperStyle" />
3696             <element ref="cpp:memberFunction" />
3697             <element ref="cpp:parameter" />
3698         </choice>
3699     </complexType>
3700
3701     <element name="namespace" type="cpp:NamespaceType" />
3702     <complexType name="NamespaceType">
3703         <attribute name="name" type="xsd:anyURI" use="required" />
3704     </complexType>
3705
3706     <element name="class" type="cpp:ClassType" />
3707     <complexType name="ClassType">
```

```
3708         <attribute name="name" type="xsd:NCName" use="required" />
3709     </complexType>
3710
3711     <element name="memberFunction" type="cpp:MemberFunctionType" />
3712     <complexType name="MemberFunctionType">
3713         <attribute name="name" type="xsd:NCName" use="required" />
3714     </complexType>
3715
3716     <element name="parameter" type="cpp:ParameterType" />
3717     <complexType name="ParameterType">
3718         <attribute name="part" type="xsd:string" use="required" />
3719         <attribute name="childElementName" type="xsd:QName"
3720             use="required" />
3721         <attribute name="name" type="xsd:NCName" use="required" />
3722         <attribute name="type" type="xsd:string" use="optional" />
3723     </complexType>
3724
3725     <element name="enableWrapperStyle" type="xsd:boolean" />
3726 </schema>
```

3727 | [Snippet D-11: SCA C++ WSDL Extension Schema](#)

3728

## E XML Schemas

3729  
3730

Three XML schemas are defined to support the use of C++ for implementation and definition of interfaces.

3731

3732  
3733  
3734

The XML schema pointed to by the RDDL document at the SCA namespace URI, defined by the Assembly specification [ASSEMBLY] and extended by this specification, are considered to be authoritative and take precedence over the XML schema in this appendix.

3735

### E.1 sca-interface-cpp-1.1.xsd

3736  
3737  
3738  
3739  
3740  
3741  
3742  
3743  
3744  
3745  
3746  
3747  
3748  
3749  
3750  
3751  
3752  
3753  
3754  
3755  
3756  
3757  
3758  
3759  
3760  
3761  
3762  
3763  
3764  
3765  
3766  
3767  
3768  
3769  
3770  
3771  
3772  
3773  
3774  
3775  
3776  
3777  
3778  
3779  
3780  
3781  
3782  
3783

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
  elementFormDefault="qualified">

  <include schemaLocation="sca-core.xsd" />

  <element name="interface.cpp" type="sca:CPPInterface"
    substitutionGroup="sca:interface" />

  <complexType name="CPPInterface">
    <complexContent>
      <extension base="sca:Interface">
        <sequence>
          <element name="function" type="sca:CPPFunction"
            minOccurs="0" maxOccurs="unbounded" />
          <element name="callbackFunction" type="sca:CPPFunction"
            minOccurs="0" maxOccurs="unbounded" />
          <any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded" />
        </sequence>
        <attribute name="header" type="string" use="required" />
        <attribute name="class" type="Name" use="required" />
        <attribute name="callbackHeader" type="string" use="optional" />
        <attribute name="callbackClass" type="Name" use="optional" />
        <del><anyAttribute namespace="##other" processContents="lax" /></del>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="CPPFunction">
    <del><sequence>
      <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="sca:requires" />
        <element ref="sca:policySetAttachment" />
      </choice>
      <any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded" />
    </del></sequence>
    <attribute name="name" type="NCName" use="required" />
    <attribute name="requires" type="sca:listOfQNames" use="optional" />
    <del><attribute name="policySets" type="sca:listOfQNames" use="optional" /></del>
    <attribute name="oneWay" type="boolean" use="optional" />
    <del><attribute name="exclude" type="boolean" use="optional" /></del>
    <anyAttribute namespace="##other" processContents="lax" />
  </complexType>
```

3784 </schema>

3785 [Snippet E-1: SCA <interface.cpp> Schema](#)

