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

Committee Draft 04 / Public Review Draft 02

21 January 2010

Specification URIs:
This Version:

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

Previous Version:

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

Latest Version:

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

Technical Committee:

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

Chair:

Bryan Aupperle, IBM

Editors:

Bryan Aupperle, IBM
David Haney
Pete Robbins, IBM

Related work:

This specification replaces or supercedes:

- OSOA SCA C++ Client and Implementation V1.00

This specification is related to:

- OASIS Service Component Architecture Assembly Model Version 1.1
- OASIS SCA Policy Framework Version 1.1
- OASIS Service Component Architecture Web Service Binding Specification Version 1.1

Declared XML Namespace(s):

http://docs.oasis-open.org/ns/opencsa/sca/~~200903~~200912
http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901

Abstract:

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

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

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

Status:

This document was last revised or approved by the Service Component Architecture / C and C++ TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/sca-c-cpp/.

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

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

Notices

Table of Contents

1 Introduction

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

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

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

1.1 Terminology

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

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

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

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

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

1.2 Normative References

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

[ASSEMBLY] OASIS Committee Draft ~~03~~05, *Service Component Architecture Assembly Model Specification Version 1.1*, ~~March 2009~~January 2010. http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-~~cd03~~cd05.pdf

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

[SDO21] OSOA, *Service Data Objects For C++ Specification*, December 2006. http://www.osoa.org/download/attachments/36/CPP-SDO-Spec-v2.1.0-FINAL.pdf

[WSDL11] World Wide Web Consortium, *Web Service Description Language (WSDL)*, March 2001. http://www.w3.org/TR/wsdl

[XSD] World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*, October 2004. http://www.w3.org/TR/xmlschema-2/

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

1.3 Conventions

1.3.1 Naming Conventions

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

- For the names of elements and the names of attributes within XSD files, the names follow the CamelCase convention, with all names starting with a lower case letter.

  e.g. <element name="componentType" type="sca:ComponentType"/>

- For the names of types within XSD files, the names follow the CamelCase convention with all names starting with an upper case letter

  e.g. <complexType name="ComponentService">

- For the names of intents, the names follow the CamelCase convention, with all names starting with a lower case letter, EXCEPT for cases where the intent represents an established acronym, in which case the entire name is in upper case.

  An example of an intent which is an acronym is the "SOAP" intent.

1.3.2 Typographic Conventions

This specification follows ~~some~~ typographic conventions for ~~some~~ specific constructs~~.~~:

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

2 Basic Component Implementation Model

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

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

2.1 Implementing a Service

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

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

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

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

~~The following snippets~~Snippet 2-1 – Snippet 2-3 show ~~the~~a C++ service interface and the C++ implementation class of a C++ implementation.

~~Service interface.~~

```
// LoanService interface
class LoanService {
public:
   virtual bool approveLoan(unsigned long customerNumber,
                            unsigned long loanAmount) = 0;
};
```

~~Implementation declaration header file.~~

Snippet 2-1: A C++ Service Interface

```
class LoanServiceImpl : public LoanService {
public:
   LoanServiceImpl();
   virtual ~LoanServiceImpl();

   virtual bool approveLoan(unsigned long customerNumber,
                            unsigned long loanAmount);
};
```

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

```
#include "LoanServiceImpl.h"

LoanServiceImpl::LoanServiceImpl()
{
   …
}

LoanServiceImpl::~LoanServiceImpl()
{
   …
}

bool LoanServiceImpl::approveLoan(unsigned long customerNumber,
                                  unsigned long loanAmount)
{
   …
}
```

