



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

Committee Draft 05 / Public Review Draft 03

4 March 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd05.html>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd05.doc>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd05.pdf> (Authoritative)

Previous 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)

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/200912>
<http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901>

C++ Artifacts:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-apis-cd05.zip>

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, 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 | 11 |
| 2.1.2 | AllowsPassByReference | 12 |
| 2.1.3 | Implementing a Local Service | 12 |
| 2.2 | Component Implementation Scopes | 13 |
| 2.2.1 | Stateless Scope | 13 |
| 2.2.2 | Composite Scope | 13 |
| 2.3 | Implementing a Configuration Property | 13 |
| 2.4 | Component Type and Component | 14 |
| 2.4.1 | Interface.cpp | 15 |
| 2.4.2 | Function and CallbackFunction | 16 |
| 2.4.3 | Implementation.cpp | 17 |
| 2.4.4 | Implementation Function | 18 |
| 2.5 | Instantiation..... | 19 |
| 3 | Basic Client Model..... | 20 |
| 3.1 | Accessing Services from Component Implementations | 20 |
| 3.2 | Interface Proxies | 21 |
| 3.3 | Accessing Services from non-SCA Component Implementations | 23 |
| 3.4 | Calling Service Operations | 23 |
| 3.5 | Long Running Request-Response Operations..... | 23 |
| 3.5.1 | Response Callback | 25 |
| 3.5.2 | Response Polling | 26 |
| 3.5.3 | Synchronous Response Access..... | 26 |
| 3.5.4 | Response Class | 27 |
| 4 | Asynchronous Programming | 29 |
| 4.1 | Non-blocking Calls | 29 |
| 4.2 | Callbacks | 29 |
| 4.2.1 | Using Callbacks..... | 30 |
| 4.2.2 | Callback Instance Management | 31 |
| 4.2.3 | Implementing Multiple Bidirectional Interfaces | 32 |
| 5 | Error Handling | 33 |
| 6 | C++ API | 34 |
| 6.1 | Reference Counting Pointers..... | 34 |
| 6.1.1 | operator* | 34 |
| 6.1.2 | operator-> | 35 |
| 6.1.3 | operator void* | 35 |

| | |
|---|----|
| 6.1.4 operator! | 35 |
| 6.1.5 constCast..... | 35 |
| 6.1.6 dynamicCast..... | 35 |
| 6.1.7 reinterpretCast..... | 36 |
| 6.1.8 staticCast..... | 36 |
| 6.2 Component Context..... | 36 |
| 6.2.1 getCurrent..... | 37 |
| 6.2.2 getURI | 37 |
| 6.2.3 getService..... | 37 |
| 6.2.4 getServices..... | 37 |
| 6.2.5 getServiceReference..... | 38 |
| 6.2.6 getServiceReferences | 38 |
| 6.2.7 getProperties | 38 |
| 6.2.8 getDataFactory..... | 39 |
| 6.2.9 getSelfReference..... | 39 |
| 6.3 ServiceReference | 39 |
| 6.3.1 getService..... | 40 |
| 6.3.2 getCallback..... | 40 |
| 6.4 DomainContext | 40 |
| 6.4.1 getService..... | 40 |
| 6.5 SCAException..... | 41 |
| 6.5.1 getEClassName | 41 |
| 6.5.2 getMessageText..... | 41 |
| 6.5.3 getFileName | 41 |
| 6.5.4 getLineNumber | 42 |
| 6.5.5 getFunctionName | 42 |
| 6.6 SCANullPointerException | 42 |
| 6.7 ServiceRuntimeException..... | 42 |
| 6.8 ServiceUnavailableException | 43 |
| 6.9 MultipleServicesException..... | 43 |
| 7 C++ Contributions..... | 44 |
| 7.1 Executable files..... | 44 |
| 7.1.1 Executable in contribution | 44 |
| 7.1.2 Executable shared with other contribution(s) (Export) | 44 |
| 7.1.3 Executable outside of contribution (Import)..... | 45 |
| 7.2 componentType files | 46 |
| 7.3 C++ Contribution Extensions | 47 |
| 7.3.1 Export.cpp | 47 |
| 7.3.2 Import.cpp..... | 47 |
| 8 C++ Interfaces | 48 |
| 8.1 Types Supported in Service Interfaces..... | 48 |
| 8.1.1 Local Service | 48 |
| 8.1.2 Remotable Service | 48 |
| 8.2 Header Files..... | 48 |
| 9 WSDL to C++ and C++ to WSDL Mapping | 49 |

| | |
|---|----|
| 9.1 Augmentations for WSDL to C++ Mapping | 49 |
| 9.1.1 Mapping WSDL targetNamespace to a C++ namespace | 49 |
| 9.1.2 Mapping WSDL Faults to C++ Exceptions | 50 |
| 9.1.3 Mapping of in, out, in/out parts to C++ member function parameters..... | 50 |
| 9.2 Augmentations for C++ to WSDL Mapping | 50 |
| 9.2.1 Mapping C++ namespaces to WSDL namespaces | 51 |
| 9.2.2 Parameter and return type classification..... | 51 |
| 9.2.3 C++ to WSDL Type Conversion | 51 |
| 9.2.4 Service-specific Exceptions..... | 51 |
| 9.3 SDO Data Binding | 51 |
| 9.3.1 Simple Content Binding | 51 |
| 9.3.2 Complex Content Binding..... | 53 |
| 10 Conformance | 54 |
| 10.1 Conformance Targets | 54 |
| 10.2 SCA Implementations | 54 |
| 10.3 SCA Documents | 55 |
| 10.4 C++ Files..... | 55 |
| 10.5 WSDL Files | 55 |
| A C++ SCA Annotations | 56 |
| A.1 Application of Annotations to C++ Program Elements..... | 56 |
| A.2 Interface Header Annotations..... | 56 |
| A.2.1 @Interface | 57 |
| A.2.2 @Remotable | 57 |
| A.2.3 @Callback..... | 57 |
| A.2.4 @OneWay | 58 |
| A.2.5 @Function..... | 59 |
| A.3 Implementation Header Annotations..... | 59 |
| A.3.1 @ComponentType..... | 59 |
| A.3.2 @Scope | 60 |
| A.3.3 @EagerInit | 60 |
| A.3.4 @AllowsPassByReference | 61 |
| A.3.5 @Property | 61 |
| A.3.6 @Reference | 62 |
| A.4 Base Annotation Grammar..... | 63 |
| B C++ SCA Policy Annotations..... | 64 |
| B.1 General Intent Annotations..... | 64 |
| B.2 Specific Intent Annotations | 65 |
| B.2.1 Security Interaction | 66 |
| B.2.2 Security Implementation | 66 |
| B.2.3 Reliable Messaging..... | 67 |
| B.2.4 Transactions..... | 67 |
| B.2.5 Miscellaneous | 67 |
| B.3 Policy Set Annotations | 67 |
| B.4 Policy Annotation Grammar Additions | 68 |
| B.5 Annotation Constants | 68 |

| | | |
|----------|---|------------|
| C | C++ WSDL Mapping Annotations | 69 |
| C.1 | Interface Header Annotations | 69 |
| C.1.1 | @WebService | 69 |
| C.1.2 | @WebFunction | 70 |
| C.1.3 | @OneWay | 72 |
| C.1.4 | @WebParam | 73 |
| C.1.5 | @WebResult | 75 |
| C.1.6 | @SOAPBinding | 77 |
| C.1.7 | @WebFault | 78 |
| C.1.8 | @WebThrows | 80 |
| D | WSDL C++ Mapping Extensions | 81 |
| D.1 | <cpp:bindings> | 81 |
| D.2 | <cpp:class> | 81 |
| D.3 | <cpp:enableWrapperStyle> | 82 |
| D.4 | <cpp:namespace> | 83 |
| D.5 | <cpp:memberFunction> | 84 |
| D.6 | <cpp:parameter> | 85 |
| D.7 | JAX-WS WSDL Extensions | 87 |
| D.8 | sca-wsdl-ext-cpp-1.1.xsd | 87 |
| E | XML Schemas | 89 |
| E.1 | sca-interface-cpp-1.1.xsd | 89 |
| E.2 | sca-implementation-cpp-1.1.xsd | 89 |
| E.3 | sca-contribution-cpp-1.1.xsd | 90 |
| F | Normative Statement Summary | 92 |
| F.1 | Annotation Normative Statement Summary | 95 |
| F.2 | WSDL Extention Normative Statement Summary | 96 |
| F.3 | JAX-WS Normative Statements | 96 |
| F.3.1 | Ignored Normative Statements | 99 |
| G | Migration | 101 |
| G.1 | Method child elements of interface.cpp and implementation.cpp | 101 |
| H | Acknowledgements | 102 |
| I | Revision History | 103 |

1 Introduction

2 This document describes the SCA Client and Implementation Model for the C++ programming language.
3 The SCA C++ implementation model describes how to implement SCA components in C++. A component
4 implementation itself can also be a client to other services provided by other components or external
5 services. The document describes how a C++ implemented component gets access to services and calls
6 their operations.
7 The document also explains how non-SCA C++ components can be clients to services provided by other
8 components or external services. The document shows how those non-SCA C++ component
9 implementations access services and call their operations.

10 1.1 Terminology

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

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

16

| 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/200912" | Defined by the SCA specifications |
| cpp | "http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901" | |

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

18 1.2 Normative References

- 19 [[RFC2119](#)] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF
20 RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>
- 21 [[ASSEMBLY](#)] OASIS Committee Draft 05, *Service Component Architecture Assembly Model
22 Specification Version 1.1*, January 2010. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd05.pdf>
- 23 [[POLICY](#)] OASIS Committee Draft 02, *SCA Policy Framework Version 1.1*, March 2009.
24 <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf>
- 25 [[SDO21](#)] OSOA, *Service Data Objects For C++ Specification*, December 2006.
26 <http://www.osoa.org/download/attachments/36/CPP-SDO-Spec-v2.1.0-FINAL.pdf>
- 27 [[WSDL11](#)] World Wide Web Consortium, *Web Service Description Language (WSDL)*,
28 March 2001. <http://www.w3.org/TR/wsdl>
- 29 [[XSD](#)] World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*,
30 October 2004. <http://www.w3.org/TR/xmlschema-2/>
- 31 [[JAXWS21](#)] Doug. Kohlert and Arun Gupta, *The Java API for XML-Based Web Services
32 (JAX-WS) 2.1*, JSR, JCP, May 2007.
33 <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>

35 **1.3 Conventions**

36 **1.3.1 Naming Conventions**

37 This specification follows naming conventions for artifacts defined by the specification:

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

48 **1.3.2 Typographic Conventions**

49 This specification follows typographic conventions for specific constructs:

- 50 • Conformance points are highlighted, [numbered] and cross-referenced to Normative Statement
51 Summary
- 52 • XML attributes are identified in text as @attribute
- 53 • Language identifiers used in text are in courier
- 54 • Literals in text are in *italics*

55 2 Basic Component Implementation Model

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

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

61 2.1 Implementing a Service

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

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

- 66 • a C++ abstract base class
- 67 • a WSDL 1.1 portType **[WSDL11]**

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

70 Snippet 2-1 – Snippet 2-3 show a C++ service interface and the C++ implementation class of a C++
71 implementation.

```
72
73     // LoanService interface
74     class LoanService {
75     public:
76         virtual bool approveLoan(unsigned long customerNumber,
77                               unsigned long loanAmount) = 0;
78     };
```

79 *Snippet 2-1: A C++ Service Interface*

```
80
81     class LoanServiceImpl : public LoanService {
82     public:
83         LoanServiceImpl();
84         virtual ~LoanServiceImpl();
85
86         virtual bool approveLoan(unsigned long customerNumber,
87                               unsigned long loanAmount);
88     };
```

89 *Snippet 2-2: C++ Service Implementation Declaration*

```
90
91     #include "LoanServiceImpl.h"
92
93     LoanServiceImpl::LoanServiceImpl()
94     {
95         ...
96     }
97
98     LoanServiceImpl::~LoanServiceImpl()
99     {
100        ...
101    }
102
103    bool LoanServiceImpl::approveLoan(unsigned long customerNumber,
```

```

104             {
105                 ...
106             }
107         }
```

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

109

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

111

```

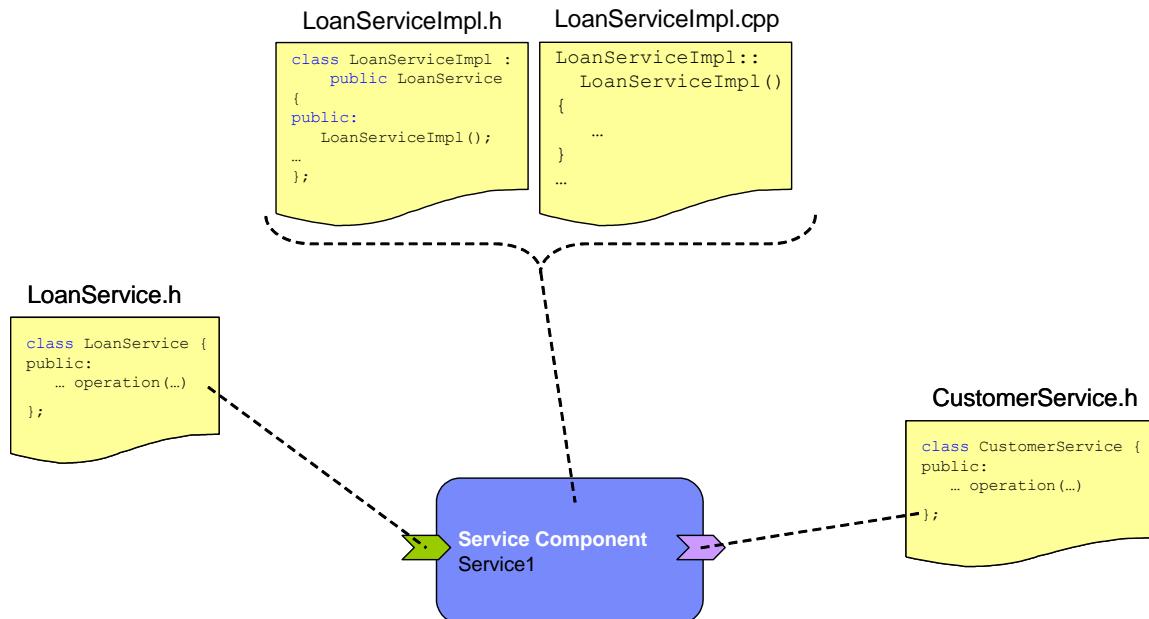
112 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
113     <service name="LoanService">
114         <interface.cpp header="LoanService.h"/>
115     </service>
116 </componentType>
```

117 *Snippet 2-4: Component Type for Service Implementation in Snippet 2-3*

118

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

121



122

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

2.1.1 Implementing a Remotable Service

125 A @remotable="true" attribute on an *interface.cpp* element indicates that the interface is **remotable** as
126 described in the Assembly Specification [ASSEMBLY]. Snippet 2-5 shows the component type for a
127 remotable service:

128

```

129 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
130     <service name="LoanService">
131         <interface.cpp header="LoanService.h" remotable="true"/>
132     </service>
133 </componentType>
```

134 *Snippet 2-5: ComponentType for a Remotable Service*

135 **2.1.2 AllowsPassByReference**

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

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

154 **2.1.2.1 Marking services and references as “allows pass by reference”**

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

- 157 • Member function execution will not modify any input parameter before the member function returns.
- 158 • The service implementation will not retain a reference or pointer to any by-reference input parameter,
159 return value or exception after the member function returns.
- 160 • The member function will observe “allows pass by value” client semantics, as defined in the next
161 paragraph for any callbacks that it makes.

162 Marking a client as “allows pass by reference” asserts that for all reference’s member functions, the client
163 observes the restrictions:

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

169 **2.1.2.2 Using “allows pass by reference” to optimize remotable calls**

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

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

177 **2.1.3 Implementing a Local Service**

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

179 2.2 Component Implementation Scopes

180 SCA defines the concept of implementation scope, which specifies the lifecycle contract an
181 implementation has with the runtime. Invocations on a service offered by a component will be dispatched
182 by the SCA runtime to an implementation instance according to the semantics of its scope.

183 Scopes are specified using the @scope attribute of the *implementation.cpp* element.

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

186 An SCA runtime **MUST** support these scopes; **stateless** and **composite**. Additional scopes **MAY** be
187 provided by SCA runtimes. An SCA runtime **MUST** support these scopes; **stateless** and **composite**.
188 Additional scopes **MAY** be provided by SCA runtimes. [CPP2003]

189 Snippet 2-6 shows the component type for a composite scoped component:

```
190
191 <component name="LoanService">
192   <implementation.cpp library="loan" class="LoanServiceImpl"
193     scope="composite"/>
194 </component>
```

195 Snippet 2-6: Component Type for a Composite Scoped Component

196

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

203 2.2.1 Stateless Scope

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

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

210 2.2.2 Composite Scope

211 All service requests are dispatched to the same implementation instance for the lifetime of the composite
212 containing the component. The lifetime of the containing composite is defined as the time it placed in the
213 running state to the time it is removed from the running state, either normally or abnormally.

214 A composite scoped implementation can also specify eager initialization using the @eagerInit="true"
215 attribute on the *implementation.cpp* element of a component definition. When marked for eager
216 initialization, the implementation instance will be created when the component is placed in running state,
217 otherwise, initialization is lazy and the instance will be created when the first client request is received.

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

220 2.3 Implementing a Configuration Property

221 Component implementations can be configured through properties. The properties and their types (not
222 their values) are defined in the component type file. The C++ component can retrieve the properties using
223 the *getProperties()* on the *ComponentContext* class.

224 Snippet 2-7 shows how to get the property values.

```

225
226 #include "ComponentContext.h"
227 using namespace oasis::sca;
228
229 void clientFunction()
230 {
231     ...
232
233     ComponentContextPtr context = ComponentContext::getCurrent();
234
235     DataObjectPtr properties = context->getProperties();
236
237     long loanRating = properties->getInteger("maxLoanValue");
238
239     ...
240 }
```

241 *Snippet 2-7: Retrieving Property Values*

242 **2.4 Component Type and Component**

243 For a C++ component implementation, a component type is specified in a side file. By default, the
 244 componentType side file is in the root directory of the composite containing the component or some
 245 subdirectory of the composite root directory with a name matching the implementation class of the
 246 component. The location can be modified as described in Implementation.cpp.

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

249 Snippet 2-8 – Snippet 2-10 show a C++ service interface and the C++ implementation class of a C++
 250 service.

```

251
252 // LoanService interface
253 class LoanService {
254 public:
255     virtual bool approveLoan(unsigned long customerNumber,
256                             unsigned long loanAmount) = 0;
257 };
```

258 *Snippet 2-8: A C++ Service Interface*

```

259
260 class LoanServiceImpl : public LoanService {
261 public:
262     LoanServiceImpl();
263     virtual ~LoanServiceImpl();
264
265     virtual bool approveLoan(unsigned long customerNumber,
266                             unsigned long loanAmount);
267 };
```

268 *Snippet 2-9: C++ Service Implementation Declaration*

```

269
270 #include "LoanServiceImpl.h"
271
272 // Construction/Destruction
273
274 LoanServiceImpl::LoanServiceImpl()
275 {
276     ...
277 }
278 LoanServiceImpl::~LoanServiceImpl()
279 {
```

```
280     ...
281 }
282 // Implementation
283
284 bool LoanServiceImpl::approveLoan(unsigned long customerNumber,
285                                     unsigned long loanAmount)
286 {
287     ...
288 }
```

289 *Snippet 2-10: C++ Service Implementation*

290

291 Snippet 2-11 shows the component type for this component implementation.

292

```
293 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
294     <service name="LoanService">
295         <interface.cpp header="LoanService.h"/>
296     </service>
297 </componentType>
```

298 *Snippet 2-11: Component Type for Service Implementation in Snippet 2-10*

299

300 Snippet 2-12 shows a component using the implementation.

301

```
302 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
303     name="LoanComposite" >
304     ...
305
306     <component name="LoanService">
307         <implementation.cpp library="loan" class="LoanServiceImpl"/>
308     </component>
309 </composite>
```

310 *Snippet 2-12: Component Using Implementation in Snippet 2-10*

311 **2.4.1 Interface.cpp**

312 Snippet 2-13 shows the pseudo-schema for the C++ interface element used to type services and
313 references of component types.