## 3786 E.2 sca-implementation-cpp-1.1.xsd

```
3787 <?xml version="1.0" encoding="UTF-8"?>
3788 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3789         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
3790         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
3791         elementFormDefault="qualified">
3792
3793     <include schemaLocation="sca-core.xsd" />
3794
3795     <element name="implementation.cpp" type="sca:CPPImplementation"
3796             substitutionGroup="sca:implementation" />
3797     <complexType name="CPPImplementation">
3798         <complexContent>
3799             <extension base="sca:Implementation">
3800                 <sequence>
3801                     <element name="function" type="sca:CPPImplementationFunction"
3802                             minOccurs="0" maxOccurs="unbounded" />
3803                     <any namespace="##other" processContents="lax"
3804                             minOccurs="0" maxOccurs="unbounded" />
3805                 </sequence>
3806                 <attribute name="library" type="NCName" use="required" />
3807                 <attribute name="header" type="NCName" use="required" />
3808                 <attribute name="path" type="string" use="optional" />
3809                 <attribute name="class" type="Name" use="optional" />
3810                 <attribute name="componentType" type="string" use="optional" />
3811                 <attribute name="scope" type="sca:CPPImplementationScope"
3812                             use="optional" />
3813                 <attribute name="eagerInit" type="boolean" use="optional" />
3814                 <attribute name="allowsPassByReference" type="boolean"
3815                             use="optional" />
3816                 <anyAttribute namespace="##other" processContents="lax" />
3817             </extension>
3818         </complexContent>
3819     </complexType>
3820
3821     <simpleType name="CPPImplementationScope">
3822         <restriction base="string">
3823             <enumeration value="stateless" />
3824             <enumeration value="composite" />
3825         </restriction>
3826     </simpleType>
3827
3828     <complexType name="CPPImplementationFunction">
3829         <sequence>
3830             <choice minOccurs="0" maxOccurs="unbounded">
3831                 <element ref="sca:requires" />
3832                 <element ref="sca:policySetAttachment" />
3833             </choice>
3834             <any namespace="##other" processContents="lax" minOccurs="0"
3835                     maxOccurs="unbounded" />
3836         </sequence>
3837         <attribute name="name" type="NCName" use="required" />
3838         <attribute name="requires" type="sca:listOfQNames" use="optional" />
3839         <attribute name="policySets" type="sca:listOfQNames" use="optional" />
3840         <attribute name="allowsPassByReference" type="boolean"
3841                 use="optional" />
3842         <anyAttribute namespace="##other" processContents="lax" />
3843     </complexType>
```

3844  
3845

```
</schema>
```

3846 [Snippet E-2 : SCA <implementation.cpp> Schema](#)

### 3847 E.3 sca-contribution-cpp-1.1.xsd

```
3848 <?xml version="1.0" encoding="UTF-8"?>
3849 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3850         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
3851         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
3852         elementFormDefault="qualified">
3853
3854     <include schemaLocation="sca-contributions.xsd"/>
3855
3856     <element name="export.cpp" type="sca:CPPExport"
3857             substitutionGroup="sca:Export"/>
3858
3859     <complexType name="CPPExport">
3860         <complexContent>
3861             <attribute name="name" type="QName" use="required"/>
3862             <attribute name="path" type="string" use="optional"/>
3863         </complexContent>
3864     </complexType>
3865
3866     <element name="import.cpp" type="sca:CPPImport"
3867             substitutionGroup="sca:Import"/>
3868
3869     <complexType name="CPPImport">
3870         <complexContent>
3871             <attribute name="name" type="QName" use="required"/>
3872             <attribute name="location" type="string" use="required"/>
3873         </complexContent>
3874     </complexType>
3875
3876 </schema>
```

3877

---

## ~~B Conformance Items~~

3878

~~This section contains a list of conformance items for the SCA C++ Client and Implementation Model specification.~~

3879

3880

~~Snippet E-3: SCA <export.cpp> and <import.cpp> Schema~~

## F Normative Statement Summary

This section contains a list of normative statements for this specification.