Snippet 2-3: C++ Service Implementation

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

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

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

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

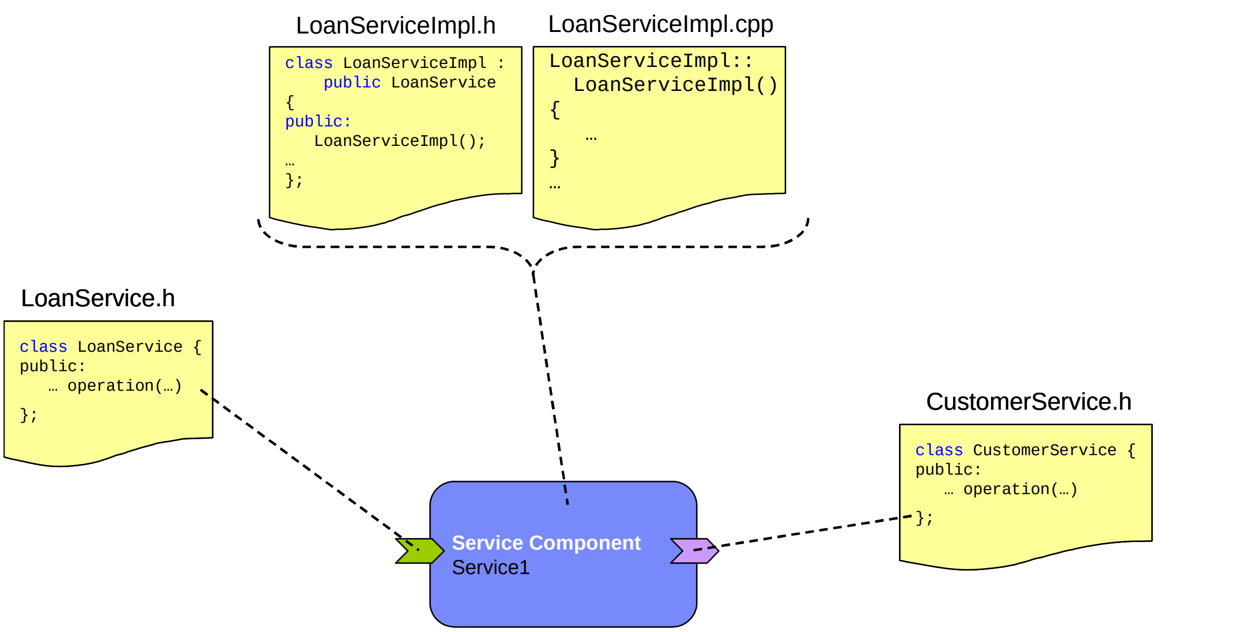

Figure 2-1: Relationship of C++ Implemenation Artifacts

2.1.1 Implementing a Remotable Service

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

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

Snippet 2-5: ComponentType for a Remotable Service

2.1.2 AllowsPassByReference

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

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

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

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

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

Marking a client as “allows pass by reference” asserts that for all reference’s member functions, the client ~~observe~~observes the ~~following~~ restrictions ~~for all references’ member functions~~:

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

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

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

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”. [CPP20015]

2.1.3 Implementing a Local Service

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

2.2 Component Implementation Scopes

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

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

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

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

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

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

~~1.1.1 Stateless scope~~

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

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

2.2.1 Stateless Scope

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

The concurrency model for the stateless scope is single threaded. 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. [CPP20012]

~~2.2.1~~2.2.2 Composite ~~scope~~Scope

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

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

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

2.3 Implementing a Configuration Property

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

~~The following code extract~~Snippet 2-7 shows how to get the property values.

```
#include "ComponentContext.h"
using namespace oasis::sca;

void clientFunction()
{
   …

   ComponentContextComponentContextPtr context = 
ComponentContext::getCurrent();

   DataObjectPtr properties = context.->getProperties();

   long loanRating = properties->getInteger(“maxLoanValue”);

   …
}
```

Snippet 2-7: Retrieving Property Values

2.4 Component Type and Component

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

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

~~The following snippets~~Snippet 2-8 – Snippet 2-10 show ~~the~~a C++ service interface and the C++ implementation class of a C++ service.

```
// LoanService interface
class LoanService {
public:
   virtual bool approveLoan(unsigned long customerNumber,
                            unsigned long loanAmount) = 0;
};
```