314

```
315 <!-- interface.cpp schema snippet -->
316 <interface.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
317     header="string" class="Name"? remotable="boolean"?
318     callbackHeader="string" callbackClass="Name"?
319     requires="listOfQNames"? policySets="listOfQNames"? >
320
321     <function ... />*
322     <callbackFunction ... />*
323     <requires/>*
324     <policySetAttachment/>*
325
326 </interface.cpp>
```

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

328

329 The **interface.cpp** element has the **attributes**:

- 380 • **header : string (1..1)** – full name of the header file that describes the interface, including relative path
 381 from the composite root.
- 382 • **class : Name (0..1)** – name of the class declaration for the interface in the header file, including any
 383 namespace definition. If the header file identified by the @header attribute of an *<interface.cpp/>*
 384 element contains more than one class, then the @class attribute MUST be specified for the
 385 *<interface.cpp/>* element. If the header file identified by the @header attribute of an *<interface.cpp/>*
 386 element contains more than one class, then the @class attribute MUST be specified for the
 387 *<interface.cpp/>* element. [CPP20005]
- 388 • **callbackHeader : string (0..1)** – full name of the header file that describes the callback interface,
 389 including relative path from the composite root.
- 390 • **callbackClass : Name (0..1)** – name of the class declaration for the callback interface in the callback
 391 header file, including any namespace definition. If the header file identified by the @callbackHeader
 392 attribute of an *<interface.cpp/>* element contains more than one class, then the @callbackClass
 393 attribute MUST be specified for the *<interface.cpp/>* element. If the header file identified by the
 394 @callbackHeader attribute of an *<interface.cpp/>* element contains more than one class, then the
 395 @callbackClass attribute MUST be specified for the *<interface.cpp/>* element. [CPP20006]
- 396 • **remotable : boolean (0..1)** – indicates whether the service is remotable or local. The default is local.
 397 See Implementing a Remotable Service
- 398 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification
 399 [POLICY] for a description of this attribute. If intents are specified at both the class and member
 400 function level, the effective intents for the member function is determined by merging the combined
 401 intents from the member function with the combined intents for the class according to the Policy
 402 Framework rules for merging intents within a structural hierarchy, with the member function at the
 403 lower level and the class at the higher level.
- 404 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification
 405 [POLICY] for a description of this attribute.

406
 407 The *interface.cpp* element has the **child elements**:

- 408 • **function : CPPFunction (0..n)** – see Function and CallbackFunction
- 409 • **callbackFunction : CPPFunction (0..n)** – see Function and CallbackFunction
- 410 • **requires : requires (0..n)** - See the Policy Framework specification [POLICY] for a description of this
 411 element.
- 412 • **policySetAttachment : policySetAttachment (0..n)** - See the Policy Framework specification
 413 [POLICY] for a description of this element.

414 2.4.2 Function and CallbackFunction

415 A member function of an interface might have behavioral characteristics that need to be identified. This is
 416 done using a *function* or *callbackFunction* child element of *interface.cpp*. These child elements are also
 417 used when not all functions in a class are part of the interface.

- 418 • If the header file identified by the @header attribute of an *<interface.cpp/>* element contains function
 419 declarations that are not operations of the interface, then the functions that are not operations of the
 420 interface MUST be excluded using *<function/>* child elements of the *<interface.cpp/>* element with
 421 @exclude="true". If the header file identified by the @header attribute of an *<interface.cpp/>* element
 422 contains function declarations that are not operations of the interface, then the functions that are not
 423 operations of the interface MUST be excluded using *<function/>* child elements of the
 424 *<interface.cpp/>* element with @exclude="true". [CPP20016]
- 425 • If the header file identified by the @callbackHeader attribute of an *<interface.cpp/>* element contains
 426 function declarations that are not operations of the callback interface, then the functions that are not
 427 operations of the callback interface MUST be excluded using *<callbackFunction/>* child elements of
 428 the *<interface.cpp/>* element with @exclude="true". If the header file identified by the
 429 @callbackHeader attribute of an *<interface.cpp/>* element contains function declarations that are not

Formatter

Formatter

430 operations of the callback interface, then the functions that are not operations of the callback interface
431 MUST be excluded using <callbackFunction> child elements of the <interface.cpp> element with
432 @exclude="true". [CPP20017]

433 Snippet 2-14 shows the *interface.cpp* pseudo-schema with the pseudo-schema for the *function* and
434 *callbackFunction* child elements:

435

```
436 <!-- Function schema snippet -->
437 <interface.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
438
439     <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
440         oneWay="Boolean"? exclude="Boolean"? >
441         <requires/>*
442         <policySetAttachment/>*
443     </function> *
444
445     <callbackFunction name="NCName" requires="listOfQNames"?
446         policySets="listOfQNames"? oneWay="Boolean"? exclude="Boolean"? >
447         <requires/>*
448         <policySetAttachment/>*
449     </callbackFunction> *
450
451 </interface.cpp>
```

452 Snippet 2-14: Pseudo-schema for Interface Function and CallbackFunction Sub-elements

453

454 The **function** and **callbackFunction** elements have the **attributes**:

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

455 The @name attribute of a <callbackFunction> child element of a <interface.cpp> MUST be unique
456 amongst the <callbackFunction> elements of that <interface.cpp>. The @name attribute of a
457 <callbackFunction> child element of a <interface.cpp> MUST be unique amongst the
458 <callbackFunction> elements of that <interface.cpp>. [CPP20008]

- **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute.
- **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.
- **oneWay : boolean (0..1)** – see Non-blocking Calls
- **exclude : boolean (0..1)** – if true, the member function is excluded from the interface. The default is false.

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

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

475 2.4.3 Implementation.cpp

476 Snippet 2-15 shows the pseudo-schema for the C++ implementation element used to define the
477 implementation of a component.

478

```

479 <!-- implementation.cpp schema snippet -->
480 <implementation.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
481   library="NCName" path="string"? class="Name"
482   scope="scope"? componentType="string"? allowsPassByReference="Boolean"?
483   eagerInit="boolean"? requires="listOfQNames"?
484   policySets="listOfQNames"? >
485
486   <function ... />*
487   <requires/>*
488   <policySetAttachment/>*
489
490 </implementation.cpp>
```

491 Snippet 2-15: Pseudo-schema for C++ Implementation Element

492

493 The ***implementation.cpp*** element has the ***attributes***:

- 494 • ***library : NCName (1..1)*** – name of the dll or shared library that holds the factory for the service component. This is the root name of the library.
- 495
- 496 • ***path : string (0..1)*** - path to the library which is either relative to the root of the contribution containing the composite or is prefixed with a contribution import name and is relative to the root of the import. See C++ Contributions.
- 497
- 498
- 499 • ***class : Name (1..1)*** – name of the class declaration of the implementation, including any namespace definition. 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. 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. [CPP20009] The SCA runtime will append .componentType to the class name to find the componentType file.
- 500
- 501
- 502
- 503
- 504
- 505
- 506 • ***scope : CPPImplementationScope (0..1)*** – identifies the scope of the component implementation. The default is stateless. See Component Implementation Scopes
- 507
- 508 • ***componentType : string (0..1)*** – path to the componentType file which is relative to the root of the contribution containing the composite or is prefixed with a contribution import name and is relative to the root of the import.
- 509
- 510
- 511 • ***allowsPassByReference : boolean (0..1)*** – indicates the implementation allows pass by reference data exchange semantics on calls to it or from it. These semantics apply to all services provided by and references used by an implementation. See AllowsPassByReference
- 512
- 513
- 514 • ***eagerInit : boolean (0..1)*** – indicates a composite scoped implementation is to be initialized when it is loaded. See Composite ScopeComposite Scope
- 515
- 516 • ***requires : listOfQNames (0..1)*** – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute. If intents are specified at both the class and member function level, the effective intents for the member function is determined by merging the combined intents from the member function with the combined intents for the class according to the Policy Framework rules for merging intents within a structural hierarchy, with the member function at the lower level and the class at the higher level.
- 517
- 518
- 519
- 520
- 521
- 522 • ***policySets : listOfQNames (0..1)*** – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.
- 523
- 524 The ***implementation.cpp*** element has the ***child elements***:
- 525 • ***function : CPPImplementationMethod (0..n)*** – see Implementation Function
- 526 • ***requires : requires (0..n)*** - See the Policy Framework specification [POLICY] for a description of this element.
- 527
- 528 • ***policySetAttachment : policySetAttachment (0..n)*** - See the Policy Framework specification [POLICY] for a description of this element.
- 529

530 **2.4.4 Implementation Function**

531 A member function of an implementation might have operational characteristics that need to be identified.
532 This is done using a *function* child element of *implementation.cpp*

533 Snippet 2-16 shows the *implementation.cpp* schema with the schema for a *method* child element:

534

```
535 <!-- ImplementationFunction schema snippet -->
536 <implementation.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ...
537 >
538
539     <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
540         allowsPassByReference="boolean"? >
541         <requires/>*
542         <policySetAttachment/>*
543     </function> *
544
545 </implementation.cpp>
```

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

547

548 The **function** element has the **attributes**:

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

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

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

556 **2.5 Instantiation**

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

568 3 Basic Client Model

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

571 3.1 Accessing Services from Component Implementations

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

573 Snippet 3-1 shows the `ComponentContext` C++ class with its `getService()` member function.

574

```
575 namespace oasis {
576     namespace sca {
577
578         class ComponentContext {
579             public:
580                 static ComponentContextPtr getCurrent();
581                 virtual ServiceProxyPtr getService(
582                     const std::string& referenceName) const = 0;
583
584             ...
585         }
586     }
}
```

587 *Snippet 3-1: Partial ComponetContext Class Definition*

588

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

593 Snippet 3-2 a sample of how the `ComponentContext` is used in a C++ component implementation. The
594 `getService()` member function is called on the `ComponentContext` passing the reference name as
595 input. The return of the `getService()` member function is cast to the abstract base class defined for the
596 reference.

597

```
598 #include "ComponentContext.h"
599 #include "CustomerServiceProxy.h"
600
601 using namespace oasis::sca;
602
603 void clientFunction()
{
604
605     unsigned long customerNumber = 1234;
606
607     ComponentContextPtr context = ComponentContext::getCurrent();
608
609     ServiceProxyPtr service = context->getService("customerService");
610     CustomerServiceProxyPtr customerService =
611         dynamicCast<CustomerServiceProxy>(service);
612
613     if (customerService)
614         short rating = customerService->getCreditRating(customerNumber);
615
616 }
617 }
```

618 Snippet 3-2: Using ComponentContext

619 3.2 Interface Proxies

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

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

625

```
626     class ServiceProxy {  
627         public:  
628             //Possible future extensions  
629     };
```

630 Snippet 3-3: ServiceProxy Class Definition

631

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

637 For the interface definition:

638

```
639 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
640     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
641     xmlns:tns="http://www.example.org/"  
642     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"  
643     targetNamespace="http://www.example.org/">  
644  
645     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
646         xmlns:tns="http://www.example.org/"  
647         attributeFormDefault="unqualified"  
648         elementFormDefault="unqualified"  
649         targetNamespace="http://www.example.org/">  
650         <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
651         <xsd:element name="GetLastTradePriceResponse"  
652             type="tns:GetLastTradePriceResponse"/>  
653         <xsd:complexType name="GetLastTradePrice">  
654             <xsd:sequence>  
655                 <xsd:element name="tickerSymbol" type="xsd:string"/>  
656             </xsd:sequence>  
657         </xsd:complexType>  
658         <xsd:complexType name="GetLastTradePriceResponse">  
659             <xsd:sequence>  
660                 <xsd:element name="return" type="xsd:float"/>  
661             </xsd:sequence>  
662         </xsd:complexType>  
663     </xsd:schema>  
664  
665     <message name="GetLastTradePrice">  
666         <part name="parameters" element="tns:GetLastTradePrice">  
667             </part>  
668     </message>  
669  
670     <message name="GetLastTradePriceResponse">  
671         <part name="parameters" element="tns:GetLastTradePriceResponse">  
672             </part>
```

```

673     </message>
674
675     <portType name="StockQuote">
676         <cpp:bindings>
677             <cpp:class name="StockQuoteService"/>
678         </cpp:bindings>
679         <operation name="GetLastTradePrice">
680             <cpp:bindings>
681                 <cpp:memberFunction name="getTradePrice"/>
682             </cpp:bindings>
683             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
684             </input>
685             <output name="GetLastTradePriceResponse"
686                 message="tns:GetLastTradePriceResponse">
687             </output>
688         </operation>
689     </portType>
690 </definitions>

```

691 *Snippet 3-4: Sample WSDL Interface*

692

693 The resulting abstract proxy class is:

694

```

695     // @WebService(name="StockQuote", targetNamespace="http://www.example.org/")
696     //       serviceName="StockQuoteService")
697     class StockQuoteServiceProxy: public ServiceProxy {
698
699         // @WebFunction(operationName="GetLastTradePrice",
700         //           action="urn:GetLastTradePrice")
701         float getTradePrice(const std::string& tickerSymbol);
702     };

```

703 *Snippet 3-5: Proxy Class for Interface in Snippet 3-4*

704

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

707

```

708     // LoanService interface
709     class LoanService {
710     public:
711         virtual bool approveLoan(unsigned long customerNumber,
712                               unsigned long loanAmount) = 0;
713     };

```

714 *Snippet 3-6: Sample C++ Interface*

715

716 The resulting proxy class is:

717

```

718     class LoanServiceProxy : public ServiceProxy {
719     public:
720         virtual bool approveLoan(unsigned long customerNumber,
721                               unsigned long loanAmount) = 0;
722     };

```

723 *Snippet 3-7: Proxy Class for Interface in Snippet 3-6*

724

725 | [For each reference of a component, an SCA implementation MUST generate a service proxy derived](#)
726 | [from `ServiceProxy` that contains the operations of the reference's interface definition.](#) [For each](#)

727 | reference of a component, an SCA implementation MUST generate a service proxy derived from
728 | ~~ServiceProxy~~ that contains the operations of the reference's interface definition. [CPP30002]

729 | 3.3 Accessing Services from non-SCA Component Implementations

730 | Non-SCA components can access component services by obtaining a `DomainContextPtr` from the
731 | SCA runtime and then following the same steps as a component implementation as described above.
732 | Snippet 3-8 shows a sample of how the `DomainContext` is used in non-SCA C++ code.

```
733 |
734 | #include "DomainContext.h"
735 | #include "CustomerServiceProxy.h"
736 |
737 | void externalFunction()
738 | {
739 |
740 |     unsigned long customerNumber = 1234;
741 |
742 |     DomainContextPtr context = myImplGetDomain("http://example.com/mydomain");
743 |
744 |     ServiceProxyPtr service = context->getService("customerService");
745 |     CustomerServiceProxyPtr customerService =
746 |         dynamicCast<CustomerServiceProxy>(service);
747 |
748 |     if (customerService)
749 |         short rating = customerService->getCreditRating(customerNumber);
750 |
751 }
```

752 | *Snippet 3-8: Using DomianContext*

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

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

- 756 | • componentName/serviceName/bindingName

757 | The function `myImplGetDomain()` in Snippet 3-8 is an example of how a non-SCA client might get a
758 | `DomainContextPtr` and is not a function defined by this specification. The specific mechanism for how
759 | an SCA runtime implementation returns a `DomainContextPtr` is not defined by this specification.

761 | 3.4 Calling Service Operations

762 | Accessing Services from Component Implementations and [Accessing Services from non-SCA Component Implementations](#) show how
763 | to get access to a service. Once you have access to the service, calling an operation of the service is like
764 | calling a member function of a C++ class via a `ServicelyProxy`.

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

769 | 3.5 Long Running Request-Response Operations

770 | The Assembly Specification **[ASSEMBLY]** allows service interfaces or individual operations to be marked
771 | **long-running** using an `@requires="asyncInvocation"` intent, with the meaning that the operation(s) might
772 | not complete in any specified time interval, even when the operations are request-response operations.
773 | A client calling such an operation has to be prepared for any arbitrary delay between the time a request is
774 | made and the time the response is received. To support this kind of operation three invocation styles are
775 | available: asynchronous – the client provides a response handler, polling – the client will poll the SCA

776 runtime to determine if a response is available, and synchronous – the SCA runtime handles suspension
777 of the main thread, asynchronously receiving the response and resuming the main thread. The details of
778 each of these styles are provided in the following sections.

779 For a service operation with signature

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

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

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

784 *Snippet 3-9: AsynchronousInvocation Member Function Format*

785

786 where <response_message_name>Response is the response class for the operation as defined by
787 Response Class. The client uses this member function to issue a request through the SCA runtime. The
788 response is returned immediately, and can be used to access the response when it becomes available.

789

790 **An SCA runtime MUST include an asynchronous invocation member function for every operation of a**
791 **reference interface with a @requires="asyncInvocation" intent applied either to the operation or the**
792 **reference as a whole. An SCA runtime MUST include an asynchronous invocation member function for**
793 **every operation of a reference interface with a @requires="asyncInvocation" intent applied either to the**
794 **operation or the reference as a whole. [CPP30003]**

795 Snippet 3-10 shows a sample proxy class interface providing an asynchronous API.

796

```
797 using namespace oasis::sca;
798 using namespace commonj::sdo;
799
800
801 class CustomerServiceProxy : public ServiceProxy {
802 public:
803
804     // synchronous member function
805     virtual short getCreditRating(unsigned long customerNumber) = 0;
806
807     // forward declare callback class
808     class getCreditRatingCallback;
809
810     // asynchronous response object
811     class getCreditRatingResponse {
812     public:
813         // IOU/Future member functions
814         virtual void cancel() = 0;
815         virtual bool isCancelled() = 0;
816         virtual bool isReady() = 0;
817         virtual void setCallback(getCreditRatingCallbackPtr callback) = 0;
818
819         virtual short getReturn() = 0;
820     };
821
822     // asynchronous callback object
823     class getCreditRatingCallback {
824     public:
825         virtual void invoke(getCreditRatingResponsePtr) = 0;
826     };
827
828     // asynchronous member function
829     virtual getCreditRatingResponsePtr getCreditRatingAsync(unsigned long
830             customerNumber) = 0;
831 }
```

832 Snippet 3-10: Proxy with an Asynchronous API

833

834 Snippet 3-11 shows a sample of how the asynchronous invocation style is used in a C++ component
835 implementation.

836

```
#include "ComponentContext.h"
#include "CustomerServiceProxy.h"

using namespace oasis::sca;

void clientFunction()
{
    ComponentContextPtr context = ComponentContext::getCurrent();

    ServiceProxyPtr service = context->getService("customerService");
    CustomerServiceProxyPtr customerService =
        dynamicCast<CustomerServiceProxy>(service);

    if (customerService) {
        getCreditRatingResponsePtr response =
            customerService->getCreditRatingAsync(1234);

        // ...
    }
}
```

858 Snippet 3-11: Using an Asynchronous API

859

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

863 3.5.1 Response Callback

864 If a callback is specified on a response object, that callback will be invoked when the response is received
865 by the runtime. Snippet 3-12 demonstrates creating a response object and setting it on a response
866 instance:

867

```
#include "ComponentContext.h"
#include "CustomerServiceProxy.h"

using namespace oasis::sca;

class CreditRatingCallback :
    public CustomerServiceProxy::getCreditRatingCallback {

    virtual void invoke(getCreditRatingResponsePtr response) {
        try {
            short rating = response->getReturn();
        }
        catch (...) {
            // ...
        }
    }
};

void clientFunction()
{
```

```

888     ComponentContextPtr context = ComponentContext::getCurrent();
889
890     ServiceProxyPtr service = context->getService("customerService");
891     CustomerServiceProxyPtr customerService =
892         dynamicCast<CustomerServiceProxy>(service);
893
894     if (customerService) {
895         CustomerServiceProxy::getCreditRatingResponsePtr response =
896             customerService->getCreditRatingAsync(1234);
897
898         CustomerServiceProxy::getCreditRatingCallbackPtr callback =
899             new CreditRatingCallback();
900
901         response->setCallback(callback);
902
903         // ...
904     }
905 }
```

906 *Snippet 3-12: Using a Response Object*

907 3.5.2 Response Polling

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

```

909
910 #include "ComponentContext.h"
911 #include "CustomerServiceProxy.h"
912
913 using namespace oasis::sca;
914
915 void clientFunction()
916 {
917     ComponentContextPtr context = ComponentContext::getCurrent();
918
919     ServiceProxyPtr service = context->getService("customerService");
920     CustomerServiceProxyPtr customerService =
921         dynamicCast<CustomerServiceProxy>(service);
922
923     if (customerService) {
924         CustomerServiceProxy::getCreditRatingResponsePtr response =
925             customerService->getCreditRatingAsync(1234);
926
927         while (!response->isReady()) {
928             // do something else
929         }
930
931         // The response is ready and can be accessed without blocking.
932         try {
933             short rating = response->getReturn();
934         }
935         catch (...) {
936             // ...
937         }
938     }
939 }
```

940 *Snippet 3-13: Polling a Response Object*

941 3.5.3 Synchronous Response Access

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

```

945
946 #include "ComponentContext.h"
947 #include "CustomerServiceProxy.h"
948
949 using namespace oasis::sca;
950
951 void clientFunction()
952 {
953     ComponentContextPtr context = ComponentContext::getCurrent();
954
955     ServiceProxyPtr service = context->getService("customerService");
956     CustomerServiceProxyPtr customerService =
957         dynamicCast<CustomerServiceProxy>(service);
958
959     if (customerService) {
960         CustomerServiceProxy::getCreditRatingResponsePtr response =
961             customerService->getCreditRatingAsync(1234);
962
963         // The response is ready and can be accessed without blocking.
964         try {
965             short rating = response->getReturn();
966         }
967         catch (...) {
968             // ...
969         }
970     }
971 }
```

972 *Snippet 3-14: Blocking on a Response Object*

973 3.5.4 Response Class

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

977

```

978 class <response_message_name>Response {
979 public:
980     virtual <response_message_type> getReturn() const = 0;
981     virtual void setCallback(<response_message_name>CallbackPtr callback) = 0;
982     virtual boolean isReady() const = 0;
983     virtual void cancel() const = 0;
984     virtual boolean isCancelled() const = 0;
985 };
```

986 *Snippet 3-15: AsynchronousInvocation Response Class Definition*

987

988 An SCA runtime MUST include a response class for every response message of a reference interface
989 that can be returned by an operation of the interface with a @requires="asyncInvocation" intent applied
990 either to the operation of the reference as a whole. An SCA runtime MUST include a response class for
991 every response message of a reference interface that can be returned by an operation of the interface
992 with a @requires="asyncInvocation" intent applied either to the operation of the reference as a whole.
993 [CPP30004]

994 3.5.4.1 getReturn

995 A C++ component implementation uses `getReturn()` to retrieve the response data for an asynchronous
996 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. | |

997 *Table 3-1: AsynchronousInvocation Response::getReturn Details*

998 **3.5.4.2 setCallback**

999 A C++ component implementation uses `setCallback()` to set a callback object for an asynchronous
1000 invocation.

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

1001 *Table 3-2: AsynchronousInvocation Response::setCallback Details*

1002 **3.5.4.3 isReady**

1003 A C++ component implementation uses `isReady()` to determine if the response data for an polling
1004 invocation is available.