Conformance ID	Description
[CPP20001]	A C++ implementation MUST implement all of the operation(s) of the service interface(s) of its componentType.
[CPP20003]	An SCA runtime MUST support these scopes; <b>stateless</b> and <b>composite</b> . Additional scopes MAY be provided by SCA runtimes.
[CPP20005]	If the header file identified by the @header attribute of an <interface.cpp/> element contains more than one class, then the @class attribute MUST be specified for the <interface.cpp/> element.
[CPP20006]	If the header file identified by the @callbackHeader attribute of an <interface.cpp/> element contains more than one class, then the @callbackClass attribute MUST be specified for the <interface.cpp/> element.
[CPP20007]	The @name attribute of a <function/> child element of a <interface.cpp/> MUST be unique amongst the <function/> elements of that <interface.cpp/>.
[CPP20008]	The @name attribute of a <callbackFunction/> child element of a <interface.cpp/> MUST be unique amongst the <callbackFunction/> elements of that <interface.cpp/>.
[CPP20009]	The name of the componentType file for a C++ implementation MUST match the class name (excluding any namespace definition) of the implementations as defined by the @class attribute of the <implementation.cpp/> element.
[CPP20010]	The @name attribute of a <function/> child element of a <implementation.cpp/> MUST be unique amongst the <function/> elements of that <implementation.cpp/>.
[CPP20011]	A C++ implementation class MUST be default constructable by the SCA runtime to instantiate the component.
[CPP20012]	An SCA runtime MUST ensure that a stateless scoped implementation instance object is only ever dispatched on one thread at any one time. In addition, within the SCA lifecycle of an instance, an SCA runtime MUST only make a single invocation of one business <del>method</del> member function.
[CPP20013]	An SCA runtime MAY run multiple threads in a single composite scoped implementation instance object <del>and it MUST NOT perform any synchronization.</del>
[CPP20014]	The SCA runtime MAY use by-reference semantics when passing input parameters, return values or exceptions on calls to remotable services within the same system address space if both the service member function implementation and the client are marked "allows pass by reference".
[CPP20015]	The SCA runtime MUST use by-value semantics when passing input parameters, return values and exceptions on calls to remotable services within the same system address space if the service member function implementation is not marked "allows pass by reference" or the client is not marked "allows pass by reference".



Conformance ID	Description
[CPP20016]	If the header file identified by the <code>@header</code> attribute of an <code>&lt;interface.cpp/&gt;</code> element contains function declarations that are not operations of the interface, then the functions that are not operations of the interface MUST be excluded using <code>&lt;function/&gt;</code> child elements of the <code>&lt;interface.cpp/&gt;</code> element with <code>@exclude="true"</code> .
[CPP20017]	If the header file identified by the <code>@callbackHeader</code> attribute of an <code>&lt;interface.cpp/&gt;</code> element contains function declarations that are not operations of the callback interface, then the functions that are not operations of the callback interface MUST be excluded using <code>&lt;callbackFunction/&gt;</code> child elements of the <code>&lt;interface.cpp/&gt;</code> element with <code>@exclude="true"</code> .
[CPP20018]	An SCA runtime MUST NOT perform any synchronization of access to component implementations.
[CPP30001]	If a remotable interface is defined with a C++ class, an SCA implementation SHOULD map the interface definition to WSDL before generating the proxy for the interface.
[CPP30002]	For each reference of a component, an SCA implementation MUST generate a service proxy derived from <code>ServiceProxy</code> that contains the operations of the reference's interface definition.
[CPP30003]	An SCA runtime MUST include an asynchronous invocation member function for every operation of a reference interface with a <code>@requires="asyncInvocation"</code> intent applied either to the operation or the reference as a whole.
[CPP30004]	An SCA runtime MUST include a response class for every response message of a reference interface that can be returned by an operation of the interface with a <code>@requires="asyncInvocation"</code> intent applied either to the operation of the reference as a whole.
[CPP40001]	An operation marked as <code>oneWay</code> is considered non-blocking and the SCA runtime MAY use a binding that buffers the requests to the member function and sends them at some time after they are made.
[CPP40002]	For each service of a component that includes a bidirectional interface, an SCA implementation MUST generate a service proxy derived from <code>ServiceProxy</code> that contains the operations of the reference's callback interface definition.
[CPP40003]	If a service of a component that has a callback interface contains operations with a <code>@requires="asyncInvocation"</code> intent applied either to the operation of the reference as a whole, an SCA implementation MUST include asynchronous invocation member functions and response classes as described in Long Running Request-Response Operations.
[CPP70001]	The <code>@name</code> attribute of a <code>&lt;export.cpp/&gt;</code> element MUST be unique amongst the <code>&lt;export.cpp/&gt;</code> elements in a domain.
[CPP70002]	The <code>@name</code> attribute of a <code>&lt;import.cpp/&gt;</code> child element of a <code>&lt;contribution/&gt;</code> MUST be unique amongst the <code>&lt;import.cpp/&gt;</code> elements in of that contribution.