~~Implementation declaration header file.~~

Snippet 2-8: A C++ Service Interface

```
class LoanServiceImpl : public LoanService {
public:
   LoanServiceImpl();
   virtual ~LoanServiceImpl();

   virtual bool approveLoan(unsigned long customerNumber,
                            unsigned long loanAmount);
};
```

~~Implementation.~~

Snippet 2-9: C++ Service Implemenation Declaration

```
#include "LoanServiceImpl.h"

// Construction/Destruction

LoanServiceImpl::LoanServiceImpl()
{
   …
}
LoanServiceImpl::~LoanServiceImpl()
{
   …
}
// Implementation

bool LoanServiceImpl::approveLoan(unsigned long customerNumber,
                                  unsigned long loanAmount)
{
   …
}
```

~~The following snippet~~Snippet 2-10: C++ Service Implemenation

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

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

```
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912">
   <service name="LoanService">
      <interface.cpp header="LoanService.h"/>
   </service>
</componentType>
```

~~The following snippet~~Snippet 2-11: Component Type for Service Implementation in Snippet 2-10

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

```
<?xml version="1.0" encoding="ASCII"?>
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
      name="LoanComposite" >
   …

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

Snippet 2-12: Component Using Implemenation in Snippet 2-10

2.4.1 Interface.cpp

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

```
<?xml version="1.0" encoding="ASCII"?>
<!—- interface.cpp schema snippet -->
<interface.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
      header="string" class="Name"? remotable="boolean"?
      callbackHeader="string" callbackClass="Name"?
      requires="listOfQNames"? policySets="listOfQNames"? >

   <function … />*
   <callbackFunction … />*
   <requires/>*
   <policySetAttachment/>*

</interface.cpp>
```

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

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

- ***header : string (1..1) –*** full name of the header file that describes the interface, including relative path from the composite root.
- ***class : Name (0..1) –*** name of the class declaration for the interface in the header file, including any namespace definition. 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. [CPP20005]
- ***callbackHeader : string (0..1) –*** full name of the header file that describes the callback interface, including relative path from the composite root.
- ***callbackClass : Name (0..1) –*** name of the class declaration for the callback interface in the callback header file, including any namespace definition. 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. [CPP20006]

- ***remotable : boolean (0..1) –*** indicates whether the service is remotable or local. The default is local. See Implementing a Remotable Service
- ***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.
- ***policySets : listOfQNames (0..1)*** – a list of policy sets. See the Policy Framework specification **[POLICY]** for a description of this attribute.

The ***interface.cpp*** element has the ~~following~~ ***child elements***:

- ***function : CPPFunction (0..n) –*** see Function and CallbackFunction
- ***callbackFunction : CPPFunction (0..n) –*** see Function and CallbackFunction
- ***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.

2.4.2 Function and CallbackFunction

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

- ~~The following snippet~~If the header file identified by the *@header* attribute of an *<interface.cpp/>* 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 *<function/>* child elements of the *<interface.cpp/>* element with *@exclude="true"*. [CPP20016]
- If the header file identified by the *@callbackHeader* attribute of an *<interface.cpp/>* 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 *<callbackFunction/>* child elements of the *<interface.cpp/>* element with *@exclude="true"*. [CPP20017]

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

```
<?xml version="1.0" encoding="ASCII"?>
<!–- Function schema snippet -->
<interface.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
… >

   <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
         oneWay="Boolean"? />*exclude="Boolean"? >
      <requires/>*
      <policySetAttachment/>*
   </function> *

   <callbackFunction name="NCName" requires="listOfQNames"?
         policySets="listOfQNames"? oneWay="Boolean"? />exclude="Boolean"? >
```

```
      <requires/>*
      <policySetAttachment/>*
   </callbackFunction> *

</interface.cpp>
```

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

The ***function*** and ***callbackFunction*** elements have the ~~following~~ ***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/>*. [CPP20007]

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

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

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.

2.4.3 Implementation.cpp

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

```
<?xml version="1.0" encoding="ASCII"?>
<!–- implementation.cpp schema snippet -->
<implementation.cpp xmlns="http://docs.oasis-
open.org/ns/opencsa/sca/200903200912"
      library="NCName" path="string"? class="Name"
      scope="scope"? componentType="string"? allowsPassByReference="Boolean"?
      eagerInit="boolean"? >requires="listOfQNames"?