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

1005 *Table 3-3: AsynchronousInvocation Response::isReady Details*

1006 **3.5.4.4 cancel**

1007 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. | |

1008 *Table 3-4: AsynchronousInvocation Response::cancel Details*

1009 **3.5.4.5 isCancelled**

1010 A C++ component implementation uses `isCancelled()` to determine if another thread has cancelled an
1011 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 |

1012 *Table 3-5: AsynchronousInvocation Response::isCancelled Details*

1013 4 Asynchronous Programming

1014 Asynchronous programming of a service is where a client invokes a service and carries on executing
1015 without waiting for the service to execute. Typically, the invoked service executes at some later time.
1016 Output from the invoked service, if any, is fed back to the client through a separate mechanism, since no
1017 output is available at the point where the service is invoked. This is in contrast to the call-and-return style
1018 of synchronous programming, where the invoked service executes and returns any output to the client
1019 before the client continues. The SCA asynchronous programming model consists of support for non-
1020 blocking operation calls and callbacks.

1021 4.1 Non-blocking Calls

1022 Non-blocking calls represent the simplest form of asynchronous programming, where the client of the
1023 service invokes the service and continues processing immediately, without waiting for the service to
1024 execute.

1025 Any member function that returns `void`, has only by-value parameters and has no declared exceptions
1026 can be marked with the `@oneWay="true"` attribute in the interface definition of the service. An operation
1027 marked as `oneWay` is considered non-blocking and the SCA runtime MAY use a binding that buffers the
1028 requests to the member function and sends them at some time after they are made. [CPP40001]

1029 Snippet 4-1 shows the component type for a service with the `reportEvent()` member function declared
1030 as a one-way operation:

```
1031
1032 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
1033   <service name="LoanService">
1034     <interface.cpp header="LoanService.h">
1035       <function name="reportEvent" oneWay="true" />
1036     </interface.cpp>
1037   </service>
1038 </componentType>
```

1039 Snippet 4-1: ComponentType with oneWay Member Function

1040

1041 SCA does not currently define a mechanism for making non-blocking calls to methods that return values
1042 or are declared to throw exceptions. It is considered to be a best practice that service designers define
1043 one-way member function as often as possible, in order to give the greatest degree of binding flexibility to
1044 deployers.

1045 4.2 Callbacks

1046 Callback services are used by *bidirectional services* as defined in the Assembly Specification
1047 [ASSEMBLY].

1048 A callback interface is declared by the `@callbackHeader` and `@callbackClass` attributes in the interface
1049 definition of the service. Snippet 4-2 shows the component type for a service `MyService` with the interface
1050 defined in `MyService.h` and the interface for callbacks defined in `MyServiceCallback.h`,

```
1051
1052 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" >
1053   <service name="MyService">
1054     <interface.cpp header="MyService.h"
1055       callbackHeader="MyServiceCallback.h"/>
1056   </service>
1057 </componentType>
```

1058 Snippet 4-2: ComponentType with a Callback Interface

4.2.1 Using Callbacks

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

Snippet 4-3– Snippet 4-5 show a scenario in which bidirectional interfaces and callbacks could be used. A client requests a quotation from a supplier. To process the enquiry and return the quotation, some suppliers might need additional information from the client. The client does not know which additional items of information will be needed by different suppliers. This interaction can be modeled as a bidirectional interface with callback requests to obtain the additional information.

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

1085
1086
1087
1088
1089
1090 Snippet 4-4 illustrates a possible implementation of the example service.

```
1091  
1092     #include "QuotationImpl.h"  
1093     #include "QuotationCallbackProxy.h"  
1094     #include "ComponentContext.h"  
1095     using namespace oasis::sca;  
1096  
1097     double QuotationImpl::requestQuotation(std::string productCode,  
1098                                         unsigned int quantity) {  
1099         double price = getPrice(productQuote, quantity);  
1100         double discount = 0;  
1101  
1102         ComponentContextPtr context = ComponentContext::getCurrent();  
1103         ServiceReferencePtr serviceRef = context->getSelfReference();  
1104         ServiceProxyPtr callback = serviceRef->getCallback();  
1105         QuotationCallbackQuotationCallbackProxyPtr quotationCallback =  
1106             dynamicCast<QuotationCallbackProxy>(callback);  
1107  
1108         if (quotationCallback) {  
1109             if (quantity > 1000 && callback->getState().compare("FL") == 0)  
1110                 discount = 0.05;  
1111             if (quantity > 10000 && callback->getCreditRating().data() == 'A')  
1112                 discount += 0.05;  
1113         }  
1114         return price * (1-discount);  
1115     }
```

1116 Snippet 4-4: Implementation of Forward Service with Interface in Snippet 4-3

1117

1118 Snippet 4-5 is taken from the client of this example service. The client's service implementation class
1119 implements the member functions of the QuotationCallback interface as well as those of its own service
1120 interface ClientService.

1121

```
1122 #include "QuotationCallback.h"
1123 #include "QuotationServiceProxy.h"
1124 #include "ComponentContext.h"
1125 using namespace oasis::sca;
1126
1127 void ClientImpl:: aClientFunction() {
1128     ComponentContextPtr context = ComponentContext::getCurrent();
1129
1130     ServiceProxyPtr service = context->getService("quotationService");
1131     QuotationServiceProxyPtr quotationService =
1132         dynamicCast<QuotationServiceProxy>(service);
1133
1134     if (quotationService)
1135         quotationService->requestQuotation("AB123", 2000);
1136 }
1137
1138 std::string QuotationCallbackImpl::getState() {
1139     return "TX";
1140 }
1141 std::string QuotationCallbackImpl::getZipCode() {
1142     return "78746";
1143 }
1144 std::string QuotationCallbackImpl::getCreditRating() {
1145     return "AA";
1146 }
```

1147 Snippet 4-5: Implementation of Callback Interface in Snippet 4-3

1148

1149 For each service of a component that includes a bidirectional interface, an SCA implementation MUST
1150 generate a service proxy derived from ServiceProxy that contains the operations of the reference's
1151 callback interface definition. For each service of a component that includes a bidirectional interface, an
1152 SCA implementation MUST generate a service proxy derived from ServiceProxy that contains the
1153 operations of the reference's callback interface definition. [CPP40002]

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. If a service of a component that has a callback
1158 interface contains operations with a @requires="asyncInvocation" intent applied either to the operation of
1159 the reference as a whole, an SCA implementation MUST include asynchronous invocation member
1160 functions and response classes as described in Long Running Request-Response Operations.
1161 [CPP40003]

1162 In the example the callback is **stateless**, i.e., the callback requests do not need any information relating
1163 to the original service request. For a callback that needs information relating to the original service
1164 request (a **stateful** callback), this information can be passed to the client by the service provider as
1165 parameters on the callback request.

1166 4.2.2 Callback Instance Management

1167 Instance management for callback requests received by the client of the bidirectional service is handled in
1168 the same way as instance management for regular service requests. If the client implementation has
1169 stateless scope, the callback is dispatched using a newly initialized instance. If the client implementation

1170 has composite scope, the callback is dispatched using the same shared instance that is used to dispatch
1171 regular service requests.

1172 As described in Using Callbacks, a stateful callback can obtain information relating to the original service
1173 request from parameters on the callback request. Alternatively, a composite-scoped client could store
1174 information relating to the original request as instance data and retrieve it when the callback request is
1175 received. These approaches could be combined by using a key passed on the callback request (e.g., an
1176 order ID) to retrieve information that was stored in a composite-scoped instance by the client code that
1177 made the original request.

1178 **4.2.3 Implementing Multiple Bidirectional Interfaces**

1179 Since it is possible for a single class to implement multiple services, it is also possible for callbacks to be
1180 defined for each of the services that it implements. To access the callbacks the
1181 `ServiceReference::getCallback(serviceName)` member function is used, passing in the name
1182 of the service for which the callback is to be obtained.

5 Error Handling

- 1183 Clients calling service operations will experience business exceptions, and SCA runtime exceptions.
- 1184 Business exceptions are raised by the implementation of the called service operation. It is expected that
- 1185 these will be caught by client invoking the operation on the service.
- 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 SCA runtime
- 1188 exceptions defined are:
- 1189
- 1190 • SCAException – defines a root exception type from which all SCA defined exceptions derive.
 - 1191 – SCANullPointerException – signals that code attempted to dereference a null pointer from a
 - 1192 RefCountingPointer object.
 - 1193 – ServiceRuntimeException - signals problems in the management of the execution of SCA
 - 1194 components.
 - 1195 • ServiceUnavailableException – signals problems in the interaction with remote
 - 1196 services. This extends ServiceRuntimeException. These are exceptions that could be
 - 1197 transient, so retrying is appropriate. Any exception that is a ServiceRuntimeException
 - 1198 that is not a ServiceUnavailableException is unlikely to be resolved by retrying the
 - 1199 operation, since it most likely requires human intervention.
 - 1200 • MultipleServicesException – signals that a member function expecting identification of
 - 1201 a single service is called where there are multiple services defined. Thrown by
 - 1202 ComponentContext::getService(), ComponentContext::getSelfReference()
 - 1203 and ComponentContext::getServiceReference().

1204 6 C++ API

1205 All the C++ interfaces are found in the namespace `oasis::sca`, which has been omitted from the
1206 definitions for clarity.

1207 6.1 Reference Counting Pointers

1208 These are a derived version of the familiar smart-pointer. The pointer class holds a real (dumb) pointer to
1209 the object. If the reference counting pointer is copied, then a duplicate pointer is returned with the same
1210 real pointer. A reference count within the object is incremented for each copy of the pointer, so only when
1211 all pointers go out of scope will the object be freed.

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

1217

```
1218 template <typename T>
1219 class RefCountingPointer {
1220 public:
1221     T& operator* () const;
1222     T* operator-> () const;
1223     operator void* () const;
1224     bool operator! () const;
1225 };
1226
1227 template <typename T, typename U>
1228 RefCountingPointer<T> constCast(RefCountingPointer<U> other);
1229
1230 template <typename T, typename U>
1231 RefCountingPointer<T> dynamicCast(RefCountingPointer<U> other);
1232
1233 template <typename T, typename U>
1234 RefCountingPointer<T> reinterpretCast(RefCountingPointer<U> other);
1235
1236 template <typename T, typename U>
1237 RefCountingPointer<T> staticCast(RefCountingPointer<U> other);
```

1238 *Snippet 6-1: RefCountingPointer Class Definition*

1239 6.1.1 operator*

1240 A C++ component implementation uses the `*` operator to dereferences the underlying pointer of a
1241 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 NULL |
| Post Condition | No change |

1242 *Table 6-1: RefCountingPointer operator* Details*

1243 **6.1.2 operator->**

1244 A C++ component implementation uses the `->` operator to invoke member functions on the underlying
1245 pointer of a reference counting pointer. This is equivalent to invoking `p->func()` where `func()` is a
1246 member function defined on the underlying pointer type.

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

1247 *Table 6-2: RefCountingPointer operator-> Details*

1248 **6.1.3 operator void***

1249 A C++ component implementation uses the `void*` operator to determine if the underlying pointer of a
1250 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 |

1251 *Table 6-3: RefCountingPointer operator void* Details*

1252 **6.1.4 operator!**

1253 A C++ component implementation uses the `!` operator to determine if the underlying pointer of a
1254 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 |

1255 *Table 6-4: RefCountingPointer operator! Details*

1256 **6.1.5 constCast**

1257 The `constCast` global function provides the semantic behavior of the `const_cast` operator for use with
1258 `RefCountingPointers`.

| | |
|----------------|--|
| Precondition | C++ component instance is running and has a reference counting pointer |
| Return | A <code>RefCountingPointer</code> instance templatized on the target type. |
| Post Condition | No change |

1259 *Table 6-5: constCast for RefCountingPointers Details*

1260 **6.1.6 dynamicCast**

1261 The `dynamicCast` global function provides the semantic behavior of the `dynamic_cast` operator for use
1262 with `RefCountingPointers`.

| | |
|--------------|--|
| Precondition | C++ component instance is running and has a reference counting pointer |
|--------------|--|

| | |
|----------------|---|
| Return | A RefCountingPointer instance templatized on the target type. This can return an invalid RefCountingPointer instance if the cast fails. |
| Post Condition | No change |

1263 *Table 6-6: dynamicCast for RefCountingPointers Details*

6.1.7 reinterpretCast

1265 The reinterpretCast global function provides the semantic behavior of the reinterpret_cast operator for
1266 use with RefCountingPointers.

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

1267 *Table 6-7: reinterpretCast for RefCountingPointers Details*

6.1.8 staticCast

1269 The staticCast global function provides the semantic behavior of the static_cast operator for use with
1270 RefCountingPointers.

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

1271 *Table 6-8: staticCast for RefCountingPointers Details*

6.2 Component Context

1273 The complete ComponentContext interface definition is:

1274

```

1275 class ComponentContext {
1276 public:
1277     static ComponentContextPtr getCurrent();
1278
1279     virtual std::string getURI() const = 0;
1280
1281     virtual ServiceProxyPtr getService(
1282             const std::string& referenceName) const = 0;
1283     virtual std::vector<ServiceProxyPtr> getServices(
1284             const std::string& referenceName) const = 0;
1285
1286     virtual ServiceReferencePtr getServiceReference(
1287             const std::string& referenceName) const = 0;
1288     virtual std::vector<ServiceReferencePtr> getServiceReferences(
1289             const std::string& referenceName) const = 0;
1290
1291
1292     virtual DataObjectPtr getProperties() const = 0;
1293     virtual DataFactoryPtr getDataFactory() const = 0;
1294
1295     virtual ServiceReferencePtr getSelfReference() const = 0;
1296     virtual ServiceReferencePtr getSelfReference(
1297             const std::string& serviceName) const = 0;
1298 };

```

1299 Snippet 6-2: *ComponentContext Class Definition*

1300 6.2.1 getCurrent

1301 A C++ component implementation uses `ComponentContext::getCurrent()` to get a
1302 `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. | |

1303 *Table 6-9: ComponentContext::getCurrent Details*

1304 6.2.2 getURI

1305 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 | |

1306 *Table 6-10: ComponentContext::getURI Details*

1307 6.2.3 getService

1308 A C++ component implementation uses `getService()` to get a service proxy implementing the interface
1309 defined for a Reference.

| | | |
|-----------------|---|--|
| Precondition | C++ component instance is running and has a <code>ComponentContext</code> | |
| Input Parameter | referenceName | 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. | |

1310 *Table 6-11: ComponentContext::getService Details*

1311 6.2.4 getServices

1312 A C++ component implementation uses `getServices()` to get a vector of service proxies implementing
1313 the interface defined for a Reference.

| | | |
|-----------------|---|--|
| Precondition | C++ component instance is running and has a <code>ComponentContext</code> | |
| Input Parameter | referenceName | Name of the Reference to get an interface object for |
| Return | Vector of pointers to <code>ServiceProxy</code> objects implementing the interface of the | |

| | |
|----------------|--|
| | Reference. This vector will be empty if <code>referenceName</code> is not defined for the component. Operations need to be invoked on each object in the vector. |
| Post Condition | ServiceProxy objects for the Reference are constructed. These ServiceProxy objects are independent of any ServiceReferences that are obtained for the Reference. |

1314 *Table 6-12: ComponentContext::getServices Details*

6.2.5 getServiceReference

1316 A C++ component implementation uses `getServiceReference()` to get a `ServiceReference` for a
1317 Reference.

| | | |
|-----------------|--|--|
| Precondition | <code>C++ component instance is running and has a 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. | |

1318 *Table 6-13: ComponentContext::getServiceReference Details*

6.2.6 getServiceReferences

1320 A C++ component implementation uses `getServiceReferences()` to get a vector of
1321 `ServiceReference` for a Reference.

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

1322 *Table 6-14: ComponentContext::getServiceReferences Details*

6.2.7 getProperties

1324 A C++ component implementation uses `getProperties()` to get its configured property values.

| | | |
|-----------------|---|--|
| Precondition | <code>C++ component instance is running and has a ComponentContext</code> | |
| Input Parameter | | |
| Return | An SDO [SDO21] from which all the properties defined in the <code>componentType</code> file can be retrieved. | |
| Post Condition | An SDO with the property values for the component instance is constructed. | |

1325 *Table 6-15: ComponentContext::getProperties Details*

1326 **6.2.8 getDataFactory**

1327 A C++ component implementation uses `getDataFactory()` to get its an SDO `DataFactory` which
1328 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 |

1329 *Table 6-16: ComponentContext::getDataFactory Details*

1330 **6.2.9 getSelfReference**

1331 A C++ component implementation uses `getSelfReference()` to get a `ServiceReference` for use
1332 with some callback APIs.

1333

1334 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 Service |
| Post Condition | A <code>ServiceReference</code> object is constructed |

1335 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 | |

1336 *Table 6-17: ComponentContext::getSelfReference Details*

1337 **6.3 ServiceReference**

1338 The `ServiceReference` interface definition is:

1339

```
1340     class ServiceReference {
1341     public:
1342         virtual ServiceProxyPtr getService() const = 0;
1343
1344         virtual ServiceProxyPtr getCallback() const = 0;
1345     };
```

1346 *Snippet 6-3: ServiceReference Class Definition*

1347

1348 The detailed description of the usage of these member functions is described in Asynchronous
1349 Programming.

1350 **6.3.1 getService**

1351 A C++ component implementation uses `getService()` to get a service proxy implementing the interface
1352 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. | |

1353 *Table 6-18: ServiceReference::getService Details*

1354 **6.3.2 getCallback**

1355 A C++ component implementation uses `getCallback()` to get a service proxy implementing the
1356 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. | |

1357 *Table 6-19: ServiceReference::getCallback Details*

1358 **6.4 DomainContext**

1359 The `DomainContext` interface definition is:

```
1360     class DomainContext {
1361     public:
1362         virtual ServiceProxyPtr getService(
1363             const std::string& serviceURI) const = 0;
1364     };
```

1365 *Snippet 6-4: DomainContext Class Definition*

1366 **6.4.1 getService**

1367 Non-SCA C++ code uses `getService()` to get a service proxy implementing the interface of a service
1368 in an SCA domain.

| | | |
|-----------------|---|---|
| Precondition | None | |
| Input Parameter | serviceURI | 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 ServiceProxy object for the Service is constructed. |
|----------------|---|

1369 *Table 6-20: DomainContext::getService Details*

6.5 SCAException

1371 The SCAException interface definition is:

1372

```
1373     class SCAException : public std::exception {
1374     public:
1375         const char* getEClassName() const;
1376         const char* getMessageText() const;
1377         const char* getFileName() const;
1378         unsigned long getLineNumber() const;
1379         const char* getFunctionName() const;
1380     };
```

1381 *Snippet 6-5: SCAException Class Definition*

1382

1383 The details concerning this class and its derived types are described in Error Handling.

6.5.1 getEClassName

1384 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 |

1386 *Table 6-21: SCAException::getEClassName Details*

6.5.2 getMessageText

1388 A C++ component implementation uses `getMessageText()` to get any message included with the exception.

1389

| | |
|-----------------|---|
| 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 |

1390 *Table 6-22: SCAException::getMessageText Details*

6.5.3 getFileName

1392 A C++ component implementation uses `getFileName()` to get the filename containing the function where the exception occurred.

1393

| | |
|-----------------|---|
| 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 |

1394 *Table 6-23: SCAException::getFileName Details*

6.5.4 getLineNumber

1396 A C++ component implementation uses `getLineNumber()` to get the line number in the source file
 1397 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 |

1398 *Table 6-24: SCAException::getLineNumber Details*

6.5.5 getFunctionName

1400 A C++ component implementation uses `getFunctionName()` to get the function name where the
 1401 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 |

1402 *Table 6-25: SCAException::getFunctionName Details*

6.6 SCANullPointerException

1404 The `SCANullPointerException` interface definition is:

1405

```
1406 class SCANullPointerException : public SCAException {  
1407 };
```

1408 *Snippet 6-6: SCANullPointerException Class Definition*

6.7 ServiceRuntimeException

1410 The `ServiceRuntimeException` interface definition is:

1411

```
1412 class ServiceRuntimeException : public SCAException {  
1413 };
```

1414 *Snippet 6-7: ServiceRuntimeException Class Definition*

1415 **6.8 ServiceUnavailableException**

1416 The ServiceUnavailableException interface definition is:

1417

```
1418     class ServiceUnavailableException : public ServiceRuntimeException {  
1419     };
```

1420 *Snippet 6-8: ServiceUnavailableException Class Definition*

1421 **6.9 MultipleServicesException**

1422 The MultipleServicesException interface definition is:

1423

```
1424     class MultipleServicesException : public ServiceRuntimeException {  
1425     };
```

1426 *Snippet 6-9: MultipleServicesException Class Definition*

7 C++ Contributions

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

7.1 Executable files

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

7.1.1 Executable in contribution

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

Snippet 7-1 shows a contribution containing a DLL.