Conformance ID	Description
[CPP80001]	<p>The return type and types of the parameters of a member function of a local service interface <del>MUST</del> be one of:</p> <ul style="list-style-type: none"> <li>• <del>Any of the C++ primitive types (for example, int, short, char). In this case the data will be passed by value as is normal for C++.</del></li> <li>• <del>Pointers to any of the C++ primitive types (for example, int *, short *, char *).</del></li> <li>• <del>The const keyword can be used for any pointer to a C++ primitive type (for example const char *). If this is used on a parameter then the destination can not change the value.</del></li> <li>• <del>C++ class. The class will be passed by value as is normal for C++.</del></li> <li>• <del>Pointer to a C++ class. A pointer will be passed to the destination which can then modify the original contents.</del></li> <li>• <del>DataObjectPtr. An SDO pointer. This will be passed by reference.</del></li> </ul> <p><u>References to C++ classes (passed by reference). An SCA implementation MUST translate a class to tokens as part of conversion to WSDL or compatibility testing.</u></p>
[CPP80002]	<p>The return type and types of the parameters of a member function of a remotable service interface <u>MUST</u> be one of:</p> <ul style="list-style-type: none"> <li>• <u>Any of the C++ primitive types (for example, int, short, char). This will <u>Any of the C++ types specified in Simple Content Binding. These types may be copied.</u></u></li> <li>• <u>DataObjectPtr. An SDO pointer. The SDO will be copied and passed to the destination by-value, by-reference, or by-pointer. Unless the member function and client indicate that they allow by-reference semantics (see AllowsPassByReference), a copy will be explicitly created by the runtime for any parameters passed by-reference or by-pointer.</u></li> <li>• <u>An SDO DataObjectPtr instance. This type may be passed by-value, by-reference, or by-pointer. Unless the member function and client indicate that they allow by-reference semantics (see AllowsPassByReference), a deep-copy of the DataObjectPtr will be created by the runtime for any parameters passed by-value, by-reference, or by-pointer. When by-reference semantics are allowed, the DataObjectPtr itself will be passed.</u></li> </ul>
[CPP80003]	<p>A C++ header file used to define an interface <u>MUST</u>:</p> <p><u>Declare declare</u> at least one class with:</p> <ul style="list-style-type: none"> <li>• <u>At least one public member function.</u></li> <li>• <u>All public member functions <u>MUST be</u> pure virtual (virtual with no implementation).</u></li> </ul>
[CPP90002]	<p>A C++ header file used to define an interface <del>MUST NOT</del> use the following constructs:</p> <ul style="list-style-type: none"> <li>• <del>Macros</del></li> <li>• <del>Inline member functions</del></li> <li>• <del>Friend classes</del></li> </ul>
[CPP100001]	<p>A WSDL file might define a namespace using the <u>&lt;sca:namespace&gt;</u> WSDL extension, otherwise all C++ classes <u>MUST</u> be placed in a default namespace as determined by the implementation. Implementations <u>SHOULD</u> provide a mechanism for overriding the default namespace.</p>

Conformance ID	Description
[CPP100002]	If multiple operations within the same portType indicate that they throw faults that reference the same global element, an SCA implementation MUST generate a single C++ exception class with each C++ member function referencing this class in its @WebThrows annotation.
[CPP100003]	<ul style="list-style-type: none"> <li>For unwrapped messages, an SCA implementation MUST map: <ul style="list-style-type: none"> <li><b>in</b> - the message part to a member function parameter, passed by const-reference.</li> <li><b>out</b> - the message part to a member function parameter, passed by reference, or to the member function return type, returned by-value.</li> <li><b>in/out</b> - the message part to a member function parameter, passed by reference.</li> </ul> </li> </ul>
[CPP100004]	<ul style="list-style-type: none"> <li>For wrapped messages, an SCA implementation MUST map: <ul style="list-style-type: none"> <li><b>in</b> - the wrapper child to a member function parameter, passed by const-reference.</li> <li><b>out</b> - the wrapper child to a member function parameter, passed by reference, or to the member function return type, returned by-value.</li> <li><b>in/out</b> - the wrapper child to a member function parameter, passed by reference.</li> </ul> </li> </ul>
[CPP100005]	An SCA implementation SHOULD provide a mechanism for overriding the default targetNamespace.
[CPP100006]	An SCA implementation MUST map a method's return type as an <b>out</b> parameter, a parameter passed by-reference or by-pointer as an <b>in/out</b> parameter, and all other parameters, including those passed by-const-reference as <b>in</b> parameters.
[CPP100008]	<del>An SCA implementation MUST ensure each class that is referenced from an @WebThrows annotation MUST itself have a @WebFault annotation that associates the fault with a particular global element that will be associated with the fault message. An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.</del>
[CPP100009]	<del>An SCA implementation MUST map simple types as defined in Table 1 and Table 2 by default. An SCA implementation MUST map a WSDL portType to a remotable C++ interface definition.</del>
[CPP100010]	<del>An SCA implementation MUST reject a composite file that does not conform to <a href="http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-cpp-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-cpp-1.1.xsd</a> or <a href="http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-cpp-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-cpp-1.1.xsd</a>. An SCA implementation MUST map a C++ interface definition to WSDL as if it has a @WebService annotation with all default values on the class.</del>
[CPP110001]	<del>An SCA implementation MUST reject a componentType or constraining type file that does not conform to <a href="http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-cpp-1.1.xsd</a>; or <a href="http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-cpp-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-cpp-1.1.xsd</a>.</del>
[CPP110002]	<del>An SCA implementation MUST reject a contribution file that does not conform to <a href="http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contribution-interface-cpp-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contribution-interface-cpp-1.1.xsd</a>.</del>