   <method …        policySets="listOfQNames"? >

   <function … />*
   <requires/>*
   <policySetAttachment/>*

</implementation.cpp>
```

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

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

- ***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.
- ***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.
- ***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. [CPP20009] The SCA runtime will append *.componentType* to the class name to find the componentType file.
- ***scope : CPPImplementationScope (0..1)*** **–** identifies the scope of the component implementation. The default is stateless. See Component Implementation Scopes
- ***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.
- ***allowsPassByReference : boolean (0..1)*** **–** indicates the implementation allows pass by reference data exchange semantics on calls to it or from it. These sematics apply to all services provided by and references used by an implementation. See AllowsPassByReference
- ***eagerInit : boolean (0..1)*** **–** indicates a composite scoped implementation is to be initialized when it is loaded. See Composite Scope

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

The ***implementation.cpp*** element has the ~~following~~ ***child elements***:

- ***function : CPPImplementationMethod (0..n)*** **–** see Implementation Function
- ***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.

2.4.4 Implementation Function

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

~~The following snippet~~Snippet 2-16 shows the *implementation.cpp* schema with the schema for a *method* child element:

```
<?xml version="1.0" encoding="ASCII"?>
<!–- ImplementationFunction schema snippet -->
<implementation.cpp xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912" … >

   <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
         allowsPassByReference="boolean"? />>
```

```
        <requires/>*
        <policySetAttachment/>*
    </function> *

</implementation.cpp>
```

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

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

- ***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/>*. [CPP20010]
- ***requires : listOfQNames (0..1)*** – a list of policy intents. See the Policy Framework specification **[POLICY]** ~~needed by~~for a description of this ~~member function~~attribute.
- ***policySets : listOfQNames (0..1)*** – a list of policy sets. See the Policy Framework specification **[POLICY]** for a description of this attribute.
- ***allowsPassByReference : boolean (0..1) –*** indicates the member function allows pass by reference data exchange semantics. See AllowsPassByReference

The ***function*** element has 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.

2.5 Instantiation

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

3 Basic Client Model

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

3.1 Accessing Services from Component Implementations

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

~~The following snippet~~Snippet 3-1 shows the `ComponentContext` C++ class with its `getService()` member function.

```
namespace oasis {
   namespace sca {

      class ComponentContext {
      public:
         static ComponentContextPtr getCurrent();
         virtual ServiceProxyPtr getService(
                               const std::string& referenceName) const = 0;
         …
      }
   }
}
```

Snippet 3-1: Partial ComponetContext Class Definition

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

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

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

using namespace oasis::sca;

void clientFunction()
{

   unsigned long customerNumber = 1234;

   ComponentContextPtr context = ComponentContext::getCurrent();

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

   if (customerService)
```

```
        short rating = customerService->getCreditRating(customerNumber);

}
```

Snippet 3-2: Using ComponentContext

3.2 Interface Proxies

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

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

```
class ServiceProxy {
public:
   //Possible future extensions
};
```

Snippet 3-3: ServiceProxy Class Definition

A remotable interface is always mappable to to WSDL, which can be mapped to C++ as described in WSDL to C++ and C++ to WSDL Mapping. The proxy class for a remotable interface is derived from `ServiceProxy` and contains the member functions mapped from the WSDL definition for the interface. 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. [CPP30001]