1443

```
META-INF/
    sca-contribution.xml
bin/
    autoinsurance.dll
AutoInsurance/
    AutoInsurance.composite
    AutoInsuranceService/
        AutoInsurance.h
        AutoInsuranceImpl.componentType
include/
    Customers.h
    Underwriting.h
    RateUtils.h
```

1457 Snippet 7-1: Contribution Containing a DLL

1458

1459 The SCDL for the AutoInsuranceService component of Snippet 7-1 is:

1460

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

1465 Snippet 7-2: Component Definition Using Implementation in a Common DLL

1466

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

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

1471

```
<contribution>
```

```
1473     <deployable composite="myNS:RateUtilities"
1474         <export.cpp name="contribNS:rates" >
1475     </contribution>
```

1476 *Snippet 7-3: Exporting a Contribution*

1477

1478 It is also possible to export only a subtree of a contribution. For a contribution:

1479

```
1480     META-INF/
1481         sca-contribution.xml
1482     bin/
1483         rates.dll
1484     RateUtilities/
1485         RateUtilitiescomposite
1486         RateUtilitiesService/
1487             RateUtils.h
1488             RateUtilsImpl.componentType
```

1489 *Snippet 7-4: Contribution with a Subdirectory to be Shared*

1490

1491 An export of the form:

1492

```
1493     <contribution>
1494         <deployable composite="myNS:RateUtilities"
1495             <export.cpp name="contribNS:ratesbin" path="bin/" >
1496     </contribution>
```

1497 *Snippet 7-5: Exporting a Subdirectory of a Contribution*

1498

1499 only makes the contents of the bin directory visible to other contributions. By placing all of the executable
1500 files of a contribution in a single directory and exporting only that directory, the amount of information
1501 available to a contribution that uses the exported executable files is limited. This is considered a best
1502 practice.

1503 7.1.3 Executable outside of contribution (Import)

1504 When the executable that implements a C++ component is located outside of a contribution, the
1505 contribution has to import the executable. If the executable is located in another contribution, the
1506 **import.cpp** element of the contribution definition uses a **@location** attribute that identifies the name of the
1507 export as defined in the contribution that defined the export as shown in Snippet 7-6.

1508

```
1509     <contribution>
1510         <deployable composite="myNS:Underwriting"
1511             <import.cpp name="rates" location="contribNS:rates">
1512     </contribution>
```

1513 *Snippet 7-6: Contribution with an Import*

1514

1515 The SCDL for the UnderwritingService component of Snippet 7-6 is:

1516

```
1517     <component name="UnderwritingService">
1518         <implementation.cpp library="rates" path="rates:bin/"
1519             class="UnderwritingImpl" />
1520     </component>
```

1521 *Snippet 7-7: Component Definition Using Implementation in an External DLL*

1522
1523 If the executable is located in the file system, the `@location` attribute identifies the location in the files
1524 system used as the root of the import as shown in Snippet 7-8.

1525
1526 <contribution>
1527 <deployable composite="myNS:CustomerUtilities"
1528 <import.cpp name="usr-bin" location="/usr/bin/" />
1529 </contribution>

1530 *Snippet 7-8: Component Definition Using Implementation in a File System*

1531 **7.2 componentType files**

1532 As stated in Component Type and Component, each component implemented in C++ has a
1533 corresponding componentType file. This componentType file is, by default, located in the root directory of
1534 the composite containing the component or a subdirectory of the composite root with the name
1535 `<implementation class>.componentType`, as shown in Snippet 7-9.

1536
1537 META-INF/
1538 sca-contribution.xml
1539 bin/
1540 autoinsurance.dll
1541 AutoInsurance/
1542 AutoInsurance.composite
1543 AutoInsuranceService/
1544 AutoInsurance.h
1545 AutoInsuranceImpl.componentType

1546 *Snippet 7-9: Contribution with ComponentType*

1547
1548 The SCDL for the AutoInsuranceService component of Snippet 7-9 is:

1549
1550 <component name="AutoInsuranceService">
1551 <implementation.cpp library="autoinsurance" path="bin/"
1552 class="AutoInsuranceImpl" />
1553 </component>

1554 *Snippet 7-10: Component Definition with Local ComponentType*

1555
1556 Since there is a one-to-one correspondence between implementations and componentTypes, when an
1557 implementation is shared between contributions, it is desirable to also share the componentType file.
1558 ComponentType files can be exported and imported in the same manner as executable files. The
1559 location of a `.componentType` file can be specified using the `@componentType` attribute of the
1560 `implementation.cpp` element.

1561
1562 <component name="UnderwritingService">
1563 <implementation.cpp library="rates" path="rates:bin/"
1564 class="UnderwritingImpl" componentType="rates:types/UnderwritingImpl"
1565 />
1566 </component>

1567 *Snippet 7-11: Component Definition with Imported ComponentType*

1568 **7.3 C++ Contribution Extensions**

1569 **7.3.1 Export.cpp**

1570 Snippet 7-12 shows the pseudo-schema for the C++ export element used to make an executable or
1571 componentType file visible outside of a contribution.

1572

```
1573     <!-- export.cpp schema snippet -->
1574     <export.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1575         name="QName" path="string"? >
```

1576 *Snippet 7-12: Pseudo-schema for C++ Export Element*

1577

1578 The **export.cpp** element has the following **attributes**:

- 1579 • **name : QName (1..1)** – name of the export. [The @name attribute of a <export.cpp/> element MUST be unique amongst the <export.cpp/> elements in a domain.](#) [The @name attribute of a <export.cpp/> element MUST be unique amongst the <export.cpp/> elements in a domain.](#) [CPP70001]
- 1580
- 1581
- 1582 • **path : string (0..1)** – path of the exported executable relative to the root of the contribution. If not present, the entire contribution is exported.
- 1583

1584 **7.3.2 Import.cpp**

1585 Snippet 7-13 shows the pseudo-schema for the C++ import element used to reference an executable or
1586 componentType file that is outside of a contribution.

1587

```
1588     <!-- import.cpp schema snippet -->
1589     <import.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1590         name="QName" location="string" >
```

1591 *Snippet 7-13: Pseudo-schema for C++ Import Element*

1592

1593 The **import.cpp** element has the following **attributes**:

- 1594 • **name : QName (1..1)** – name of the import. [The @name attribute of a <import.cpp/> child element of a <contribution/> MUST be unique amongst the <import.cpp/> elements in of that contribution.](#) [The @name attribute of a <import.cpp/> child element of a <contribution/> MUST be unique amongst the <import.cpp/> elements in of that contribution.](#) [CPP7002]
- 1595
- 1596
- 1597
- 1598 • **location : string (1..1)** – either the QName of a export or a file system location. If the value does not
1599 match an export name it is taken as an absolute file system path.

1600 8 C++ Interfaces

1601 A service interface can be defined by a C++ class which has only pure virtual public member functions.
1602 The class might additionally have private or protected member functions, but these are not part of the
1603 service interface.

1604 When mapping a C++ interface to WSDL or when comparing two C++ interfaces for compatibility, as
1605 defined by the Assembly specification [ASSEMBLY], it is necessary for an SCA implementation to
1606 determine the signature (return type, name, and the names and types of the parameters) of every
1607 member function of the class defining the service interface. An SCA implementation MUST translate a
1608 class to tokens as part of conversion to WSDL or compatibility testing.An SCA implementation MUST
1609 translate a class to tokens as part of conversion to WSDL or compatibility testing. [CPP80001] Macros
1610 and typedefs in member function declarations might lead to portability problems. Complete member
1611 function declarations within a macro are discouraged. The processing of typedefs needs to be aware of
1612 the types that impact mapping to WSDL (see Table 9-1 and Table 9-2)

1613 8.1 Types Supported in Service Interfaces

1614 Not all service interfaces support the complete set of the types available in C++.

1615 8.1.1 Local Service

1616 Any fundamental or compound type defined by C++ can be used in the interface of a local service.

1617 8.1.2 Remotable Service

1618 For a remotable service being called by another service the data exchange semantics is by-value. The
1619 return type and types of the parameters of a member function of a remotable service interface MUST be
1620 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.

1625 An SDO DataObjectPtr instance. This type may be passed by-value, by-reference, or by-pointer.
1626 Unless the member function and client indicate that they allow by-reference semantics (see
1627 AllowsPassByReference), a deep-copy of the DataObjectPtr will be created by the runtime for any
1628 parameters passed by-value, by-reference, or by-pointer. When by-reference semantics are allowed, the
1629 DataObjectPtr itself will be passed. The return type and types of the parameters of a member function
1630 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.

1635 • An SDO DataObjectPtr instance. This type may be passed by-value, by-reference, or by-pointer.
1636 Unless the member function and client indicate that they allow by-reference semantics (see
1637 AllowsPassByReference), a deep-copy of the DataObjectPtr will be created by the runtime for any
1638 parameters passed by-value, by-reference, or by-pointer. When by-reference semantics are allowed,
1639 the DataObjectPtr itself will be passed. [CPP80002]

1640 8.2 Header Files

1641 A C++ header file used to define an interface MUST declare at least one class with:

- At least one public member function.

- 1643 All public member functions are pure virtual.A C++ header file used to define an interface **MUST** declare
1644 at least one class with:
1645 •At least one public member function.
1646 • All public member functions are pure virtual. [CPP80003]

9 WSDL to C++ and C++ to WSDL Mapping

1648 The SCA Client and Implementation Model for C++ applies the WSDL to Java and Java to WSDL
1649 mapping rules (augmented for C++) as defined by the JAX-WS specification [JAXWS21] for generating
1650 remotable C++ interfaces from WSDL portTypes and vice versa. Use of the JAX-WS specification as a
1651 guideline for WSDL to C++ and C++ to WSDL mappings does not imply that any support for the Java
1652 language is mandated by this specification.

1653 For the purposes of the C++ to WSDL mapping algorithm, the interface is treated as if it had a
1654 @WebService annotation on the class, even if it doesn't. For the WSDL to C++ mapping, the generated
1655 @WebService annotation implies that the interface is @Remotable.

1656 For the mapping from C++ types to XML schema types SCA supports the SDO 2.1 [SDO21] mapping. A
1657 detailed mapping of C++ to WSDL types and WSDL to C++ types is covered in section SDO Data
1658 Binding.

1659 Items in the JAX-WS WSDL to Java and Java to WSDL mapping that are not supported:

- 1660 • JAX-WS style external binding. (See JAX-WS Sec. 2)
- 1661 • MIME binding. (See JAX-WS Sec. 2.1.1)
- 1662 • Holder classes. (See JAX-WS Sec. 2.3.3)
- 1663 • Asynchronous mapping. (See JAX-WS Sec. 2.3.4)
- 1664 • Generation of Service classes from WSDL. (See JAX-WS Sec. 2.7)
- 1665 • Generation of WSDL from Service implementation classes (See JAX-WS Sec. 3.3)
- 1666 • Templates when converting from C++ to WSDL (See JAX-WS Sec. 3.9)

1667 General rules for the application of JAX-WS to C++.

- 1668 • References to Java are considered references to C++.
- 1669 • References to Java classes are considered references to C++ classes.
- 1670 • References to Java methods are considered references to C++ member functions.
- 1671 • References to Java interfaces are considered references to C++ classes which only define pure
1672 virtual member functions.

1673 Major divergences from JAX-WS:

- 1674 • Algorithms for converting WSDL namespaces to C++ namespaces (and vice-versa).
- 1675 • Mapping of WSDL faults to C++ exceptions and vice-versa.
- 1676 • Managing of data bindings.

9.1 Augmentations for WSDL to C++ Mapping

1678 An SCA implementation MUST map a WSDL portType to a remotable C++ interface definition.
1679 [CPP100009]

9.1.1 Mapping WSDL targetNamespace to a C++ namespace

1681 Since C++ does not define a standard convention for the use of namespaces, the SCA specification does
1682 not define an implicit mapping of WSDL targetNamespaces to C++ namespaces. A WSDL file might
1683 define a namespace using the <sca:namespace> WSDL extension, otherwise all C++ classes MUST be
1684 placed in a default namespace as determined by the implementation. Implementations SHOULD provide
1685 a mechanism for overriding the default namespace. [CPP100001]

9.1.2 Mapping WSDL Faults to C++ Exceptions

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

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

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

Snippet 9-1: Fault Exception Class Member Functions

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

9.1.2.1 Multiple Fault References

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. [CPP100002]

9.1.3 Mapping of in, out, in/out parts to C++ member function parameters

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

- For unwrapped messages, an SCA implementation MUST map:

- **in** - the message part to a member function parameter, passed by const-reference.
- **out** - the message part to a member function parameter, passed by reference, or to the member function return type, returned by-value.

- in/out** - the message part to a member function parameter, passed by reference. For unwrapped messages, an SCA implementation MUST map:

- **in** - the message part to a member function parameter, passed by const-reference.
- **out** - the message part to a member function parameter, passed by reference, or to the member function return type, returned by-value.
- **in/out** - the message part to a member function parameter, passed by reference. [CPP100003]

- For wrapped messages, an SCA implementation MUST map:

- **in** - the wrapper child to a member function parameter, passed by const-reference.
- **out** - the wrapper child to a member function parameter, passed by reference, or to the member function return type, returned by-value.

- in/out** - the wrapper child to a member function parameter, passed by reference. For wrapped messages, an SCA implementation MUST map:

- **in** - the wrapper child to a member function parameter, passed by const-reference.
- **out** - the wrapper child to a member function parameter, passed by reference, or to the member function return type, returned by-value.
- **in/out** - the wrapper child to a member function parameter, passed by reference. [CPP100004]

Formatter

Formatter

1774 9.2 Augmentations for C++ to WSDL Mapping

1775 Where annotations are discussed as a means for an application to control the mapping to WSDL, an
1776 implementation-specific means of controlling the mapping can be used instead.

1777 An SCA implementation MUST map a C++ interface definition to WSDL as if it has a @WebService
1778 annotation with all default values on the class. An SCA implementation MUST map a C++ interface
1779 definition to WSDL as if it has a @WebService annotation with all default values on the class.
1780 [CPP100010]

1781 An application can customize the name of the portType and port using the @WebService annotation.

1782 9.2.1 Mapping C++ namespaces to WSDL namespaces

1783 Since C++ does not define a standard convention for the use of namespaces, the SCA specification does
1784 not define an implicit mapping of C++ namespaces to WSDL namespace URIs. The default
1785 targetNamespace is defined by the implementation. An SCA implementation SHOULD provide a
1786 mechanism for overriding the default targetNamespace. [CPP100005]

1787 9.2.2 Parameter and return type classification

1788 The classification of parameters and return types in C++ are determined based on how the value is
1789 passed into the function.

1790 An SCA implementation MUST map a method's return type as an **out** parameter, a parameter passed by-
1791 reference or by-pointer as an **in/out** parameter, and all other parameters, including those passed by-
1792 const-reference as **in** parameters. An SCA implementation MUST map a method's return type as an **out**
1793 parameter, a parameter passed by-reference or by-pointer as an **in/out** parameter, and all other
1794 parameters, including those passed by-const-reference as **in** parameters. [CPP100006]

1795 An application can customize parameter classification using the @WebParam annotation.

1796 9.2.3 C++ to WSDL Type Conversion

1797 C++ types are mapped to WSDL and schema types based on the mapping described in Section Simple
1798 Content Binding.

1799 9.2.4 Service-specific Exceptions

1800 C++ classes that define a web service interface can indicate which faults they might throw using the
1801 @WebThrows annotation. @WebThrows lists the names of each C++ class that might be thrown as a
1802 fault from a particular member function. By default, no exceptions are mapped to operation faults.

1803 9.3 SDO Data Binding

1804 9.3.1 Simple Content Binding

1805 The translation of XSD simple content types to C++ types follows the convention defined in the SDO
1806 specification. Table 9-1 summarizes that mapping as it applies to SCA services.

| XSD Schema Type → | C++ Type | → XSD Schema Type |
|-------------------|--------------------------|-------------------|
| 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 | stirng |
| 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 |

1808 *Table 9-1: XSD simple type to C++ type mapping*

1809

1810 Table 9-2 defines the mapping of C++ types to XSD schema types that are not covered in Table 9-1.

1811

| C++ Type → | XSD Schema Type |
|--------------------|------------------------|
| 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 |

1812 *Table 9-2: C++ type to XSD type mapping*

1813

1814 The C++ standard does not define value ranges for integer types so it is possible that on a platform
 1815 parameters or return values could have values that are out of range for the default XSD schema type. In

1816 these circumstances, the mapping would need to be customized, using @WebParam or @WebResult if
1817 supported, or some other implementation-specific mechanism.

1818 | ~~An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.~~
1819 | ~~An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.~~
1820 | [CPP100008]

1821 **9.3.2 Complex Content Binding**

1822 Any XSD complex types are mapped to an instance of an SDO DataObject.

1823 10 Conformance

1824 The XML schema pointed to by the RDDL document at the SCA namespace URI, defined by the
1825 Assembly specification **[ASSEMBLY]** and extended by this specification, are considered to be
1826 authoritative and take precedence over the XML schema in this document.
1827 The XML schema pointed to by the RDDL document at the SCA C++ namespace URI, defined by this
1828 specification, is considered to be authoritative and takes precedence over the XML schema in this
1829 document.
1830 Normative code artifacts related to this specification are considered to be authoritative and take
1831 precedence over specification text.

1832 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>. An SCA implementation MUST reject a composite file that does not conform to <http://docs.oasis-open.org/openesa/sea/200912/sca-interface-cpp-1.1.xsd> or <http://docs.oasis-open.org/openesa/sea/200912/sca-implementation-cpp-1.1.xsd>. [CPP110001]
1833 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>. 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>. [CPP110002]
1834 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>. 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]
1835 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-wsdlxext-cpp-1.1.xsd>. 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-wsdlxext-cpp-1.1.xsd>. [CPP110004]

1850 10.1 Conformance Targets

1851 The conformance targets of this specification are:

- 1852 • **SCA implementations**, which provide a **runtime** for SCA components and potentially **tools** for authoring SCA artifacts, component descriptions and/or runtime operations.
- 1853 • **SCA documents**, which describe SCA artifacts, and specific **elements** within these documents.
- 1854 • **C++ component implementations**, which execute under the control of an SCA runtime.
- 1855 • **C++ files**, which define SCA service interfaces and implementations.
- 1856 • **WSDL files**, which define SCA service interfaces.

1858 10.2 SCA Implementations

1859 An implementation conforms to this specification if it meets these conditions:

- 1860 1. It MUST conform to the SCA Assembly Model Specification **[ASSEMBLY]** and the SCA Policy Framework **[POLICY]**.
- 1861 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.
- 1862 3. It MUST implement the SCA C++ API defined in section C++ API.
- 1863 4. It MUST implement the mapping between C++ and WSDL 1.1 **[WSDL11]** defined in WSDL to C++ and C++ to WSDL Mapping.

- 1867 5. It MUST support <interface.cpp/> and <implementation.cpp/> elements as defined in Component
1868 Type and Component in composite and componentType documents.
- 1869 6. It MUST support <export.cpp/> and <import.cpp/> elements as defined in C++ Contributions in
1870 contribution documents.
- 1871 7. It MAY support source file annotations as defined in C++ SCA Annotations, C++ SCA Policy
1872 Annotations and C++ WSDL Mapping Annotations. If source file annotations are supported, the
1873 implementation MUST comply with all statements in Table F-2 related to an SCA implementation,
1874 notably all mandatory statements in that section have to be implemented.
- 1875 8. It MAY support WSDL extentsions as defined in WSDL C++ Mapping Extensions. If WSDL
1876 extentsionsare supported, the implementation MUST comply with all statements in Table F-3 related
1877 to an SCA implementation, notably all mandatory statements in that section have to be implemented.

1878 **10.3 SCA Documents**

- 1879 An SCA document conforms to this specification if it meets these condition:
- 1880 1. It MUST conform to the SCA Assembly Model Specification [**ASSEMBLY**] and, if appropriate, the
1881 SCA Policy Framework [**POLICY**].
- 1882 2. If it is a composite document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-cpp-1.1.xsd> and <http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-cpp-1.1.xsd> schema and MUST comply with the
1883 additional constraints on the document contents as defined in Table F-1.
1884
1885 If it is a componentType document, it MUST conforms to the <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-cpp-1.1.xsd> schema and MUST comply with the
1886 additional constraints on the document contents as defined in Table F-1.
1887
1888 If it is a contribution document, it MUST conforms to the <http://docs.oasis-open.org/opencsa/sca/200912/sca-contribution-cpp-1.1.xsd> schema and MUST comply with the
1889 additional constraints on the document contents as defined in Table F-1.
1890
1891

1892 **10.4 C++ Files**

- 1893 A C++ files conforms to this specification if it meets the condition:
- 1894 1. It MUST comply with all statements in Table F-1, Table F-2 and Table F-4 related to C++ contents
1895 and annotations, notably all mandatory statements have to be satisfied.