Conformance ID	Description
[CPP110003]	An SCA implementation MUST reject a contribution file that does not conform to <a href="http://docs.oasis-open.org/opensca/sca/200912/sca-contribution-cpp-1.1.xsd">http://docs.oasis-open.org/opensca/sca/200912/sca-contribution-cpp-1.1.xsd</a> .
[CPP110004]	An SCA implementation MUST reject a WSDL file that does not conform to <a href="http://docs.oasis-open.org/opensca/sca-c-cpp/cpp/200901/sca-wsdlex-cpp-1.1.xsd">http://docs.oasis-open.org/opensca/sca-c-cpp/cpp/200901/sca-wsdlex-cpp-1.1.xsd</a> .

3883 *Table F-1: SCA C++ Core Normative Statements*

## 3884 **F.1 Annotation Normative Statement Summary**

3885 This section contains a list of normative statements related to source file annotations for this specification.

Conformance ID	Description
[CPPA0001]	If annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as described. The SCA runtime MUST only process the SCDL files and not the annotations.
[CPPA0002]	If multiple annotations apply to a program element, all of the annotations SHOULD be in the same comment block.
[CPPA0003]	An SCA implementation MUST treat a class with an @WebService annotation specified as if an @Interface @Remotable annotation was specified.
[CPPA0004]	An SCA implementation MUST treat a member function with a @WebFunction annotation specified as if @Function was specified with the operationName value of the @WebFunction annotation used as the name value of the @Function annotation and the exclude value of the @WebFunction annotation used as the exclude value of the @Function annotation.
[CPPC0001]	If WSDL mapping annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to WSDL as described.
[CPPC0002]	An SCA implementation MUST treat any instance of a @InterfaceRemotable annotation and without an explicit @WebService annotation as if a @WebService annotation with no parameters was specified.
[CPPC0004]	<del>If a @WebService does not have a portName element, an SCA implementation MUST use the value associated with the name element, suffixed with "Port". The value of the paramName of a @WebParam annotation MUST be the name of a parameter of the member function the annotation is applied to.</del>
[CPPC0004]	<del>Only named parameters MAY be referenced by a @WebParam annotation.</del>
[CPPC0005]	The value of the type property of a @WebParam annotation MUST be one of the simpleTypes defined in namespace <a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a> .
[CPPC0006]	The value of the type property of a @WebResult annotation MUST be one of the simpleTypes defined in namespace <a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a> .
[CPPC0007]	A C++ class with a @WebFault annotation MUST provide a constructor that takes two parameters, a std::string and a type representing the fault information. Additionally, the class MUST provide a const member function "getFaultInfo" that takes no parameters, and returns the same type as defined in the constructor.
[CPPC0008]	A C++ class that is listed in a @WebThrows annotation MUST itself have a @WebFault annotation.

<u>Conformance ID</u>	<u>Description</u>
[CPPC0009]	An SCA implementation MUST treat a member function annotated with an @Function annotation and without an explicit @WebFunction annotation as if a @WebFunction annotation with with an operationName value equal to the name value of the @Function annotation, an exclude value equal to the exclude value of the @Operation annotation and no other parameters was specified.

3886 [Table F-2: SCA C++ Annotation Normative Statements](#)

## 3887 F.2 WSDL Extention Normative Statement Summary

3888 This section contains a list of normative statements related to WSDL extensions for this specification.

<u>Conformance ID</u>	<u>Description</u>
[CPPD0001]	If WSDL extensions are supported by an implementation, all the extensions defined here MUST be supported and MUST be mapped to C++ as described.
[CPPD0002]	A <cpp:bindings/> element MUST NOT have more than one <cpp:class/> child element.
[CPPD0003]	A <cpp:bindings/> element MUST NOT have more than one <cpp:enableWrapperStyle/> child element.
[CPPD0004]	A <cpp:bindings/> element MUST NOT have more than one <cpp:namespace/> child element.
[CPPD0005]	A <cpp:bindings/> element MUST NOT have more than one <cpp:memberFunction/> child element.
[CPPD0006]	The @type attribute of a <cpp:parameter/> element MUST be a valid C++ type- <u>specified in</u> Simple Content Binding.
[CPPD0007]	An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present. <a href="#">Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted and their corresponding SCA WSDL extensions.</a>