For the interface definition:

```
<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/">

   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tns="http://www.example.org/"
         attributeFormDefault="unqualified"
         elementFormDefault="unqualified"
         targetNamespace="http://www.example.org/">
      <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
      <xs:element name="GetLastTradePriceResponse"
            type="tns:GetLastTradePriceResponse"/>
      <xs:complexType name="GetLastTradePrice">
         <xs:sequence>
            <xs:element name="tickerSymbol" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLastTradePriceResponse">
         <xs:sequence>
            <xs:element name="return" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
   </xs:schema>

   <message name="GetLastTradePrice">
      <part name="parameters" element="tns:GetLastTradePrice">
```

```
         </part>
      </message>

      <message name="GetLastTradePriceResponse">
         <part name="parameters" element="tns:GetLastTradePriceResponse">
         </part>
      </message>

      <portType name="StockQuote">
         <cpp:bindings>
            <cpp:class name="StockQuoteService"/>
         </cpp:bindings>
         <operation name="GetLastTradePrice">
            <cpp:bindings>
               <cpp:memberFunction name="getTradePrice"/>
            </cpp:bindings>
            <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
            </input>
            <output name="GetLastTradePriceResponse"
                  message="tns:GetLastTradePriceResponse">
            </output>
         </operation>
      </portType>
   </definitions>
```

Snippet 3-4: Sample WSDL Interface

The resulting abstract proxy class is:

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

   // @WebFunction(operationName="GetLastTradePrice",
   //       action="urn:GetLastTradePrice")
   float getTradePrice(const std::string& tickerSymbol);
};
```

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

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

```
// LoanService interface
class LoanService {
public:
   virtual bool approveLoan(unsigned long customerNumber,
                            unsigned long loanAmount) = 0;
};
```

Snippet 3-6: Sample C++ Interface

The resulting proxy class is:

```
class LoanServiceProxy : public ServiceProxy {
public:
   virtual bool approveLoan(unsigned long customerNumber,
                            unsigned long loanAmount) = 0;
};
```

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

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

3.3 Accessing Services from non-SCA ~~component implementations~~Component Implementations

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

~~The following~~Snippet 3-8 shows a sample of how the `DomainContext` is used in non-SCA C++ code.

```
#include "DomainContext.h"
#include "CustomerServiceProxy.h"

void externalFunction()
{

   unsigned long customerNumber = 1234;

   DomainContextPtr context = myImplGetDomain("http://example.com/mydomain");

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

   if (customerService)
      short rating = customerService->getCreditRating(customerNumber);

}
```

Snippet 3-8: Using DomianContext

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

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

- componentName/serviceName/bindingName

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

3.4 Calling Service Operations

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

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

3.5 Long Running Request-Response Operations

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

For a service operation with signature

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

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

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

Snippet 3-9: AsynchronousInvocation Member Function Format

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

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

~~The following~~Snippet 3-10 shows a sample proxy class interface providing an asynchronous API.

```
using namespace oasis::sca;
using namespace commonj::sdo;


class CustomerServiceProxy : public ServiceProxy {
public:

   // synchronous member function
   virtual short getCreditRating(unsigned long customerNumber) = 0;

   // forward declare callback class
   class getCreditRatingCallback;

   // asynchronous response object
   class getCreditRatingResponse {
   public:
      // IOU/Future member functions
      virtual void cancel() = 0;
      virtual bool isCancelled() = 0;
      virtual bool isReady() = 0;
```

```
      virtual void setCallback(getCreditRatingCallbackPtr callback) = 0;

      virtual short getReturn() = 0;
   };

   // asynchronous callback object
   class getCreditRatingCallback {
   public:
     virtual void invoke(getCreditRatingResponsePtr) = 0;
   };

   // asynchronous member function
   virtual getCreditRatingResponsePtr getCreditRatingAsync(unsigned long
         customerNumber) = 0;
};
```

~~The following~~Snippet 3-10: Proxy with an Asynchronous API

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

```
#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);

     // ...
   }

}
```

Snippet 3-11: Using an Asynchronous API

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

3.5.1 Response Callback

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

```
#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()
{
   ComponentContextPtr context = ComponentContext::getCurrent();

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

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

      CustomerServiceProxy::getCreditRatingCallbackPtr callback =
            new CreditRatingCallback();

      response->setCallback(callback);

      // ...
   }
}
```

Snippet 3-12: Using a Response Object

3.5.2 Response Polling

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