1896 **10.5 WSDL Files**

- 1897 A WSDL file conforms to this specification if it meets these conditions:
- 1898 1. It is a valid WSDL 1.1 [**WSDL11**] document.
- 1899 2. It MUST comply with all statements in Table F-1, Table F-3 and Table F-4 related to WSDL contents
1900 and extensions, notably all mandatory statements have to be satisfied.

1901 A C++ SCA Annotations

1902 To allow developers to define SCA related information directly in source files, without having to separately
1903 author SCDL files, a set of annotations is defined. If annotations are supported by an implementation, the
1904 annotations defined here MUST be supported and MUST be mapped to SCDL as described. The SCA
1905 runtime MUST only process the SCDL files and not the annotations. [CPPA0001]

1906 The annotations are defined as C++ comments in interface and implementation header files, for example:

```
1908 // @Scope("stateless")
```

1909 *Snippet A-1: Example Annotation*

1910 A.1 Application of Annotations to C++ Program Elements

1911 In general an annotation immediately precedes the program element it applies to. If multiple annotations
1912 apply to a program element, all of the annotations SHOULD be in the same comment block. [CPPA0002]

- 1913 • Class

1914 The annotation immediately precedes the class.

1915 Example:

```
1916 // @Scope("composite")
1917 class LoanServiceImpl : public LoanService {
1918     ...
1919 };
```

1920 *Snippet A-2: Example Class Annotation*

- 1921 • Member function

1922 The annotation immediately precedes the member function.

1923 Example:

```
1924 class LoanService
1925 {
1926     public:
1927         // @OneWay
1928         virtual void reportEvent(int eventId) = 0;
1929         ...
1930 };
```

1931 *Snippet A-3: Example Member Function Annotation*

- 1932 • Data Member

1933 The annotation immediately precedes the data member.

1934 Example:

```
1935 // @Property(name="loanType", type="xsd:int")
1936 long loanType;
```

1937 *Snippet A-4: Example Data Member Annotation*

1938
1939 Annotations follow normal inheritance rules. An annotation on a base class or any element of a base
1940 class applies to any classes derived from the base class.

1941 A.2 Interface Header Annotations

1942 This section lists the annotations that can be used in the header file that defines a service interface.

1943 A.2.1 @Interface

1944 Annotation used to indicate a class defines an interface when multiple classes exist in a header file.

1945 **Corresponds to:** @class attribute of an *interface.cpp* element.

1946 **Format:**

```
1947 // @Interface
```

1948 *Snippet A-5: @Interface Annotation Format*

1949 **Applies to:** Class

1950 Example:

1951 Interface header:

```
1952 // @Interface
1953 class LoanService {
1954 ...
1955 };
```

1956

1957 Service definition:

```
1958 <service name="LoanService">
1959   <interface.cpp header="LoanService.h" class="LoanService" />
1960 </service>
```

1961 *Snippet A-6: Example of @Interface Annotation*

1962 A.2.2 @Remotable

1963 Annotation on service interface class to indicate that a service is remotable and implies an @Interface
1964 annotation applies as well. [An SCA implementation MUST treat a class with an @WebService annotation](#)
1965 [specified as if a @Remotable annotation was specified. An SCA implementation MUST treat a class with](#)
1966 [an @WebService annotation specified as if a @Remotable annotation was specified. \[CPPA0003\]](#)

1967 **Corresponds to:** @remotable="true" attribute of an *interface.cpp* element.

1968 **Format:**

```
1969 // @Remotable
```

1970 *Snippet A-7: @Remotable Annotation Format*

1971 The default is **false** (not remotable).

1972 **Applies to:** Class

1973 Example:

1974 Interface header:

```
1975 // @Remotable
1976 class LoanService {
1977 ...
1978 };
```

1979

1980 Service definition:

```
1981 <service name="LoanService">
1982   <interface.cpp header="LoanService.h" remotable="true" />
1983 </service>
```

1984 *Snippet A-8: Example of @Remotable Annotation*

1985 A.2.3 @Callback

1986 Annotation on a service interface class to specify the callback interface.

1987 **Corresponds to:** @callbackHeader and @callbackClass attributes of an *interface.cpp* element.

1988 **Format:**

```
1989 // @Callback(header="headerName", class="className")
```

1990 *Snippet A-9: @Callback Annotation Format*

1991 where

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

1994 **Applies to:** Class

1995 Example:

1996 Interface header:

```
1997 // @Callback(header="MyServiceCallback.h", class="MyServiceCallback")
1998 class MyService {
1999 public:
2000     virtual void someFunction( unsigned int arg ) = 0;
2001 };
```

2002

2003 Service definition:

```
2004 <service name="MyService">
2005   <interface.cpp header="MyService.h"
2006     callbackHeader="MyServiceCallback.h"
2007     callbackClass="MyServiceCallback" />
2008 </service>
```

2009 *Snippet A-10: Example of @Callback Annotation*

2010 **A.2.4 @OneWay**

2011 Annotation on an interface member function to indicate the member function is one way. The @OneWay
2012 annotation also affects the representation of a service in WSDL, see @OneWay.

2013 **Corresponds to:** @oneWay="true" attribute of *function* element of an *interface.cpp* element.

2014 **Format:**

```
2015 // @OneWay
```

2016 *Snippet A-11: @OneWay Annotation Format*

2017 The default is **false** (not OneWay).

2018 **Applies to:** Member function

2019 Example:

2020 Interface header:

```
2021 class LoanService
2022 {
2023     public:
2024     // @OneWay
2025     virtual void reportEvent(int eventId) = 0;
2026     ...
2027 };
```

2028

2029 Service definition:

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

```
2034     </service>
2035 Snippet A-12: Example of @OneWay Annotation
```

2036 A.2.5 @Function

2037 Annotation on a interface member function to modify its representation in an SCA interface. An SCA
2038 implementation MUST treat a member function with a @WebFunction annotation specified as if
2039 @Function was specified with the operationName value of the @WebFunction annotation used as the
2040 name value of the @Function annotation and the exclude value of the @WebFunction annotation used as
2041 the exclude valued of the @Function annotation. [CPPA0004]

2042 **Corresponds to:** *function* or *callbackFunction* element of an *interface.cpp* element. If the class the
2043 function is a member of is being processed because it was identified via either a combination of
2044 *interface.cpp/@callbackHeader* and *interface.cpp/@callbackClass* or a @Callback annotation, then the
2045 @Function annotation corresponds to a *callbackFunction* element, otherwise it corresponds to a *function*
2046 element.

2047 **Format:**

```
2048 // @Function(name="operationName", exclude="true")
```

2049 Snippet A-13: @Function Annotation Format

2050 where

- **name : NCName (0..1)** – specifies the name of the operation. The default operation name is the
2052 function name.
- **exclude : boolean (0..1)** – specifies whether this member function is to be excluded from the SCA
2054 interface. Default is **false**.

2055 **Applies to:** Member function

2056 Example:

2057 Interface header:

```
2058 class LoanService
2059 {
2060     public:
2061     ...
2062     // @Function(exclude="true")
2063     virtual void reportEvent(int eventId) = 0;
2064     ...
2065 }
```

2066

2067 Service definition:

```
2068 <service name="LoanService">
2069   <interface.cpp header="LoanService.h">
2070     <function name="reportEvent" exclude="true" />
2071   </interface.cpp>
2072 </service>
```

2073 Snippet A-14: Example of @Function Annotation

2074 A.3 Implementation Header Annotations

2075 This section lists the annotations that can be used in the header file that defines a service
2076 implementation.

2077 A.3.1 @ComponentType

2078 Annotation used to indicate which class implements a componentType when multiple classes exist in an
2079 implementation file.

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

2081 **Format:**

2082

```
// @ComponentType
```

2083 *Snippet A-15: @ComponentType Annotation Format*

2084 **Applies to:** Class

2085 Example:

2086 Implementation header:

2087

```
// @ComponentType
```


2088 class LoanServiceImpl : public LoanService {
2089 ...
2090 };

2091

2092 Component definition:

2093

```
<component name="LoanService">  
    <implementation.cpp library="loan" class="LoanServiceImpl" />  
</component>
```

2096 *Snippet A-16: Example of @ComponentType Annotation*

A.3.2 @Scope

2098 Annotation on a service implementation class to indicate the scope of the service.

2099 **Corresponds to:** @scope attribute of an *implementation.cpp* element.

2100 **Format:**

2101

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

2102 *Snippet A-17: @Scope Annotation Format*

2103 where

- 2104 • **value : [stateless / composite] (1..1)** – specifies the scope of the implementation. The default value
2105 is **stateless**.

2106 **Applies to:** Class

2107 Example:

2108 Implementation header:

2109

```
// @Scope("composite")
```


2110 class LoanServiceImpl : public LoanService {
2111 ...
2112 };

2113

2114 Component definition:

2115

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

2119 *Snippet A-18: Example of @Scope Annotation*

A.3.3 @EagerInit

2121 Annotation on a service implementation class to indicate the implantation is to be instantiated when its
2122 containing component is started.

2123 **Corresponds to:** @eagerInit="true" attribute of an *implementation.cpp* element.

2124 **Format:**
2125 // @EagerInit

2126 *Snippet A-19: @EagerInit Annotation Format*
2127 The default is **false** (the service is initialized lazily).

2128 **Applies to:** Class
2129 Example:

2130 Implementation header:
2131 // @EagerInit
2132 class LoanServiceImpl : public LoanService {
2133 ...
2134 };

2135
2136 Component definition:

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

2141 *Snippet A-20: Example of @EagerInit Annotation*

2142 **A.3.4 @AllowsPassByReference**

2143 Annotation on service implementation class or member function to indicate that a service or member
2144 function allows pass by reference semantics.

2145 **Corresponds to:** `@allowsPassByReference="true"` attribute of an *implementation.cpp* element or a
2146 *function* child element of an *implementation.cpp* element.

2147 **Format:**

2148 // @AllowsPassByReference

2149 *Snippet A-21: @AllowsPassByReference Annotation Format*

2150 The default is **false** (the service does not allow by reference parameters).

2151 **Applies to:** Class or Member function

2152 Example:

2153 Implementation header:
2154 // @AllowsPassByReference
2155 class LoanService {
2156 ...
2157 };

2158
2159 Component definition:

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

2164 *Snippet A-22: Example of @AllowsPassByReference Annotation*

2165 **A.3.5 @Property**

2166 Annotation on a service implementation class data member to define a property of the service.

2167 **Corresponds to:** *property* element of a *componentType* element.

2168 **Format:**

```
2169     // @Property(name="propertyName", type="typeQName"
2170     //           default="defaultValue", required="true")
```

2171 *Snippet A-23: @Property Annotation Format*

2172 where

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

2173 **Applies to:** DataMember

2174 Example:

2175 Implementation:

```
2176     // @Property(name="loanType", type="xsd:int")
2177     long loanType;
```

2178 Component Type definition:

```
2179     <componentType ... >
2180         <service ... />
2181         <property name="loanType" type="xsd:int" />
2182     </componentType>
```

2183 *Snippet A-24: Example of @Property Annotation*

2184 A.3.6 @Reference

2185 Annotation on a service implementation class data member to define a reference of the service.

2186 **Corresponds to:** *reference* element of a *componentType* element.

2187 **Format:**

```
2188     // @Reference(name="referenceName", interfaceHeader="LoanService.h",
2189     //               interfaceClass="LoanService", required="true")
```

2190 *Snippet A-25: @Reference Annotation Format*

2191 where

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

2192 If the annotated data member is a std::vector then the implied component type has a reference with a multiplicity of either 0..n or 1..n depending on the value of the @Reference *required* attribute – 1..n applies if *required=true*. Otherwise a multiplicity of 0..1 or 1..1 is implied.

2193 **Applies to:** Data Member

2194 Example:

```

2212     Implementation:
2213     // @Reference(interfaceHeader="LoanService.h" required="true")
2214     LoanServiceProxyPtr loanService;
2215
2216     // @Reference(interfaceHeader="LoanService.h" required="false")
2217     std::vector<LoanServiceProxyPtr> loanServices;
2218
2219     Component Type definition:
2220     <componentType ... >
2221         <service ... />
2222         <reference name="loanService" multiplicity="1..1">
2223             <interface.cpp header="LoanService.h" class="LoanService" />
2224         </reference>
2225         <reference name="loanServices" multiplicity="0..n">
2226             <interface.cpp header="LoanService.h" class="LoanService" />
2227         </reference>
2228     </componentType>

```

2229 *Snippet A-26: Example of @Reference Annotation*

2230 A.4 Base Annotation Grammar

```

2231 <annotation> ::= // @<baseAnnotation>
2232
2233 <baseAnnotation> ::= <name> [(<params>)]
2234
2235 <params> ::= <paramNameValue>[, <paramNameValue>]* | 
2236     <paramValue>[, <paramValue>]*
2237
2238 <paramNameValue> ::= <name>=<value>
2239
2240 <paramValue> ::= "<value>"
2241
2242 <name> ::= NCName
2243
2244 <value> ::= string

```

2245 *Snippet A-27: Base Annotation Grammar*

- 2246 • Adjacent string constants are concatenated
- 2247 • NCName is as defined by XML schema [\[XSD\]](#)[\[XSD\]](#)
- 2248 • Whitespace including newlines between tokens is ignored.
- 2249 • Annotations with parameters can span multiple lines within a comment, and are considered complete
 2250 when the terminating ")" is reached.

2251 B C++ SCA Policy Annotations

2252 SCA provides facilities for the attachment of policy-related metadata to SCA assemblies, which influence
2253 how implementations, services and references behave at runtime. The policy facilities are described in
2254 [POLICY]. In particular, the facilities include Intents and Policy Sets, where intents express abstract,
2255 high-level policy requirements and policy sets express low-level detailed concrete policies.

2256 Policy metadata can be added to SCA assemblies through the means of declarative statements placed
2257 into Composite documents and into Component Type documents. These annotations are completely
2258 independent of implementation code, allowing policy to be applied during the assembly and deployment
2259 phases of application development.

2260 However, it can be useful and more natural to attach policy metadata directly to the code of
2261 implementations. This is particularly important where the policies concerned are relied on by the code
2262 itself. An example of this from the Security domain is where the implementation code expects to run
2263 under a specific security Role and where any service operations invoked on the implementation have to
2264 be authorized to ensure that the client has the correct rights to use the operations concerned. By
2265 annotating the code with appropriate policy metadata, the developer can rest assured that this metadata
2266 is not lost or forgotten during the assembly and deployment phases.

2267 The SCA C++ policy annotations provide the capability for the developer to attach policy information to
2268 C++ implementation code. The annotations provide both general facilities for attaching SCA Intents and
2269 Policy Sets to C++ code and annotations that deal with specific policy intents. Policy annotation can be
2270 used in header files for service interfaces or implementations.

2271 B.1 General Intent Annotations

2272 SCA provides the annotation **@Requires** for the attachment of any intent to a C++ class, to a C++
2273 interface or to elements within classes and interfaces such as member functions and data members.

2274 The **@Requires** annotation can attach one or multiple intents in a single statement. Each intent is
2275 expressed as a string. Intents are XML QNames, which consist of a Namespace URI followed by the
2276 name of the Intent. The precise form used is:

```
2277 " { " + Namespace URI + " } " + intentname
```

2278 *Snippet B-1: Intent Format*

2279

2280 Intents can be qualified, in which case the string consists of the base intent name, followed by a ".",
2281 followed by the name of the qualifier. There can also be multiple levels of qualification.

2282 This representation is quite verbose, so we expect that reusable constants will be defined for the
2283 namespace part of this string, as well as for each intent that is used by C++ code. SCA defines constants
2284 for intents such as the following:

```
2285
2286 // @Define SCA_PREFIX "{http://docs.oasis-
2287 open.org/ns/opencsa/sca/200912}"
2288 // @Define CONFIDENTIALITY SCA PREFIX ## "confidentiality"
2289 // @Define CONFIDENTIALITY_MESSAGE CONFIDENTIALITY ## ".message"
```

2290 *Snippet B-2: Example Intent Constants*

2291

2292 Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,
2293 separated by an underscore. These intent constants are defined in the file that defines an annotation for
2294 the intent (annotations for intents, and the formal definition of these constants, are covered in a following
2295 section).

2296 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.
2297 **Corresponds to:** @requires attribute of an *interface.cpp*, *implementation.cpp*, *function* or *callbackFunction*
2298 element.

2299 **Format:**

```
2300 // @Requires("qualifiedIntent" | {"qualifiedIntent" [, "qualifiedIntent"]})
```

2301 where

```
2302 qualifiedIntent ::= QName | QName.qualifier | QName.qualifier1.qualifier2
```

2303 *Snippet B-3: @Requires Annotation Format*

2304 **Applies to:** Class, Member Function

2305 Examples:

2306 Attaching the intents "confidentiality.message" and "integrity.message".

```
2307 // @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE})
```

2308 *Snippet B-4: Example @Requires Annotation*

2309

2310 A reference requiring support for confidentiality:

```
2311 class Foo {  
2312     ...  
2313     // @Requires(CONFIDENTIALITY)  
2314     // @Reference(interfaceHeader="SetBar.h")  
2315     void setBar(Bar* bar);  
2316     ...  
2317 }
```

2318 *Snippet B-5: @Requires Annotation applied with an @Reference Annotation*

2319

2320 Users can also choose to only use constants for the namespace part of the QName, so that they can add
2321 new intents without having to define new constants. In that case, this definition would instead look like
2322 this:

2323

```
2324 class Foo {  
2325     ...  
2326     // @Requires(SCA_PREFIX "confidentiality ")  
2327     // @Reference(interfaceHeader="SetBar.h")  
2328     void setBar(Bar* bar);  
2329     ...  
2330 }
```

2331 *Snippet B-6: @Requires Annotation Using Mixed Constants and Literals*

2332 **B.2 Specific Intent Annotations**

2333 In addition to the general intent annotation supplied by the @Requires annotation described above, there
2334 are C++ annotations that correspond to specific policy intents.

2335 The general form of these specific intent annotations is an annotation with a name derived from the name
2336 of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation
2337 in the form of a string or an array of strings.

2338 For example, the SCA confidentiality intent described in General Intent Annotations using the
2339 @Requires(CONFIDENTIALITY) intent can also be specified with the specific @Confidentiality intent
2340 annotation. The specific intent annotation for the "integrity" security intent is:

2341

```
2342 // @Integrity
```

2343 Snippet B-7: Example Specific Intent Annotation

2344

2345 **Corresponds to:** @requires="<Intent>" attribute of an *interface.cpp*, *implementation.cpp*, *function* or
2346 *callbackFunction* element.

2347 **Format:**

```
2348 // @<Intent>[(qualifiers)]
```

2349 where Intent is an NCName that denotes a particular type of intent.

```
2350 Intent ::= NCName
2351 qualifiers ::= "qualifier" | {"qualifier" [, "qualifier"] }
2352 qualifier ::= NCName | NCName/qualifier
```

2353 Snippet B-8: @<Intent> Annotation Format

2354 **Applies to:** Class, Member Function – but see specific intents for restrictions

2355 Example:

```
2356 // @ClientAuthentication( {"message", "transport"} )
```

2357 Snippet B-9: Example @<Intent> Annotation

2358

2359 This annotation attaches the pair of qualified intents: *authentication.message* and *authentication.transport*
2360 (the sca: namespace is assumed in both of these cases – "http://docs.oasis-
2361 open.org/opencsa/ns/sca/200912").

2362

2363 The Policy Framework **[POLICY]** defines a number of intents and qualifiers. Security Interaction –
2364 Miscellaneous define the annotations for those intents.

2365 B.2.1 Security Interaction

| Intent | Annotation |
|----------------------|-----------------------|
| clientAuthentication | @ClientAuthentication |
| serverAuthentication | @ServerAuthentication |
| mutualAuthentication | @MutualAuthentication |
| confidentiality | @Confidentiality |
| integrity | @Integrity |

2366 Table B-1: Security Interaction Intent Annotations

2367

2368 These three intents can be qualified with

- 2369 • transport
2370 • message

2371 B.2.2 Security Implementation

| Intent | Annotation | Qualifiers |
|---------------|----------------|------------|
| authorization | @Authorization | fine_grain |

2372 Table B-2: Security Implementation Intent Annotations

2373 **B.2.3 Reliable Messaging**

| Intent | Annotation |
|-------------|--------------|
| atLeastOnce | @AtLeastOnce |
| atMostOnce | @AtMostOnce |
| ordered | @Ordered |
| exactlyOnce | @ExactlyOnce |

2374 *Table B-3: Reliable Messaging Intent Annotations*

2375 **B.2.4 Transactions**

| Intent | Annotation | Qualifiers |
|------------------------|------------------------|-----------------|
| managedTransaction | @ManagedTransaction | local global |
| noManagedTransaction | @NoManagedTransaction | |
| transactedOneWay | @TransactedOneWay | |
| immediateOneWay | @ImmediateOneWay | |
| propagates Transaction | @PropagatesTransaction | |
| suspendsTransaction | @SuspendsTransaction | |

2376 *Table B-4: Transaction Intent Annotations*

2377 **B.2.5 Miscellaneous**

| Intent | Annotation | Qualifiers |
|--------|------------|--------------|
| SOAP | @SOAP | v1_1 v1_2 |

2378 *Table B-5: Miscellaneous Intent Annotations*

2379 **B.3 Policy Set Annotations**

2380 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies (for example,
2381 a concrete policy is the specific encryption algorithm to use when encrypting messages when using a
2382 specific communication protocol to link a reference to a service).