3889 [Table F-3 SCA C++ WSDL Extension Normative Statements](#)

## 3890 E.4F.3 JAX-WS Conformance Normative Statements

3891 The JAX-WS 2.1 specification [JAXWS21] defines conformance normative statements for various  
 3892 requirements defined by that specification. The following table Table F-4 outlines those conformance  
 3893 statements, and describes whether the conformance statement applies normative statemetns which apply  
 3894 to the WSDL binding mapping described in this specification.

<u>SectionNumber</u>	<u>Conformance StatementPoint</u>	<u>Notes</u>	<u>Conformance ID</u>
<u>2.1</u>	WSDL 1.1 support	[A]	[CPPF0001]
<u>2.2</u>	Customization required	[CPPD0001] The reference to the JAX-WS binding language <u>are is</u> treated as a reference to the C++ WSDL extensions defined in section WSDL C++ Mapping Extensions.	

<u>SectionNumber</u>	<u>Conformance StatementPoint</u>	Notes	Conformance ID
<u>2.3</u>	Annotations on generated classes		[CPPF0002]
<u>2.44</u>	Definitions mapping	[CPP100001]	
<u>2.45</u>	WSDL and XML Schema import directives		[CPPF0003]
<u>2.4.46</u>	Optional WSDL extensions		[CPPF0004]
<u>2.27</u>	SEI naming		[CPPF0005]
<u>2.28</u>	javax.jws.WebService required	[B] References to javax.jws.WebService in the conformance statement are treated as the C++ annotation @WebService.	[CPPF0006]
<u>2.310</u>	Method naming		[CPPF0007]
<u>2.311</u>	javax.jws.WebMethod required	[A], [B] References to javax.jws.WebMethod in the conformance statement are treated as the C++ annotation @WebFunction.	[CPPF0008]
<u>2.312</u>	Transmission primitive support		[CPPF0009]
<u>2.313</u>	Using javax.jws.OneWay	[A], [B] References to javax.jws.OneWay in the conformance statement are treated as the C++ annotation @OneWay.	[CPPF0010]
<u>2.3.414</u>	Using javax.jws.SOAPBinding	[A], [B] References to javax.jws.SOAPBinding in the conformance statement are treated as the C++ annotation @SOAPBinding.	[CPPF0011]
<u>2.3.415</u>	Using javax.jws.WebParam	[A], [B] References to javax.jws.WebParam in the conformance statement are treated as the C++ annotation @WebParam.	[CPPF0012]

<u>SectionNumber</u>	<u>Conformance StatementPoint</u>	Notes	Conformance ID
<a href="#">2.3.4.16</a>	Using javax.jws.WebResult	[A], [B] References to javax.jws.WebResult in the conformance statement are treated as the C++ annotation @WebResult.	[CPPF0013]
<a href="#">2.3.4.18</a>	Non-wrapped parameter naming		[CPPF0014]
<a href="#">2.3.4.219</a>	Default mapping mode		[CPPF0015]
<a href="#">2.3.4.220</a>	Disabling wrapper style	[B] References to jaxws:enableWrapperStyle in the conformance statement are treated as the WSDL extension cpp:enableWrapperStyle.	[CPPF0016]
<a href="#">2.3.4.221</a>	Wrapped parameter naming		[CPPF0017]
<a href="#">2.3.4.222</a>	Parameter name clash	[A]	[CPPF0018]
<a href="#">2.538</a>	javax.xml.ws.WebFault required	[B] References to javax.jws.WebFault in the conformance statement are treated as the C++ annotation @WebFault.	[CPPF0019]
<a href="#">2.539</a>	Exception naming		[CPPF0020]
<a href="#">2.540</a>	Fault equivalence	<del>[A]</del> [CPP100002]	[CPPF0024]
<a href="#">2.642</a>	Required WSDL extensions	MIME Binding not necessary	[CPPF0022]
<a href="#">2.6.443</a>	Unbound message parts	[A]	[CPPF0023]
<a href="#">2.6.2.144</a>	Duplicate headers in binding		[CPPF0024]
<a href="#">2.6.2.145</a>	Duplicate headers in message		[CPPF0025]
<a href="#">3.1</a>	WSDL 1.1 support	[A]	[CPPF0026]
<a href="#">3.2</a>	Standard annotations	[A] [CPPC0001]	
<a href="#">3.43</a>	Java identifier mapping	[A]	[CPPF0027]