```
#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) {
      CustomerServiceProxy::getCreditRatingResponsePtr response =
            customerService->getCreditRatingAsync(1234);

      while (!response->isReady()) {
         // do something else
      }

      // The response is ready and can be accessed without blocking.
```

```
      try {
         short rating = response->getReturn();
      }
      catch (...) {
         // ...
      }
   }
}
```

Snippet 3-13: Polling a Response Object

3.5.3 Synchronous Response Access

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

```
#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) {
      CustomerServiceProxy::getCreditRatingResponsePtr response =
            customerService->getCreditRatingAsync(1234);

      // The response is ready and can be accessed without blocking.
      try {
         short rating = response->getReturn();
      }
      catch (...) {
         // ...
      }
   }
}
```

Snippet 3-14: Blocking on a Response Object

3.5.4 Response Class

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

```
class <response_message_name>Response {
public:
   virtual <response_message_type> getReturn() const = 0;
   virtual void setCallback(<response_message_name>CallbackPtr callback) = 0;
   virtual boolean isReady() const = 0;
   virtual void cancel() const = 0;
   virtual boolean isCancelled() const = 0;
};
```

Snippet 3-15: AsynchronouseInvocation Response Class Definition

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 *@requires="asyncInvocation"* intent applied either to the operation of the reference as a whole. [CPP30004]

3.5.4.1 getReturn

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

Table 3-1: AsynchronousInvocation Response::getReturn Details

3.5.4.2 setCallback

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

Table 3-2: AsynchronousInvocation Response::setCallback Details

3.5.4.3 isReady

A C++ component implementation uses `isReady()` to determine if the response data for an polling 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	

Table 3-3: AsynchronousInvocation Response::isReady Details

3.5.4.4 cancel

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.

Table 3-4: AsynchronousInvocation Response::cancel Details

3.5.4.5 isCancelled

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

Table 3-5: AsynchronousInvocation Response::isCancelled Details

4 Asynchronous Programming

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

4.1 Non-blocking Calls

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

Any member function that returns `void`, has only by-value parameters and has no declared exceptions can be marked with the *@oneWay="true"* attribute in the interface definition of the service. An operation marked as oneWay 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. [CPP40001]

~~The following snippet~~Snippet 4-1 shows the component type for a service with the `reportEvent()` member fucntion declared as a one-way operation:

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

Snippet 4-1: ComponentType with oneWay Member Function

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

4.2 Callbacks

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

A callback interface is declared by the *@callbackHeader* and *@callbackClass* attributes in the interface definition of the service. ~~The following snippet~~Snippet 4-2 shows the component type for a service *MyService* with the interface defined in *MyService.h* and the interface for callbacks defined in *MyServiceCallback.h*,

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

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

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.

~~The following example shows~~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.

```
class Quotation {
public:
   virtual double requestQuotation(std::string productCode,
                                   unsigned int quantity) = 0;
};

class QuotationCallback {
public:
   virtual std::string getState() = 0;
   virtual std::string getZipCode() = 0;
   virtual std::string getCreditRating() = 0;
};
```

Snippet 4-3: C++ Interface with a Callback Interface

In ~~this example~~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.

~~The following code snippet~~Snippet 4-4 illustrates a possible implementation of the example service.

```
#include "QuotationImpl.h"
#include "QuotationCallbackProxy.h"
#include "ComponentContext.h"
using namespace oasis::sca;

double QuotationImpl::requestQuotation(std::string productCode,
                                        unsigned int quantity) {
   double price = getPrice(productQuote, quantity);
   double discount = 0;

   ComponentContextPtr context = ComponentContext::getCurrent();
   ServiceReferencePtr serviceRef = context->getSelfReference();
   ServiceProxyPtr callback = serviceRef->getCallback();
   QuotationCallbackQuotationCallbackProxyPtr quotationCallback =
```

```
        dynamicCast<QuotationCallbackProxy>(callback);

   if (quotationCallback) {
      if (quantity > 1000 && callback->getState().compare(“FL”) == 0)
         discount = 0.05;
      if (quantity > 10000 && callback->getCreditRating().data() == ‘A’)
         discount += 0.05;
   }
   return price * (1-discount);
}
```

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

~~The code snippet below~~Snippet 4-5 is taken from the client of this example service. The client’s service implementation class implements the member functions of the QuotationCallback interface as well as those of its own service interface ClientService.