2383 Policy Sets can be applied directly to C++ implementations using the **@PolicySets** annotation. The
2384 PolicySets annotation either takes the QName of a single policy set as a string or the name of two or
2385 more policy sets as an array of strings.

2386 **Corresponds to:** @policySets attribute of an *interface.cpp*, *implementation.cpp*, *function* or
2387 *callbackFuncion* element.

2388 **Format:**

```
2389 // @PolicySets("<policy set QName>" |
2390 //           { "<policy set QName>" [, "<policy set QName>"] })
```

2391 *Snippet B-10: @PolicySets Annotation Format*

2392 As for intents, PolicySet names are QNames – in the form of “{Namespace-URI}localPart”.

2393 **Applies to:** Class, Member Function,

2394 Example:

```
2395 // @Reference(name="helloService", interfaceHeader="helloService.h",
2396 //           required="true")
2397 // @PolicySets({ MY_NS "WS_Encryption_Policy",
2398 //                 MY_NS "WS_Authentication_Policy"})
2399 HelloService* helloService;
2400 ...
```

2401 *Snippet B-11: Example @PolicySets Annotation*

2402

2403 In this case, the Policy Sets WS_Encryption_Policy and WS_Authentication_Policy are applied, both
2404 using the namespace defined for the constant MY_NS.

2405 PolicySets satisfy intents expressed for the implementation when both are present, according to the rules
2406 defined in [POLICY].

2407 **B.4 Policy Annotation Grammar Additions**

```
2408 <annotation> ::= // @<baseAnnotation> | @<requiresAnnotation> | @<intentAnnotation> | @<policySetAnnotation>
2409
2410
2411 <requiresAnnotation> ::= Requires(<intents>)
2412
2413 <intents> ::= "<qualifiedIntent>" |
2414   {"<qualifiedIntent>"[, "<qualifiedIntent>"]*})
2415
2416 <qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |
2417   <intentName>.<qualifier>.qualifier>
2418
2419 <intentName> ::= {aAnyURI}NCName
2420
2421 <intentAnnotation> ::= <intent>[(<qualifiers>)]
2422
2423 <intent> ::= NCName [(param)]
2424
2425 <qualifiers> ::= "<qualifier>" | {"<qualifier>"[, "<qualifier>"]*}
2426
2427 <qualifier> ::= NCName | NCName/<qualifier>
2428
2429 <policySetAnnotation> ::= policySets(<policysets>)
2430
2431 <policySets> ::= "<policySetName>" | {"<policySetName>"[, "<policySetName>"]*}
2432
2433 <policySetName> ::= {aAnyURI}NCName
```

2434 *Snippet B-12: Annotation Grammar Additions for Policy Annotations*

- 2435 • anyURI is as defined by XML schema [\[XSD\]](#)[\[FXML\]](#)

2436 **B.5 Annotation Constants**

```
2437 <annotationConstant> ::= // @Define <identifier> <token string>
2438
2439 <identifier> ::= token
2440
2441 <token string> ::= "string" | "string"[ ## <token string>]
```

2442 *Snippet B-13: Annotation Constants Grammar*

- 2443 • Constants are immediately expanded

2444 C C++ WSDL Mapping Annotations

2445 To allow developers to control the mapping of C++ to WSDL, a set of annotations is defined. If WSDL
2446 mapping annotations are supported by an implementation, the annotations defined here MUST be
2447 supported and MUST be mapped to WSDL as described. [CPPC0001]

2448 C.1 Interface Header Annotations

2449 C.1.1 @WebService

2450 Annotation on a C++ class indicating that it represents a web service. An SCA implementation MUST
2451 treat any instance of a @Remotable annotation and without an explicit @WebService annotation as if a
2452 @WebService annotation with no parameters was specified. An SCA implementation MUST treat any
2453 instance of a @Remotable annotation and without an explicit @WebService annotation as if a
2454 @WebService annotation with no parameters was specified. [CPPC0002]

2455 **Corresponds to:** javax.jws.WebService annotation in the JAX-WS specification (7.11.1)

2456 **Format:**

```
2457 // @WebService(name="portTypeName", targetNamespace="namespaceURI",
2458 //                 serviceName="WSDLServiceName", portName="WSDLPortName")
```

2459 *Snippet C-1: @WebService Annotation Format*

2460 where:

- **name : NCName (0..1)** – specifies the name of the web service portType. The default is the name of the C++ class the annotation is applied to. The name of the associated binding is also determined by the portType. The binding name is the name of the portType suffixed with “Binding”.
- **targetNamespace : anyURI (0..1)** – specifies the target namespace for the web service. The default namespace is determined by the implementation.
- **serviceName : NCName (0..1)** – specifies the target name for the associated service. The default service name is the name of the C++ class suffixed with “Service”.
- **portName : NCName (0..1)** – specifies the name to be used for the associated WSDL port for the service. If portName is not specified, the name of the WSDL port is the name of the portType suffixed with “Port”. See [CPPF0042][CPPF0042]

2471 **Applies to:** Class

2472 Example:

2473 Input C++ source file:

```
2474 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2475 //                 serviceName="StockQuoteService")
2476 class StockQuoteService {
2477 };
```

2478

2479 Generated WSDL file:

```
2480 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2481   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2482   xmlns:tns="http://www.example.org/"
2483   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2484   targetNamespace="http://www.example.org/">
2485
2486   <portType name="StockQuote">
2487     <cpp:bindings>
2488       <cpp:class name="StockQuoteService"/>
```

```

2489     </cpp:bindings>
2490   </portType>
2491
2492   <binding name="StockQuoteServiceSoapBinding">
2493     <soap:binding style="document"
2494       transport="http://schemas.xmlsoap.org/soap/http"/>
2495   </binding>
2496
2497   <service name="StockQuoteService">
2498     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2499       <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2500     </port>
2501   </service>
2502 </definitions>

```

2503 *Snippet C-2: Example @WebService Annotation*

2504 C.1.2 @WebFunction

2505 Annotation on a C++ member function indicating that it represents a web service operation. An SCA
 2506 implementation MUST treat a member function annotated with an @Function annotation and without an
 2507 explicit @WebFunction annotation as if a @WebFunction annotation with with an operationName value
 2508 equal to the name value of the @Function annotation, an exclude value equal to the exclude value of the
 2509 @Operation annotation and no other parameters was specified. [CPPC0009]

2510 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2511 **Format:**

```

2512 // @WebFunction(operationName="operation",    action="SOAPAction",
2513 //                 exclude="false")

```

2514 *Snippet C-3: @WebFunction Annotation Format*

2515 where:

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

2522 **Applies to:** Member function.

2523 **Example:**

2524 Input C++ source file:

```

2525 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/")
2526 //           serviceName="StockQuoteService")
2527 class StockQuoteService {
2528
2529   // @WebFunction(operationName="GetLastTradePrice",
2530   //               action="urn:GetLastTradePrice")
2531   float getLastTradePrice(const std::string& tickerSymbol);
2532
2533   // @WebFunction(exclude=true)
2534   void setLastTradePrice(const std::string& tickerSymbol, float value);
2535 }

```

2536

2537 Generated WSDL file:

```

2538 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2539   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2540   xmlns:tns="http://www.example.org/"

```

```

2541     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2542     targetNamespace="http://www.example.org/">
2543
2544     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2545         xmlns:tns="http://www.example.org/"
2546         attributeFormDefault="unqualified"
2547         elementFormDefault="unqualified"
2548         targetNamespace="http://www.example.org/">
2549         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2550         <xs:element name="GetLastTradePriceResponse"
2551             type="tns:GetLastTradePriceResponse"/>
2552         <xs:complexType name="GetLastTradePrice">
2553             <xs:sequence>
2554                 <xs:element name="tickerSymbol" type="xs:string"/>
2555             </xs:sequence>
2556         </xs:complexType>
2557         <xs:complexType name="GetLastTradePriceResponse">
2558             <xs:sequence>
2559                 <xs:element name="return" type="xs:float"/>
2560             </xs:sequence>
2561         </xs:complexType>
2562     </xs:schema>
2563
2564     <message name="GetLastTradePrice">
2565         <part name="parameters" element="tns:GetLastTradePrice">
2566             </part>
2567     </message>
2568
2569     <message name="GetLastTradePriceResponse">
2570         <part name="parameters" element="tns:GetLastTradePriceResponse">
2571             </part>
2572     </message>
2573
2574     <portType name="StockQuote">
2575         <cpp:bindings>
2576             <cpp:class name="StockQuoteService"/>
2577         </cpp:bindings>
2578         <operation name="GetLastTradePrice">
2579             <cpp:bindings>
2580                 <cpp:memberFunction name="getLastTradePrice"/>
2581             </cpp:bindings>
2582             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2583                 </input>
2584             <output name="GetLastTradePriceResponse"
2585                 message="tns:GetLastTradePriceResponse">
2586                 </output>
2587             </operation>
2588         </portType>
2589
2590         <binding name="StockQuoteServiceSoapBinding">
2591             <soap:binding style="document"
2592                 transport="http://schemas.xmlsoap.org/soap/http"/>
2593             <operation name="GetLastTradePrice">
2594                 <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2595                 <input name="GetLastTradePrice">
2596                     <soap:body use="literal"/>
2597                 </input>
2598                 <output name="GetLastTradePriceResponse">
2599                     <soap:body use="literal"/>
2600                 </output>
2601             </operation>
2602         </binding>
2603
2604     <service name="StockQuoteService">

```

```

2605     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2606         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2607     </port>
2608 </service>
2609 </definitions>
```

2610 *Snippet C-4: Example @WebFunction Annotation*

2611 **C.1.3 @OneWay**

2612 Annotation on a C++ member function indicating that it represents a one-way request. The @OneWay
2613 annotation also affects the service interface, see @OneWay.

2614 **Corresponds to:** javax.jws.OneWay annotation in the JAX-WS specification (7.11.3)

2615 **Format:**

```

2616 // @OneWay
```

2617 *Snippet C-5: @OneWay Annotation Format*

2618 **Applies to:** Member function.

2619 Example:

2620 Input C++ source file:

```

2621 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
2622 //               serviceName="StockQuoteService")
2623 class StockQuoteService {
2624
2625     // @WebFunction(operationName="SetTradePrice",
2626     //               action="urn:SetTradePrice")
2627     // @OneWay
2628     void setTradePrice(const std::string& tickerSymbol, float price);
2629 }
```

2630

2631 Generated WSDL file:

```

2632 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2633   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2634   xmlns:tns="http://www.example.org/"
2635   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2636   targetNamespace="http://www.example.org/">
2637
2638   <xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema"
2639     xmlns:tns="http://www.example.org/"
2640     attributeFormDefault="unqualified"
2641     elementFormDefault="unqualified"
2642     targetNamespace="http://www.example.org/">
2643     <xss:element name="SetTradePrice" type="tns:SetTradePrice"/>
2644     <xss:complexType name="SetTradePrice">
2645       <xss:sequence>
2646         <xss:element name="tickerSymbol" type="xs:string"/>
2647         <xss:element name="price" type="xs:float"/>
2648       </xss:sequence>
2649     </xss:complexType>
2650   </xss:schema>
2651
2652   <message name="SetTradePrice">
2653     <part name="parameters" element="tns:SetTradePrice">
2654     </part>
2655   </message>
2656
2657   <portType name="StockQuote">
2658     <cpp:bindings>
2659       <cpp:class name="StockQuoteService"/>
```

```

2660     </cpp:bindings>
2661     <operation name="SetTradePrice">
2662         <cpp:bindings>
2663             <cpp:memberFunction name="setTradePrice"/>
2664         </cpp:bindings>
2665         <input name="SetTradePrice" message="tns:SetTradePrice">
2666             </input>
2667         </operation>
2668     </portType>
2669
2670     <binding name="StockQuoteServiceSoapBinding">
2671         <soap:binding style="document"
2672             transport="http://schemas.xmlsoap.org/soap/http"/>
2673         <operation name="SetTradePrice">
2674             <soap:operation soapAction="urn:SetTradePrice" style="document"/>
2675             <input name="SetTradePrice">
2676                 <soap:body use="literal"/>
2677             </input>
2678         </operation>
2679     </binding>
2680
2681     <service name="StockQuoteService">
2682         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2683             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2684         </port>
2685     </service>
2686 </definitions>
```

2687 *Snippet C-6: Example @OneWay Annotation*

2688 C.1.4 @WebParam

2689 Annotation on a C++ member function indicating the mapping of a parameter to the associated input and
2690 output WSDL messages.

2691 **Corresponds to:** javax.jws.WebParam annotation in the JAX-WS specification (7.11.4)

2692 **Format:**

```

2693 // @WebParam(paramName=<"parameter", name="WSDLElement",
2694 //             targetNamespace="namespaceURI", mode="IN" | "OUT" | "INOUT",
2695 //             header="false", partName="WSDLPart", type="xsdType")
```

2696 *Snippet C-7: @WebParam Annotation Format*

2697 where:

- **paramName : NCName (1..1)** – specifies the name of the parameter that this annotation applies to. 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. [CPPC0004]
- **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default value is the name of the parameter. If an @WebParam annotation is not present, and the parameter is unnamed, then a name of “argN”, where N is an incrementing value from 1 indicating the position of the parameter in the argument list, will be used.
- **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default namespace is the namespace of the associated @WebService. The targetNamespace attribute is ignored unless the binding style is document, and the binding parameterStyle is bare. See @SOAPBinding.
- **mode : token (0..1)** – specifies whether the parameter is associated with the input message, output message, or both. The default value is determined by the passing mechanism for the parameter, see Parameter and return type classification.
- **header : boolean (0..1)** – specifies whether this parameter is associated with a SOAP header element. The default value is “false”.

- 2714 • ***partName : NCName (0..1)*** – specifies the name of the WSDL part associated with this item. The
 2715 default value is the value of name.
- 2716 • ***type : NCName (0..1)*** – specifies the XML Schema type of the WSDL part or element associated with
 2717 this parameter. **The value of the type property of a @WebParam annotation MUST be one of the**
 2718 **simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema.****The value of the type**
 2719 **property of a @WebParam annotation MUST be one of the simpleTypes defined in namespace**
 2720 **http://www.w3.org/2001/XMLSchema.** [CPPC0005] The default type is determined by the mapping
 2721 defined in Simple Content Binding.

2722 **Applies to:** Member function parameter.

2723 Example:

2724 Input C++ source file:

```
2725 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/")
2726 //           serviceName="StockQuoteService")
2727 class StockQuoteService {
2728
2729     // @WebFunction(operationName="GetLastTradePrice",
2730     //               action="urn:GetLastTradePrice")
2731     // @WebParam(paramName="tickerSymbol", name="symbol")
2732     float getLastTradePrice(const std::string& tickerSymbol);
2733 }
```

2734 Generated WSDL file:

```
2735 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2736   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2737   xmlns:tns="http://www.example.org/"
2738   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2739   targetNamespace="http://www.example.org/">
2740
2741   <xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2742     xmlns:tns="http://www.example.org/"
2743     attributeFormDefault="unqualified"
2744     elementFormDefault="unqualified"
2745     targetNamespace="http://www.example.org/">
2746     <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2747     <xsd:element name="GetLastTradePriceResponse"
2748       type="tns:GetLastTradePriceResponse"/>
2749     <xsd:complexType name="GetLastTradePrice">
2750       <xsd:sequence>
2751         <xsd:element name="symbol" type="xsd:string"/>
2752       </xsd:sequence>
2753     </xsd:complexType>
2754     <xsd:complexType name="GetLastTradePriceResponse">
2755       <xsd:sequence>
2756         <xsd:element name="return" type="xsd:float"/>
2757       </xsd:sequence>
2758     </xsd:complexType>
2759   </xsd:schema>
2760
2761   <message name="GetLastTradePrice">
2762     <part name="parameters" element="tns:GetLastTradePrice">
2763       </part>
2764     </message>
2765
2766   <message name="GetLastTradePriceResponse">
2767     <part name="parameters" element="tns:GetLastTradePriceResponse">
2768       </part>
2769     </message>
2770
2771 <portType name="StockQuote">
```

```

2773     <cpp:bindings>
2774         <cpp:class name="StockQuoteService"/>
2775     </cpp:bindings>
2776     <operation name="GetLastTradePrice">
2777         <cpp:bindings>
2778             <cpp:memberFunction name="getLastTradePrice"/>
2779             <cpp:parameter name="tickerSymbol"
2780                 part="tns:GetLastTradePrice/parameter"
2781                 childElementName="symbol"/>
2782         </cpp:bindings>
2783         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2784             </input>
2785         <output name="GetLastTradePriceResponse"
2786             message="tns:GetLastTradePriceResponse">
2787             </output>
2788         </operation>
2789     </portType>
2790
2791     <binding name="StockQuoteServiceSoapBinding">
2792         <soap:binding style="document"
2793             transport="http://schemas.xmlsoap.org/soap/http"/>
2794         <operation name="GetLastTradePrice">
2795             <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2796             <input name="GetLastTradePrice">
2797                 <soap:body use="literal"/>
2798             </input>
2799             <output name="GetLastTradePriceResponse">
2800                 <soap:body use="literal"/>
2801             </output>
2802         </operation>
2803     </binding>
2804
2805     <service name="StockQuoteService">
2806         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2807             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2808         </port>
2809     </service>
2810 </definitions>

```

2811 *Snippet C-8: Example @WebParam Annotation*

2812 C.1.5 @WebResult

2813 Annotation on a C++ member function indicating the mapping of the member function's return type to the
2814 associated output WSDL message.

2815 **Corresponds to:** javax.jws.WebResult annotation in the JAX-WS specification (7.11.5)

2816 **Format:**

```

2817 // @WebResult(name=<="WSDLElement", targetNamespace="namespaceURI",
2818 //               header="false", partName="WSDLPart", type="xsdType")

```

2819 *Snippet C-9: @WebResult Annotation Format*

2820 where:

- **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default value is “return”.
- **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default namespace is the namespace of the associated @WebService. The targetNamespace attribute is ignored unless the binding style is document, and the binding parameterStyle is bare. See @SOAPBinding.

- 2827 • **header : boolean (0..1)** – specifies whether the result is associated with a SOAP header element.
 2828 The default value is “false”.
- 2829 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The
 2830 default value is the value of name.
- 2831 • **type : NCName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with
 2832 this parameter. The value of the type property of a @WebResult annotation MUST be one of the
2833 simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema.The value of the type
2834 property of a @WebResult annotation MUST be one of the simpleTypes defined in namespace
2835 http://www.w3.org/2001/XMLSchema. [CPPC0006] The default type is determined by the mapping
 2836 defined in Simple Content Binding.

2837 **Applies to:** Member function return value.

2838 Example:

2839 Input C++ source file:

```
2840 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
2841 //               serviceName="StockQuoteService")
2842 class StockQuoteService {
2843
2844     // @WebFunction(operationName="GetLastTradePrice",
2845     //               action="urn:GetLastTradePrice")
2846     // @WebResult(name="price")
2847     float getLastTradePrice(const std::string& tickerSymbol);
2848 }
```

2849

2850 Generated WSDL file:

```
2851 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2852   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2853   xmlns:tns="http://www.example.org/"
2854   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2855   targetNamespace="http://www.example.org/">
2856
2857   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2858     xmlns:tns="http://www.example.org/"
2859     attributeFormDefault="unqualified"
2860     elementFormDefault="unqualified"
2861     targetNamespace="http://www.example.org/">
2862     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2863     <xs:element name="GetLastTradePriceResponse"
2864       type="tns:GetLastTradePriceResponse"/>
2865     <xs:complexType name="GetLastTradePrice">
2866       <xs:sequence>
2867         <xs:element name="tickerSymbol" type="xs:string"/>
2868       </xs:sequence>
2869     </xs:complexType>
2870     <xs:complexType name="GetLastTradePriceResponse">
2871       <xs:sequence>
2872         <xs:element name="price" type="xs:float"/>
2873       </xs:sequence>
2874     </xs:complexType>
2875   </xs:schema>
2876
2877   <message name="GetLastTradePrice">
2878     <part name="parameters" element="tns:GetLastTradePrice">
2879       </part>
2880   </message>
2881
2882   <message name="GetLastTradePriceResponse">
2883     <part name="parameters" element="tns:GetLastTradePriceResponse">
2884       </part>
2885   </message>
```

```

2886
2887     <portType name="StockQuote">
2888         <cpp:bindings>
2889             <cpp:class name="StockQuoteService"/>
2890         </cpp:bindings>
2891         <operation name="GetLastTradePrice">
2892             <cpp:bindings>
2893                 <cpp:memberFunction name="getLastTradePrice"/>
2894             </cpp:bindings>
2895             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2896             </input>
2897             <output name="GetLastTradePriceResponse"
2898                 message="tns:GetLastTradePriceResponse">
2899             </output>
2900         </operation>
2901     </portType>
2902
2903     <binding name="StockQuoteServiceSoapBinding">
2904         <soap:binding style="document"
2905             transport="http://schemas.xmlsoap.org/soap/http"/>
2906         <operation name="GetLastTradePrice">
2907             <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2908             <input name="GetLastTradePrice">
2909                 <soap:body use="literal"/>
2910             </input>
2911             <output name="GetLastTradePriceResponse">
2912                 <soap:body use="literal"/>
2913             </output>
2914         </operation>
2915     </binding>
2916
2917     <service name="StockQuoteService">
2918         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2919             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2920         </port>
2921     </service>
2922 </definitions>

```

2923 *Snippet C-10: Example @WebResult Annotation*

2924 **C.1.6 @SOAPBinding**

2925 Annotation on a C++ member function indicating that it represents a web service operation.

2926 **Corresponds to:** javax.jws.SOAPBinding annotation in the JAX-WS specification (7.11.6)

2927 **Format:**

```

2928     // @SOAPBinding(style="DOCUMENT" | "RPC",  use="LITERAL" | "ENCODED",
2929     //           parameterStyle="BARE" | "WRAPPED")

```

2930 *Snippet C-11: @SOAPBinding Annotation Format*

2931 where:

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

2936 **Applies to:** Class, Member function.