<u>SectionNumber</u>	<u>Conformance StatementPoint</u>	Notes	Conformance ID
<u>3.4.14</u>	Method name disambiguation	[A] References to javax.jws.WebMethod in the conformance statement are treated as the C++ annotation @WebFunction.	[CPPF0028]
<u>3.26</u>	WSDL and XML Schema import directives		[CPPF0029]
<u>3.48</u>	portType naming		[CPPF0030]
<u>3.4.49</u>	Inheritance flattening	[A]	[CPPF0044]
<u>3.4.410</u>	Inherited interface mapping		[CPPF0045]
<u>3.511</u>	Operation naming		[CPPF0031]
<u>3.5.412</u>	One-way mapping	[B] References to javax.jws.OneWay in the conformance statement are treated as the C++ annotation @OneWay.	[CPPF0032]
<u>3.5.413</u>	One-way mapping errors		[CPPF0033]
<u>3.6.415</u>	Parameter classification	[CPP100006]	
<u>3.6.416</u>	Parameter naming		[CPPF0035]
<u>3.6.417</u>	Result naming		[CPPF0036]
<u>3.6.418</u>	Header mapping of parameters and results	References to javax.jws.WebParam in the conformance statement are treated as the C++ annotation @WebParam.  References to javax.jws.WebResult in the conformance statement are treated as the C++ annotation @WebResult.	[CPPF0037]



<b>SectionNumber</b>	<b>Conformance StatementPoint</b>	<b>Notes</b>	<b>Conformance ID</b>
3.727	Exception namingBinding selection	[A] References to javax.jws.WebFault in the conformance statement are treated as the C++ annotation @WebFault. References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CPPF0038][CPPF0039]
3.828	Binding-selectionSOAP binding support	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY]. [A]	[CPPF0039][CPPF0040]
3.4029	SOAP binding supportstyle required	[A]	[CPPF0040][CPPF0041]
3.40.431	SOAP binding style requiredPort selection		[CPPF0041][CPPF0042]
3.4432	Port selectionbinding	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CPPF0042][CPPF0043]

3.11 Port-binding References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY]. [CPPF0043]

3895 [A] All references to Java in the conformance statementpoint are treated as references to C++.

3896 [B] Annotation generation is only necessary if annotations are supported by an SCA implementation.

### 3897 B.1.1 Ignored Conformance Statements

<b>Section</b>	<b>Conformance Statement</b>	<b>Notes</b>
2.2	javax.xml.bind.XmlSeeAlso required	
2.3.1	use of JAXB annotations	
2.3.1.2	Using javax.xml.ws.RequestWrapper	
2.3.1.2	Using javax.xml.ws.ResponseWrapper	
2.3.3	Use of Holder	
2.3.4	Asynchronous mapping required	
2.3.4	Asynchronous mapping option	
2.3.4.2	Asynchronous method naming	
2.3.4.2	Asynchronous parameter naming	

2.3.4.2	Failed method invocation	
2.3.4.4	Response bean naming	
2.3.4.5	Asynchronous fault reporting	
2.3.4.5	Asynchronous fault cause	
2.4	JAXB class mapping	
2.4	JAXB customization use	
2.4	JAXB customization clash	
2.4.4	javax.xml.ws.wsaddressing.W3CEndpointReference	
2.5	Fault Equivalence	
2.6.3.1	Use of MIME type information	
2.6.3.1	MIME type mismatch	
2.6.3.1	MIME part identification	
2.7	Service superclass required	
2.7	Service class naming	
2.7	javax.xml.ws.WebServiceClient required	
2.7	Default constructor required	
2.7	2 argument constructor required	
2.7	Failed getPort Method	
2.7	javax.xml.ws.WebEndpoint required	
3.2	Package name mapping	
3.3	Class mapping	
3.6	use of JAXB annotations	
3.6.2.1	Default wrapper bean names	
3.6.2.1	Default wrapper bean package	
3.6.2.3	Null Values in rpc/literal	
3.7	java.lang.RuntimeExceptions and java.rmi.RemoteExceptions	
3.7	Fault bean name clash	
3.11	Service creation	

3898 *Table F-4: JAX-WS Normative Statements that are Applicable to SCA C++*

3899 **F.3.1 Ignored Normative Statements**

<b>Number</b>	<b>Conformance Point</b>
---------------	--------------------------