```
#include "QuotationCallback.h"
#include "QuotationServiceProxy.h"
#include "ComponentContext.h"
using namespace oasis::sca;

void ClientImpl:: aClientFunction() {
   ComponentContextPtr context = ComponentContext::getCurrent();

   ServiceProxyPtr service = context->getService("quotationService");
   QuotationServiceProxyPtr quotationService =
         dynamicCast<QuotationServiceProxy>(service);

   if (quotationService)
      quotationService->requestQuotation(“AB123”, 2000);
}

std::string QuotationCallbackImpl::getState() {
   return “TX”;
}
std::string QuotationCallbackImpl::getZipCode() {
   return “78746”;
}
std::string QuotationCallbackImpl::getCreditRating() {
   return “AA”;
}
```

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

For each service of a component that includes a bidirectional interface, an SCA implementation MUST generate a service proxy derived from `ServiceProxy` that contains the operations of the reference’s callback interface definition. [CPP40002]

If a service of a component that has a callback interface contains operations with a *@requires=”asyncInvocation”* 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. [CPP40003]

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

4.2.2 Callback Instance Management

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

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

4.2.3 Implementing Multiple Bidirectional Interfaces

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

5 Error Handling

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

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

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

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

6 C++ API

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

6.1 Reference Counting Pointers

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

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

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

```
template <typename T>
class RefCountingPointer {
public:
   T& operator* () const;
   T* operator-> () const;
   operator void* () const;
   bool operator! () const;
};

template <typename T, typename U>
RefCountingPointer<T> constCast(RefCountingPointer<U> other);

template <typename T, typename U>
RefCountingPointer<T> dynamicCast(RefCountingPointer<U> other);

template <typename T, typename U>
RefCountingPointer<T> reinterpretCast(RefCountingPointer<U> other);

template <typename T, typename U>
RefCountingPointer<T> staticCast(RefCountingPointer<U> other);
```

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

Snippet 6-1: RefCountingPointer Class Definition

6.1.1 operator*

A C++ component implementation uses the `*` operator to dereferences the underlying pointer of a 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	`SCANullPointerException` if the pointer is NULL

Post Condition	No change

Table 6-1: RefCountingPointer operator* Details

6.1.2 operator->

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

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

Table 6-2: RefCountingPointer operator-> Details

6.1.3 operator void*

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

Table 6-3: RefCountingPointer operator void* Details

6.1.4 operator!

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

Table 6-4: RefCountingPointer operator! Details

6.1.5 constCast

The constCast global function provides the semantic behavior of the const_cast operator for 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

Table 6-5: constCast for RefCountingPointers Details

6.1.6 dynamicCast

The dynamicCast global function provides the semantic behavior of the dynamic_cast operator for use 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

Table 6-6: dynamicCast for RefCountingPointers Details

6.1.7 reinterpretCast

The reinterpretCast global function provides the semantic behavior of the reinterpret_cast operator for 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

Table 6-7: reinterpretCast for RefCountingPointers Details

6.1.8 staticCast

The staticCast global function provides the semantic behavior of the static_cast operator for 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

Table 6-8: staticCast for RefCountingPointers Details

6.2 Component Context

The ~~following shows the~~complete `ComponentContext` interface~~.~~ definition is:

```
class ComponentContext {
public:
   static ComponentContextPtr getCurrent();

   virtual std::string getURI() const = 0;

   virtual ServiceProxyPtr getService(
                            const std::string& referenceName) const = 0;
   virtual std::~~list~~vector<ServiceProxyPtr> getServices(
                            const std::string& referenceName) const = 0;

   virtual ServiceReferencePtr getServiceReference(
                               const std::string& referenceName) const = 0;
   virtual std::~~list~~vector<ServiceReferencePtr> getServiceReferences(
                               const std::string& referenceName) const = 0;
```