2937 Example:

2938 Input C++ source file:

```
// @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
```

```

2940     //      serviceName="StockQuoteService")
2941     // @SOAPBinding(style="RPC")
2942     class StockQuoteService {
2943   };
2944
2945     Generated WSDL file:
2946 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2947   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2948   xmlns:tns="http://www.example.org/"
2949   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
2950   targetNamespace="http://www.example.org/">
2951
2952   <portType name="StockQuote">
2953     <cpp:bindings>
2954       <cpp:class name="StockQuoteService"/>
2955     </cpp:bindings>
2956   </portType>
2957
2958   <binding name="StockQuoteServiceSoapBinding">
2959     <soap:binding style="document"
2960       transport="http://schemas.xmlsoap.org/soap/http"/>
2961   </binding>
2962
2963   <service name="StockQuoteService">
2964     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2965       <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2966     </port>
2967   </service>
2968 </definitions>

```

2969 *Snippet C-12: Example @SOAPBinding Annotation*

2970 C.1.7 @WebFault

2971 Annotation on a C++ exception class indicating that it might be thrown as a fault by a web service
2972 function. A C++ class with a @WebFault annotation MUST provide a constructor that takes two
2973 parameters, a std::string and a type representing the fault information. Additionally, the class MUST
2974 provide a const member function “getFaultInfo” that takes no parameters, and returns the same type as
2975 defined in the constructor. [CPPC0007]

2976 **Corresponds to:** javax.xml.ws.WebFault annotation in the JAX-WS specification (7.2)

2977 **Format:**

```

2978 // @WebFault(name="WSDLElement", targetNamespace="namespaceURI")

```

2979 *Snippet C-13: @WebFault Annotation Format*

2980 where:

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

2984 **Applies to:** Class.

2985 **Example:**

2986 Input C++ source file:

```

2987 // @WebFault(name="UnknownSymbolFault",
2988 //           targetNamespace="http://www.example.org/")
2989 class UnknownSymbol {
2990   UnknownSymbol(const char* message,
2991                 const std::string& faultInfo);
2992

```

```

2993     std::string getFaultInfo() const;
2994 };
2995
2996 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
2997 //             serviceName="StockQuoteService")
2998 class StockQuoteService {
2999
3000     // @WebFunction(operationName="GetLastTradePrice",
3001     //               action="urn:GetLastTradePrice")
3002     // @WebThrows(faults="UnknownSymbol")
3003     float getLastTradePrice(const std::string& tickerSymbol);
3004 };

```

3005

3006 Generated WSDL file:

```

3007 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3008   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3009   xmlns:tns="http://www.example.org/"
3010   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3011   targetNamespace="http://www.example.org/">
3012
3013   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3014     xmlns:tns="http://www.example.org/"
3015     attributeFormDefault="unqualified"
3016     elementFormDefault="unqualified"
3017     targetNamespace="http://www.example.org/">
3018     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3019     <xs:element name="GetLastTradePriceResponse"
3020       type="tns:GetLastTradePriceResponse"/>
3021     <xs:complexType name="GetLastTradePrice">
3022       <xs:sequence>
3023         <xs:element name="tickerSymbol" type="xs:string"/>
3024       </xs:sequence>
3025     </xs:complexType>
3026     <xs:complexType name="GetLastTradePriceResponse">
3027       <xs:sequence>
3028         <xs:element name="return" type="xs:float"/>
3029       </xs:sequence>
3030     </xs:complexType>
3031     <xs:element name="UnknownSymbolFault" type="xs:string"/>
3032   </xs:schema>
3033
3034   <message name="GetLastTradePrice">
3035     <part name="parameters" element="tns:GetLastTradePrice">
3036     </part>
3037   </message>
3038
3039   <message name="GetLastTradePriceResponse">
3040     <part name="parameters" element="tns:GetLastTradePriceResponse">
3041     </part>
3042   </message>
3043
3044   <message name="UnknownSymbol">
3045     <part name="parameters" element="tns:UnknownSymbolFault">
3046     </part>
3047   </message>
3048
3049   <portType name="StockQuote">
3050     <cpp:bindings>
3051       <cpp:class name="StockQuoteService"/>
3052     </cpp:bindings>
3053     <operation name="GetLastTradePrice">
3054       <cpp:bindings>
3055         <cpp:memberFunction name="getLastTradePrice"/>

```

```

3056     </cpp:bindings>
3057     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3058     </input>
3059     <output name="GetLastTradePriceResponse"
3060             message="tns:GetLastTradePriceResponse">
3061     </output>
3062     <fault name="UnknownSymbol" message="tns:UnknownSymbol">
3063     </fault>
3064   </operation>
3065 </portType>
3066
3067 <binding name="StockQuoteServiceSoapBinding">
3068   <soap:binding style="document"
3069     transport="http://schemas.xmlsoap.org/soap/http"/>
3070   <operation name="GetLastTradePrice">
3071     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3072     <input name="GetLastTradePrice">
3073       <soap:body use="literal"/>
3074     </input>
3075     <output name="GetLastTradePriceResponse">
3076       <soap:body use="literal"/>
3077     </output>
3078     <fault>
3079       <soap:fault name="UnknownSymbol" use="literal"/>
3080     </fault>
3081   </operation>
3082 </binding>
3083
3084 <service name="StockQuoteService">
3085   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3086     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3087   </port>
3088 </service>
3089 </definitions>

```

3090 *Snippet C-14: Example @WebFault Annotation*

3091 **C.1.8 @WebThrows**

3092 Annotation on a C++ class indicating which faults might be thrown by this class.

3093 **Corresponds to:** No equivalent in JAX-WS.

3094 **Format:**

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

3096 *Snippet C-15: @WebThrows Annotation Format*

3097 where:

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

3101 **Applies to:** Member function.

3102 Example:

3103 See @WebFault.

3104

D WSDL C++ Mapping Extensions

3105

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

3109

[CPPD0001]

3111

D.1 <cpp:bindings>

3112

<cpp:bindings> is a container type which can be used as a WSDL extension. All other SCA wsdl extensions will be specified as children of a <cpp:bindings> element. A <cpp:bindings> element can be used as an extension to any WSDL type that accepts extensions.

3113

3114

D.2 <cpp:class>

3116 <cpp:class> provides a mechanism for defining an alternate C++ class name for a WSDL construct.

3117 **Format:**

3118

```
<cpp:class name="xsd:string"/>
```

3119

Snippet D-1: <cpp:class> Element Format

3120

where:

3121

- **class/@name : NCName (1..1)** – specifies the name of the C++ class associated with this WSDL element.

3123

Applicable WSDL element(s):

3124

- wsdl:portType
- wsdl:fault

3126

A <cpp:bindings/> element MUST NOT have more than one <cpp:class/> child element. [CPPD0002]

3127

Example:

3128

Input WSDL file:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
    xmlns:tns="http://www.example.org/"  
    xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"  
    targetNamespace="http://www.example.org/">  
  
    <portType name="StockQuote">  
        <cpp:bindings>  
            <cpp:class name="StockQuoteService"/>  
        </cpp:bindings>  
    </portType>  
</definitions>
```

3141

3142

Generated C++ file:

```
// @WebService(name="StockQuote", targetNamespace="http://www.example.org/"  
//             serviceName="StockQuoteService")  
class StockQuoteService {  
};
```

3147

Snippet D-2: Example <cpp:class> Element

3148 D.3 <cpp:enableWrapperStyle>

3149 <cpp:enableWrapperStyle> indicates whether or not the wrapper style for messages is applied, when
3150 otherwise applicable. If false, the wrapper style will never be applied.

3151 Format:

```
3152 <cpp:enableWrapperStyle>value</cpp:enableWrapperStyle>
```

3153 *Snippet D-3: <cpp:enableWrapperStyle> Element Format*

3154 where:

- 3155 • **enableWrapperStyle/text() : boolean (1..1)** – specifies whether wrapper style is enabled or disabled
3156 for this element and any of it's children. The default value is “true”.

3157 Applicable WSDL element(s):

- 3158 • wsdl:definitions
- 3159 • wsdl:portType – overrides a binding applied to wsdl:definitions
- 3160 • wsdl:portType/wsdl:operation – overrides a binding applied to wsdl:definitions or the enclosing
3161 wsdl:portType

3162 A <cpp:bindings/> element MUST NOT have more than one <cpp:enableWrapperStyle/> child element.
3163 [CPPD0003]

3164 Example:

3165 Input WSDL file:

```
3166 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3167   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3168   xmlns:tns="http://www.example.org/"  
3169   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"  
3170   targetNamespace="http://www.example.org/">  
3171  
3172   <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"  
3173     xmlns:tns="http://www.example.org/"  
3174     attributeFormDefault="unqualified"  
3175     elementFormDefault="unqualified"  
3176     targetNamespace="http://www.example.org/">  
3177     <xselement name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
3178     <xselement name="GetLastTradePriceResponse"  
3179       type="tns:GetLastTradePriceResponse"/>  
3180     <xsccomplexType name="GetLastTradePrice">  
3181       <xsssequence>  
3182         <xselement name="tickerSymbol" type="xs:string"/>  
3183       </xsssequence>  
3184     </xsccomplexType>  
3185     <xsccomplexType name="GetLastTradePriceResponse">  
3186       <xsssequence>  
3187         <xselement name="return" type="xs:float"/>  
3188       </xsssequence>  
3189     </xsccomplexType>  
3190   </xsschema>  
3191  
3192   <message name="GetLastTradePrice">  
3193     <part name="parameters" element="tns:GetLastTradePrice">  
3194     </part>  
3195   </message>  
3196  
3197   <message name="GetLastTradePriceResponse">  
3198     <part name="parameters" element="tns:GetLastTradePriceResponse">  
3199     </part>  
3200   </message>  
3201  
3202   <portType name="StockQuote">
```

```

3203     <cpp:bindings>
3204         <cpp:class name="StockQuoteService"/>
3205             <cpp:enableWrapperStyle>false</cpp:enableWrapperStyle>
3206         <cpp:bindings>
3207             <operation name="GetLastTradePrice">
3208                 <cpp:bindings>
3209                     <cpp:memberFunction name="getLastTradePrice"/>
3210                 </cpp:bindings>
3211                 <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3212                 </input>
3213                 <output name="GetLastTradePriceResponse"
3214                     message="tns:GetLastTradePriceResponse">
3215                 </output>
3216             </operation>
3217         </portType>
3218     </definitions>

```

3219

3220 Generated C++ file:

```

3221 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3222 //               serviceName="StockQuoteService")
3223 class StockQuoteService {
3224
3225     // @WebFunction(operationName="GetLastTradePrice",
3226     //               action="urn:GetLastTradePrice")
3227     commonj::sdo::DataObjectPtr
3228         getLastTradePrice(commonj::sdo::DataObjectPtr parameters);
3229 }

```

3230 *Snippet D-4: Example <cpp:enableWrapperStyle> Element*

3231 D.4 <cpp:namespace>

3232 <cpp:namespace> specifies the name of the C++ namespace that the associated WSDL element (and
3233 any of its children) are created in.

3234 **Format:**

```

3235     <cpp:namespace name="namespaceURI"/>

```

3236 *Snippet D-5: <cpp:namespace> Element Format*

3237 where:

- **namespace/@name : anyURI (1..1)** – specifies the name of the C++ namespace associated with
3239 this WSDL element.

3240 **Applicable WSDL element(s):**

- wsdl:definitions

3242 A <cpp:bindings/> element MUST NOT have more than one <cpp:namespace/> child element.
3243 [CPPD0004]

3244 Example:

3245 Input WSDL file:

```

3246 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3247     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3248     xmlns:tns="http://www.example.org/"
3249     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3250     targetNamespace="http://www.example.org/">
3251     <cpp:bindings>
3252         <cpp:namespace name="stock"/>
3253     </cpp:bindings>
3254
3255     <portType name="StockQuote">

```

```
3256     <cpp:bindings>
3257         <cpp:class name="StockQuoteService"/>
3258     </cpp:bindings>
3259 </portType>
3260 </definitions>
```

3261

3262 Generated C++ file:

```
3263 namespace stock
3264 {
3265     // @WebService(name="StockQuote", targetNamespace="http://www.example.org/")
3266     //     serviceName="StockQuoteService")
3267     // @WebService(name="StockQuote",
3268     class StockQuoteService {
3269     };
3270 }
```

3271 *Snippet D-6: Example <cpp:namespace> Element*

3272 D.5 <cpp:memberFunction>

3273 <cpp:memberFunction> specifies the name of the C++ member function that the associated WSDL
3274 operation is associated with.

3275 **Format:**

```
3276 <cpp:memberFunction name="myFunction"/>
```

3277 *Snippet D-7: <cpp:memberFunction> Element Format*

3278 where:

- **memberFunction/@name : NCName (1..1)** – specifies the name of the C++ member function associated with this WSDL operation.

3281 **Applicable WSDL element(s):**

- wsdl:portType/wsdl:operation

3283 A <cpp:bindings/> element MUST NOT have more than one <cpp:memberFunction/> child element.
3284 [CPPD0005]

3285 Example:

3286 Input WSDL file:

```
3287 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3288     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3289     xmlns:tns="http://www.example.org/"
3290     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3291     targetNamespace="http://www.example.org/">
3292
3293     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3294         xmlns:tns="http://www.example.org/"
3295         attributeFormDefault="unqualified"
3296         elementFormDefault="unqualified"
3297         targetNamespace="http://www.example.org/">
3298         <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3299         <xsd:element name="GetLastTradePriceResponse"
3300             type="tns:GetLastTradePriceResponse"/>
3301         <xsd:complexType name="GetLastTradePrice">
3302             <xsd:sequence>
3303                 <xsd:element name="tickerSymbol" type="xsd:string"/>
3304             </xsd:sequence>
3305         </xsd:complexType>
3306         <xsd:complexType name="GetLastTradePriceResponse">
3307             <xsd:sequence>
3308                 <xsd:element name="return" type="xsd:float"/>
```

```

3309      </xs:sequence>
3310    </xs:complexType>
3311  </xs:schema>
3312
3313  <message name="GetLastTradePrice">
3314    <part name="parameters" element="tns:GetLastTradePrice">
3315    </part>
3316  </message>
3317
3318  <message name="GetLastTradePriceResponse">
3319    <part name="parameters" element="tns:GetLastTradePriceResponse">
3320    </part>
3321  </message>
3322
3323  <portType name="StockQuote">
3324    <cpp:bindings>
3325      <cpp:class name="StockQuoteService"/>
3326    </cpp:bindings>
3327    <operation name="GetLastTradePrice">
3328      <cpp:bindings>
3329        <cpp:memberFunction name="getTradePrice"/>
3330      </cpp:bindings>
3331      <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3332      </input>
3333      <output name="GetLastTradePriceResponse"
3334        message="tns:GetLastTradePriceResponse">
3335      </output>
3336    </operation>
3337  </portType>
3338</definitions>
3339
3340  Generated C++ file:
3341 // @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3342 //           serviceName="StockQuoteService")
3343 class StockQuoteService {
3344
3345   // @WebFunction(operationName="GetLastTradePrice",
3346   //               action="urn:GetLastTradePrice")
3347   float getTradePrice(const std::string& tickerSymbol);
3348 }

```

3349 *Snippet D-8: Example <cpp:memberFunction> Element*

3350 D.6 <cpp:parameter>

3351 <cpp:parameter> specifies the name of the C++ member function parameter associated with a specific
 3352 WSDL message part or wrapper child element.

3353 **Format:**

```

3354 <cpp:parameter name="CPPParameter" part="WSDLPart"
3355   childElementName="WSDLElement" type="CPPType"/>

```

3356 *Snippet D-9: <cpp:parameter> Element Format*

3357 where:

- 3358 • **parameter/@name : NCName (1..1)** – specifies the name of the C++ member function parameter
 3359 associated with this WSDL operation. “return” is used to denote the return value.
- 3360 • **parameter/@part : string (1..1)** - an XPath expression identifying the wsdl:part of a wsdl:message.
- 3361 • **parameter/@childElementName : QName (1..1)** – specifies the qualified name of a child element of
 3362 the global element identified by parameter/@part.

- 3363 • ***parameter/@type : string (0..1)*** – specifies the type of the parameter or struct member or return
 3364 type. **The @type attribute of a <cpp:parameter/> element MUST be a C++ type specified in Simple**
 3365 **Content Binding.** **The @type attribute of a <cpp:parameter/> element MUST be a C++ type specified**
 3366 **in Simple Content Binding.** [CPPD0006] The default type is determined by the mapping defined in
 3367 Simple Content Binding.

3368 **Applicable WSDL element(s):**

- 3369 • wsdl:portType/wsdl:operation

3370 Example:

3371 Input WSDL file:

```

3372 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  

3373   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  

3374   xmlns:tns="http://www.example.org/"  

3375   xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"  

3376   targetNamespace="http://www.example.org/">  

3377  

3378   <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema"  

3379     xmlns:tns="http://www.example.org/"  

3380     attributeFormDefault="unqualified"  

3381     elementFormDefault="unqualified"  

3382     targetNamespace="http://www.example.org/">  

3383   <xselement name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  

3384   <xselement name="GetLastTradePriceResponse"  

3385     type="tns:GetLastTradePriceResponse"/>  

3386   <xsccomplexType name="GetLastTradePrice">  

3387     <xsssequence>  

3388       <xselement name="symbol" type="xs:string"/>  

3389     </xsssequence>  

3390   </xsccomplexType>  

3391   <xsccomplexType name="GetLastTradePriceResponse">  

3392     <xsssequence>  

3393       <xselement name="return" type="xs:float"/>  

3394     </xsssequence>  

3395   </xsccomplexType>  

3396 </xsschema>  

3397  

3398   <message name="GetLastTradePrice">  

3399     <part name="parameters" element="tns:GetLastTradePrice">  

3400     </part>  

3401   </message>  

3402  

3403   <message name="GetLastTradePriceResponse">  

3404     <part name="parameters" element="tns:GetLastTradePriceResponse">  

3405     </part>  

3406   </message>  

3407  

3408   <portType name="StockQuote">  

3409     <cpp:bindings>  

3410       <cpp:class name="StockQuoteService"/>  

3411     </cpp:bindings>  

3412     <operation name="GetLastTradePrice">  

3413       <cpp:bindings>  

3414         <cpp:memberFunction name="getLastTradePrice"/>  

3415         <cpp:parameter name="tickerSymbol"  

3416           part="tns:GetLastTradePrice/parameter"  

3417           childElementName="symbol"/>  

3418       </cpp:bindings>  

3419       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">  

3420       </input>  

3421       <output name="GetLastTradePriceResponse"  

3422         message="tns:GetLastTradePriceResponse">  

3423       </output>
  
```

```

3424    </operation>
3425  </portType>
3426
3427  <binding name="StockQuoteServiceSoapBinding">
3428    <soap:binding style="document"
3429      transport="http://schemas.xmlsoap.org/soap/http"/>
3430    <operation name="GetLastTradePrice">
3431      <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3432      <input name="GetLastTradePrice">
3433        <soap:body use="literal"/>
3434      </input>
3435      <output name="GetLastTradePriceResponse">
3436        <soap:body use="literal"/>
3437      </output>
3438    </operation>
3439  </binding>
3440
3441  <service name="StockQuoteService">
3442    <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3443      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3444    </port>
3445  </service>
3446</definitions>

```

3447

3448 Generated C++ file:

```

// @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
//             serviceName="StockQuoteService")
class StockQuoteService {

    // @WebFunction(operationName="GetLastTradePrice",
    //               action="urn:GetLastTradePrice")
    // @WebParam(paramName="tickerSymbol", name="symbol")
    float getLastTradePrice(const std::string& tickerSymbol);
}

```

3458 *Snippet D-10: Example <cpp:parameter> Element*

3459 D.7 JAX-WS WSDL Extensions

3460 An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL
3461 extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.
3462 Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted and their corresponding SCA
3463 WSDL extensions. An SCA implementation MAY support the reading and interpretation of JAX-WS
3464 defined WSDL extensions; however it MUST give precedence to the corresponding SCA WSDL
3465 extension if present. Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted and their
3466 corresponding SCA WSDL extensions. [CPPD0007]

3467

| 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 |

3468 *Table D-1: Allowed JAX-WS Extensions*

3469 **D.8 sca-wsdlext-cpp-1.1.xsd**