<b>Number</b>	<b>Conformance Point</b>
<a href="#">2.9</a>	<a href="#">javax.xml.bind.XmlSeeAlso required</a>
<a href="#">2.17</a>	<a href="#">use of JAXB annotations</a>
<a href="#">2.23</a>	<a href="#">Using javax.xml.ws.RequestWrapper</a>
<a href="#">2.24</a>	<a href="#">Using javax.xml.ws.ResponseWrapper</a>
<a href="#">2.25</a>	<a href="#">Use of Holder</a>
<a href="#">2.26</a>	<a href="#">Asynchronous mapping required</a>
<a href="#">2.27</a>	<a href="#">Asynchronous mapping option</a>
<a href="#">2.28</a>	<a href="#">Asynchronous method naming</a>
<a href="#">2.29</a>	<a href="#">Asynchronous parameter naming</a>
<a href="#">2.30</a>	<a href="#">Failed method invocation</a>
<a href="#">2.31</a>	<a href="#">Response bean naming</a>
<a href="#">2.32</a>	<a href="#">Asynchronous fault reporting</a>
<a href="#">2.33</a>	<a href="#">Asynchronous fault cause</a>
<a href="#">2.34</a>	<a href="#">JAXB class mapping</a>
<a href="#">2.35</a>	<a href="#">JAXB customization use</a>
<a href="#">2.36</a>	<a href="#">JAXB customization clash</a>
<a href="#">2.37</a>	<a href="#">javax.xml.ws.wsaddressing.W3CEndpointReference</a>
<a href="#">2.41</a>	<a href="#">Fault Equivalence</a>
<a href="#">2.46</a>	<a href="#">Use of MIME type information</a>
<a href="#">2.47</a>	<a href="#">MIME type mismatch</a>
<a href="#">2.48</a>	<a href="#">MIME part identification</a>
<a href="#">2.49</a>	<a href="#">Service superclass required</a>
<a href="#">2.50</a>	<a href="#">Service class naming</a>
<a href="#">2.51</a>	<a href="#">javax.xml.ws.WebServiceClient required</a>
<a href="#">2.52</a>	<a href="#">Default constructor required</a>
<a href="#">2.53</a>	<a href="#">2 argument constructor required</a>
<a href="#">2.54</a>	<a href="#">Failed getPort Method</a>
<a href="#">2.55</a>	<a href="#">javax.xml.ws.WebEndpoint required</a>
<a href="#">3.5</a>	<a href="#">Package name mapping</a>
<a href="#">3.7</a>	<a href="#">Class mapping</a>
<a href="#">3.14</a>	<a href="#">use of JAXB annotations</a>
<a href="#">3.19</a>	<a href="#">Default wrapper bean names</a>

<u>Number</u>	<u>Conformance Point</u>
<u>3.20</u>	<u>Default wrapper bean package</u>
<u>3.21</u>	<u>Null Values in rpc/literal</u>
<u>3.24</u>	<u>Exception naming</u>
<u>3.25</u>	<u>java.lang.RuntimeExceptions and java.rmi.RemoteExceptions</u>
<u>3.26</u>	<u>Fault bean name clash</u>
<u>3.30</u>	<u>Service creation</u>

3900

Table F-5: JAX-WS Normative Statements that Are Not Applicable to SCA C++

3901

---

## **FG Migration**

3902

To aid migration of an implementation or clients using an implementation based the version of the Service

3903

Component Architecture for C++ defined in [OSOA SCA C++ Client and Implementation V1.00](#), this

3904

appendix identifies the relevant changes to APIs, annotations, or behavior defined in V1.00.

3905

### **F.1G.1 Method child elements of `interface.cpp` and `implementation.cpp`**

3906

3907

The `<method/>` child element of `<interface.cpp/>` and the `<method/>` child element of

3908

`<implementation.cpp/>` have both been renamed to `<function/>` to be consistent with C++ terminology.

3909

---

## **GH Acknowledgements**

3910 The following individuals have participated in the creation of this specification and are gratefully  
3911 acknowledged:

3912 **Participants:**

3913

<b>Participant Name</b>	<b>Affiliation</b>
Bryan Aupperle	IBM
Andrew Borley	IBM
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
David Haney	Individual
Mark Little	Red Hat
Jeff Mischkinsky	Oracle Corporation
Peter Robbins	IBM

3914

---

## HI Revision History

3915

[optional; should not be included in OASIS Standards]

3916

Revision	Date	Editor	Changes Made
			•

3917