```
3470 <?xml version="1.0" encoding="UTF-8"?>
3471 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3472     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-
3473     cpp/cpp/200901"
3474     xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
3475     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3476     elementFormDefault="qualified">
3477
3478     <element name="bindings" type="cpp:BindingsType" />
3479     <complexType name="BindingsType">
3480         <choice minOccurs="0" maxOccurs="unbounded">
3481             <element ref="cpp:namespace" />
3482             <element ref="cpp:class" />
3483             <element ref="cpp:enableWrapperStyle" />
3484             <element ref="cpp:memberFunction" />
3485             <element ref="cpp:parameter" />
3486         </choice>
3487     </complexType>
3488
3489     <element name="namespace" type="cpp:NamespaceType" />
3490     <complexType name="NamespaceType">
3491         <attribute name="name" type="xsd:anyURI" use="required" />
3492     </complexType>
3493
3494     <element name="class" type="cpp:ClassType" />
3495     <complexType name="ClassType">
3496         <attribute name="name" type="xsd:NCName" use="required" />
3497     </complexType>
3498
3499     <element name="memberFunction" type="cpp:MemberFunctionType" />
3500     <complexType name="MemberFunctionType">
3501         <attribute name="name" type="xsd:NCName" use="required" />
3502     </complexType>
3503
3504     <element name="parameter" type="cpp:ParameterType" />
3505     <complexType name="ParameterType">
3506         <attribute name="part" type="xsd:string" use="required" />
3507         <attribute name="childElementName" type="xsd:QName"
3508             use="required" />
3509         <attribute name="name" type="xsd:NCName" use="required" />
3510         <attribute name="type" type="xsd:string" use="optional" />
3511     </complexType>
3512
3513         <element name="enableWrapperStyle" type="xsd:boolean" />
3514     </schema>
```

3515 *Snippet D-11: SCA C++ WSDL Extension Schema*

3516

E XML Schemas

3517

E.1 sca-interface-cpp-1.1.xsd

```

3518 <?xml version="1.0" encoding="UTF-8"?>
3519 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3520     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3521     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3522     elementFormDefault="qualified">
3523
3524     <include schemaLocation="sca-core.xsd"/>
3525
3526     <element name="interface.cpp" type="sca:CPPInterface"
3527         substitutionGroup="sca:interface"/>
3528
3529     <complexType name="CPPInterface">
3530         <complexContent>
3531             <extension base="sca:Interface">
3532                 <sequence>
3533                     <element name="function" type="sca:CPPFunction"
3534                         minOccurs="0" maxOccurs="unbounded" />
3535                     <element name="callbackFunction" type="sca:CPPFunction"
3536                         minOccurs="0" maxOccurs="unbounded" />
3537                     <any namespace="#other" processContents="lax"
3538                         minOccurs="0" maxOccurs="unbounded"/>
3539                 </sequence>
3540                 <attribute name="header" type="string" use="required"/>
3541                 <attribute name="class" type="Name" use="required"/>
3542                 <attribute name="callbackHeader" type="string" use="optional"/>
3543                 <attribute name="callbackClass" type="Name" use="optional"/>
3544             </extension>
3545         </complexContent>
3546     </complexType>
3547
3548     <complexType name="CPPFunction">
3549         <sequence>
3550             <choice minOccurs="0" maxOccurs="unbounded">
3551                 <element ref="sca:requires"/>
3552                 <element ref="sca:policySetAttachment"/>
3553             </choice>
3554             <any namespace="#other" processContents="lax" minOccurs="0"
3555                 maxOccurs="unbounded" />
3556         </sequence>
3557         <attribute name="name" type="NCName" use="required"/>
3558         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3559         <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3560         <attribute name="oneWay" type="boolean" use="optional"/>
3561         <attribute name="exclude" type="boolean" use="optional"/>
3562         <anyAttribute namespace="#other" processContents="lax"/>
3563     </complexType>
3564
3565 </schema>
```

3566

Snippet E-1: SCA <interface.cpp> Schema

3567

E.2 sca-implementation-cpp-1.1.xsd

```

3568 <?xml version="1.0" encoding="UTF-8"?>
3569 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3570     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
```

```

3571      xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3572      elementFormDefault="qualified">
3573
3574     <include schemaLocation="sca-core.xsd"/>
3575
3576     <element name="implementation.cpp" type="sca:CPPImplementation"
3577           substitutionGroup="sca:implementation" />
3578     <complexType name="CPPImplementation">
3579       <complexContent>
3580         <extension base="sca:Implementation">
3581           <sequence>
3582             <element name="function" type="sca:CPPImplementationFunction"
3583                 minOccurs="0" maxOccurs="unbounded" />
3584             <any namespace="#other" processContents="lax"
3585                 minOccurs="0" maxOccurs="unbounded"/>
3586           </sequence>
3587           <attribute name="library" type="NCName" use="required"/>
3588           <attribute name="header" type="NCName" use="required"/>
3589           <attribute name="path" type="string" use="optional"/>
3590           <attribute name="class" type="Name" use="optional"/>
3591           <attribute name="componentType" type="string" use="optional"/>
3592           <attribute name="scope" type="sca:CPPImplementationScope"
3593               use="optional"/>
3594           <attribute name="eagerInit" type="boolean" use="optional"/>
3595           <attribute name="allowsPassByReference" type="boolean"
3596               use="optional"/>
3597         </extension>
3598       </complexContent>
3599     </complexType>
3600
3601     <simpleType name="CPPImplementationScope">
3602       <restriction base="string">
3603         <enumeration value="stateless"/>
3604         <enumeration value="composite"/>
3605       </restriction>
3606     </simpleType>
3607
3608     <complexType name="CPPImplementationFunction">
3609       <sequence>
3610         <choice minOccurs="0" maxOccurs="unbounded">
3611           <element ref="sca:requires"/>
3612           <element ref="sca:policySetAttachment"/>
3613         </choice>
3614         <any namespace="#other" processContents="lax" minOccurs="0"
3615             maxOccurs="unbounded" />
3616       </sequence>
3617       <attribute name="name" type="NCName" use="required"/>
3618       <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3619       <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3620       <attribute name="allowsPassByReference" type="boolean"
3621           use="optional"/>
3622       <anyAttribute namespace="#other" processContents="lax"/>
3623     </complexType>
3624
3625   </schema>

```

3626 *Snippet E-2 : SCA <implementation.cpp> Schema*

3627 **E.3 sca-contribution-cpp-1.1.xsd**

```

3628   <?xml version="1.0" encoding="UTF-8"?>
3629   <schema xmlns="http://www.w3.org/2001/XMLSchema"
3630         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">

```

```
3631      xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3632      elementFormDefault="qualified">
3633
3634      <include schemaLocation="sca-contributions.xsd"/>
3635
3636      <element name="export.cpp" type="sca:CPPExport"
3637          substitutionGroup="sca:Export"/>
3638
3639      <complexType name="CPPExport">
3640          <complexContent>
3641              <attribute name="name" type="QName" use="required"/>
3642              <attribute name="path" type="string" use="optional"/>
3643          </complexContent>
3644      </complexType>
3645
3646      <element name="import.cpp" type="sca:CPPIImport"
3647          substitutionGroup="sca:Import"/>
3648
3649      <complexType name="CPPIImport">
3650          <complexContent>
3651              <attribute name="name" type="QName" use="required"/>
3652              <attribute name="location" type="string" use="required"/>
3653          </complexContent>
3654      </complexType>
3655
3656  </schema>
```

3657 *Snippet E-3: SCA <export.cpp> and <import.cpp> Schema*

3658

F Normative Statement Summary

3659

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; stateless and composite . 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 member function. |
| [CPP20013] | An SCA runtime MAY run multiple threads in a single composite scoped implementation instance object. |
| [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><interface.cpp></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><function></code> child elements of the <code><interface.cpp></code> element with <code>@exclude="true"</code> . |
| [CPP20017] | If the header file identified by the <code>@callbackHeader</code> attribute of an <code><interface.cpp></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><callbackFunction></code> child elements of the <code><interface.cpp></code> element with <code>@exclude="true"</code> . |
| [CPP20018][CPP2 0018] | An SCA runtime MUST NOT perform any synchronization of access to component implementations. |
| [CPP30001][CPP3 0001] | 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][CPP3 0002] | 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][CPP3 0003] | 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][CPP3 0004] | 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][CPP4 0001] | 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][CPP4 0002] | 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][CPP4 0003] | 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][CPP7 0001] | The <code>@name</code> attribute of a <code><export.cpp></code> element MUST be unique amongst the <code><export.cpp></code> elements in a domain. |
| [CPP70002][CPP7 0002] | The <code>@name</code> attribute of a <code><import.cpp></code> child element of a <code><contribution></code> MUST be unique amongst the <code><import.cpp></code> elements in of that contribution. |
| [CPP80001][CPP8 0001] | An SCA implementation MUST translate a class to tokens as part of conversion to WSDL or compatibility testing. |

| Conformance ID | Description |
|------------------------|--|
| [CPP80002] | <p>The return type and types of the parameters of a member function of a remutable service interface MUST be one of:</p> <ul style="list-style-type: none"> 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. 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. |
| [CPP80003][CPP80003] | <p>A C++ header file used to define an interface MUST declare at least one class with:</p> <ul style="list-style-type: none"> At least one public member function. All public member functions are pure virtual. |
| [CPP100001][CPP100001] | <p>A WSDL file might define a namespace using the <sca:namespace> WSDL extension, otherwise all C++ classes MUST be placed in a default namespace as determined by the implementation. Implementations SHOULD provide a mechanism for overriding the default namespace.</p> |
| [CPP100002][CPP100002] | <p>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.</p> |
| [CPP100003][CPP100003] | <ul style="list-style-type: none"> For unwrapped messages, an SCA implementation MUST map: <ul style="list-style-type: none"> in - the message part to a member function parameter, passed by const-reference. out - the message part to a member function parameter, passed by reference, or to the member function return type, returned by-value. in/out - the message part to a member function parameter, passed by reference. |
| [CPP100004][CPP100004] | <ul style="list-style-type: none"> For wrapped messages, an SCA implementation MUST map: <ul style="list-style-type: none"> in - the wrapper child to a member function parameter, passed by const-reference. out - the wrapper child to a member function parameter, passed by reference, or to the member function return type, returned by-value. in/out - the wrapper child to a member function parameter, passed by reference. |
| [CPP100005][CPP100005] | <p>An SCA implementation SHOULD provide a mechanism for overriding the default targetNamespace.</p> |
| [CPP100006][CPP100006] | <p>An SCA implementation MUST map a method's return type as an out parameter, a parameter passed by-reference or by-pointer as an in/out parameter, and all other parameters, including those passed by-const-reference as in parameters.</p> |
| [CPP100008] | <p>An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.</p> |

| Conformance ID | Description |
|----------------------|--|
| [CPP10009][CPP10009] | An SCA implementation MUST map a WSDL portType to a remotable C++ interface definition. |
| [CPP10010][CPP10010] | 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. |
| [CPP110001] | 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] | 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] | 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 . |
| [CPP110004] | 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 . |

3660 Table F-1: SCA C++ Core Normative Statements

3661 F.1 Annotation Normative Statement Summary

3662 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 a @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][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 @Remotable annotation and without an explicit @WebService annotation as if a @WebService annotation with no parameters was specified. |
| [CPPC0004] | 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. |
| [CPPC0005] | The value of the type property of a @WebParam annotation MUST be one of the simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema . |

| Conformance ID | Description |
|----------------|--|
| [CPPC0006] | The value of the type property of a @WebResult annotation MUST be one of the simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema . |
| [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. |
| [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 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. |

3663 *Table F-2: SCA C++ Annotation Normative Statements*

3664 F.2 WSDL Extention Normative Statement Summary

3665 This section contains a list of normative statements related to WSDL extensions for this specification.

| Conformance ID | Description |
|----------------------|--|
| [CPPD0001][CPPD0004] | 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][CPPD0002] | A <cpp:bindings/> element MUST NOT have more than one <cpp:class/> child element. |
| [CPPD0003][CPPD0003] | A <cpp:bindings/> element MUST NOT have more than one <cpp:enableWrapperStyle/> child element. |
| [CPPD0004][CPPD0004] | A <cpp:bindings/> element MUST NOT have more than one <cpp:namespace/> child element. |
| [CPPD0005][CPPD0005] | A <cpp:bindings/> element MUST NOT have more than one <cpp:memberFunction/> child element. |
| [CPPD0006][CPPD0006] | The @type attribute of a <cpp:parameter/> element MUST be a C++ type specified in Simple Content Binding. |
| [CPPD0007][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. Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted and their corresponding SCA WSDL extensions. |

3666 *Table F-3 SCA C++ WSDL Extension Normative Statements*

3667 **F.3 JAX-WS Normative Statements**

3668 The JAX-WS 2.1 specification [JAXWS21] defines normative statements for various requirements defined
 3669 by that specification. Table F-4 outlines those normative statements which apply to the WSDL mapping
 3670 described in this specification.

| Number | Conformance Point | Notes | Conformance ID |
|--------|---------------------------------------|--|----------------|
| 2.1 | WSDL 1.1 support | [A] | [CPPF0001] |
| 2.2 | Customization required | <p>[CPPD0001][CPPD0001]</p> <p>The reference to the JAX-WS binding language is treated as a reference to the C++ WSDL extensions defined in section WSDL C++ Mapping Extensions.</p> | |
| 2.3 | Annotations on generated classes | | [CPPF0002] |
| 2.4 | Definitions mapping | [CPP100001] [CPP100001] | |
| 2.5 | WSDL and XML Schema import directives | | [CPPF0003] |
| 2.6 | Optional WSDL extensions | | [CPPF0004] |
| 2.7 | SEI naming | | [CPPF0005] |
| 2.8 | javax.jws.WebService required | <p>[B]</p> <p>References to javax.jws.WebService in the conformance statement are treated as the C++ annotation @WebService.</p> | [CPPF0006] |
| 2.10 | Method naming | | [CPPF0007] |
| 2.11 | javax.jws.WebMethod required | <p>[A], [B]</p> <p>References to javax.jws.WebMethod in the conformance statement are treated as the C++ annotation @WebFunction.</p> | [CPPF0008] |
| 2.12 | Transmission primitive support | | [CPPF0009] |
| 2.13 | Using javax.jws.OneWay | <p>[A], [B]</p> <p>References to javax.jws.OneWay in the conformance statement are treated as the C++ annotation @OneWay.</p> | [CPPF0010] |
| 2.14 | Using javax.jws.SOAPBinding | <p>[A], [B]</p> <p>References to javax.jws.SOAPBinding in the conformance statement are treated as the C++ annotation @SOAPBinding.</p> | [CPPF0011] |
| 2.15 | Using javax.jws.WebParam | <p>[A], [B]</p> <p>References to javax.jws.WebParam in the conformance statement are treated as the C++ annotation @WebParam.</p> | [CPPF0012] |

| Number | Conformance Point | Notes | Conformance ID |
|--------|---------------------------------------|--|----------------|
| 2.16 | Using javax.jws.WebResult | [A], [B] References to javax.jws.WebResult in the conformance statement are treated as the C++ annotation @WebResult. | [CPPF0013] |
| 2.18 | Non-wrapped parameter naming | | [CPPF0014] |
| 2.19 | Default mapping mode | | [CPPF0015] |
| 2.20 | Disabling wrapper style | [B] References to jaxws:enableWrapperStyle in the conformance statement are treated as the WSDL extension cpp:enableWrapperStyle. | [CPPF0016] |
| 2.21 | Wrapped parameter naming | | [CPPF0017] |
| 2.22 | Parameter name clash | [A] | [CPPF0018] |
| 2.38 | javax.xml.ws.WebFault required | [B] References to javax.jws.WebFault in the conformance statement are treated as the C++ annotation @WebFault. | [CPPF0019] |
| 2.39 | Exception naming | | [CPPF0020] |
| 2.40 | Fault equivalence | [CPP100002] [CPP100002] | |
| 2.42 | Required WSDL extensions | MIME Binding not necessary | [CPPF0022] |
| 2.43 | Unbound message parts | [A] | [CPPF0023] |
| 2.44 | Duplicate headers in binding | | [CPPF0024] |
| 2.45 | Duplicate headers in message | | [CPPF0025] |
| 3.1 | WSDL 1.1 support | [A] | [CPPF0026] |
| 3.2 | Standard annotations | [A] [CPPC0001] [CPPC0001] | |
| 3.3 | Java identifier mapping | [A] | [CPPF0027] |
| 3.4 | Method name disambiguation | [A] References to javax.jws.WebMethod in the conformance statement are treated as the C++ annotation @WebFunction. | [CPPF0028] |
| 3.6 | WSDL and XML Schema import directives | | [CPPF0029] |
| 3.8 | portType naming | | [CPPF0030] |

| Number | Conformance Point | Notes | Conformance ID |
|--------|--|--|----------------|
| 3.9 | Inheritance flattening | [A] | [CPPF0044] |
| 3.10 | Inherited interface mapping | | [CPPF0045] |
| 3.11 | Operation naming | | [CPPF0031] |
| 3.12 | One-way mapping | [B] References to javax.jws.OneWay in the conformance statement are treated as the C++ annotation @OneWay. | [CPPF0032] |
| 3.13 | One-way mapping errors | | [CPPF0033] |
| 3.15 | Parameter classification | [CPP100006][CPP100006] | |
| 3.16 | Parameter naming | | [CPPF0035] |
| 3.17 | Result naming | | [CPPF0036] |
| 3.18 | 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] |
| 3.27 | Binding selection | References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY]. | [CPPF0039] |
| 3.28 | SOAP binding support | [A] | [CPPF0040] |
| 3.29 | SOAP binding style required | | [CPPF0041] |
| 3.31 | Port selection | | [CPPF0042] |
| 3.32 | Port binding | References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY]. | [CPPF0043] |

3671 [A] All references to Java in the conformance point are treated as references to C++.

3672 [B] Annotation generation is only necessary if annotations are supported by an SCA implementation.

3673 *Table F-4: JAX-WS Normative Statements that are Applicable to SCA C++*

3674 F.3.1 Ignored Normative Statements

| Number | Conformance Point |
|--------|------------------------------------|
| 2.9 | javax.xml.bind.XmlSeeAlso required |
| 2.17 | use of JAXB annotations |
| 2.23 | Using javax.xml.ws.RequestWrapper |
| 2.24 | Using javax.xml.ws.ResponseWrapper |

| Number | Conformance Point |
|---------------|---|
| 2.25 | Use of Holder |
| 2.26 | Asynchronous mapping required |
| 2.27 | Asynchronous mapping option |
| 2.28 | Asynchronous method naming |
| 2.29 | Asynchronous parameter naming |
| 2.30 | Failed method invocation |
| 2.31 | Response bean naming |
| 2.32 | Asynchronous fault reporting |
| 2.33 | Asynchronous fault cause |
| 2.34 | JAXB class mapping |
| 2.35 | JAXB customization use |
| 2.36 | JAXB customization clash |
| 2.37 | javax.xml.ws.wsaddressing.W3CEndpointReference |
| 2.41 | Fault Equivalence |
| 2.46 | Use of MIME type information |
| 2.47 | MIME type mismatch |
| 2.48 | MIME part identification |
| 2.49 | Service superclass required |
| 2.50 | Service class naming |
| 2.51 | javax.xml.ws.WebServiceClient required |
| 2.52 | Default constructor required |
| 2.53 | 2 argument constructor required |
| 2.54 | Failed getPort Method |
| 2.55 | javax.xml.ws.WebEndpoint required |
| 3.5 | Package name mapping |
| 3.7 | Class mapping |
| 3.14 | use of JAXB annotations |
| 3.19 | Default wrapper bean names |
| 3.20 | Default wrapper bean package |
| 3.21 | Null Values in rpc/literal |
| 3.24 | Exception naming |
| 3.25 | java.lang.RuntimeExceptions and java.rmi.RemoteExceptions |

| Number | Conformance Point |
|--------|-----------------------|
| 3.26 | Fault bean name clash |
| 3.30 | Service creation |

3675 *Table F-5: JAX-WS Normative Statements that Are Not Applicable to SCA C++*

3676

G Migration

3677

To aid migration of an implementation or clients using an implementation based the version of the Service Component Architecture for C++ defined in [OSOA SCA C++ Client and Implementation V1.00](#), this appendix identifies the relevant changes to APIs, annotations, or behavior defined in V1.00.

3678

3679

G.1 Method child elements of interface.cpp and implementation.cpp

3680

The `<method/>` child element of `<interface.cpp/>` and the `<method/>` child element of

`<implementation.cpp/>` have both been renamed to `<function/>` to be consistent with C++ terminology.

3683 **H Acknowledgements**

3684 The following individuals have participated in the creation of this specification and are gratefully
3685 acknowledged:

3686 **Participants:**

3687

| Participant Name | Affiliation |
|------------------------|--------------------|
| 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 |

3688 **I Revision History**

3689 [optional; should not be included in OASIS Standards]

3690

| Revision | Date | Editor | Changes Made |
|----------|------|--------|--------------|
| | | | • |

3691