```
   virtual DataObjectPtr getProperties() const = 0;
   virtual DataFactoryPtr getDataFactory() const = 0;

   virtual ServiceReferencePtr getSelfReference() const = 0;
   virtual ServiceReferencePtr getSelfReference(
                                const std::string& serviceName)const = 0;
};
```

~~The **ComponentContext** C++ interface has the following member functions:~~

Snippet 6-2: ComponentContext Class Definition

6.2.1 getCurrent

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

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

Table 6-9: ComponentContext::getCurrent Details

6.2.2 getURI

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

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

Table 6-10: ComponentContext::getURI Details

6.2.3 getService

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

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

Table 6-11: ComponentContext::getService Details

6.2.4 getServices

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

Precondition	C++ component instance is running and has a `ComponentContext`	
Input Parameter	`referenceName`	Name of the Reference to get an interface object for
Return	~~List~~Vector of pointers to `ServiceProxy` objects implementing the interface of the Reference. This ~~list~~vector will be empty if `referenceName` is not defined for the component. Operations need to be invoked on each object in the ~~list.~~vector.	
Post Condition	`ServiceProxy` objects for the Reference are constructed. These `ServiceProxy` objects are independent of any ServiceReferences that are obtained for the Reference.	

Table 6-12: ComponentContext::getServices Details

6.2.5 getServiceReference

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

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

Table 6-13: ComponentContext::getServiceReference Details

6.2.6 getServiceReferences

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

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

Table 6-14: ComponentContext::getServiceReferences Details

6.2.7 getProperties

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

Precondition	C++ component instance is running and has a `ComponentContext`

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

Table 6-15: ComponentContext::getProperties Details

6.2.8 getDataFactory

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

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

Table 6-16: ComponentContext::getDataFactory Details

6.2.9 getSelfReference

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

There are two variations of this API.

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

and

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

Table 6-17: ComponentContext::getSelfReference Details

6.3 ServiceReference

The ~~following shows the~~ `ServiceReference` interface~~.~~ definition is:

```
class ServiceReference {
public:
   virtual ServiceProxyPtr getService() const = 0;

   virtual ServiceProxyPtr getCallback() const = 0;
};
```

~~The~~ Snippet 6-3: ServiceReference ~~interface has the following member functions (the~~ Class Definition

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

~~):~~

6.3.1 getService

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

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

Table 6-18: ServiceReference::getService Details

6.3.2 getCallback

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

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

Table 6-19: ServiceReference::getCallback Details

6.4 DomainContext

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

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

Snippet 6-4: DomainContext Class Definition

6.4.1 getService

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

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

Table 6-20: DomainContext::getService Details

6.5 SCAException

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

```
class SCAException : public std::exception {
public:
   const char* getEClassName() const;
   const char* getMessageText() const;
   const char* getFileName() const;
   unsigned long getLineNumber() const;
   const char* getFunctionName() const;
};
```

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

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

6.5.1 getEClassName

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	

Table 6-21: SCAException::getEClassName Details

6.5.2 getMessageText

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

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

Post Condition	No change

Table 6-22: SCAException::getMessageText Details

6.5.3 getFileName

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

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

Table 6-23: SCAException::getFileName Details

6.5.4 getLineNumber

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

Table 6-24: SCAException::getLineNumber Details

6.5.5 getFunctionName

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

Table 6-25: SCAException::getFunctionName Details

6.6 SCANullPointerException

The ~~following shows the~~ `SCANullPointerException` interface~~.~~ definition is:

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

Snippet 6-6: SCANullPointerException Class Definition

6.7 ServiceRuntimeException

The ~~following shows the~~ `ServiceRuntimeException` interface~~.~~ definition is:

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

Snippet 6-7: ServiceRuntimeException Class Definition

6.8 ServiceUnavailableException

The ~~following shows the~~ `ServiceUnavailableException` interface~~.~~ definition is:

```
class ServiceUnavailablException : public ServiceRuntimeException {
};
```

Snippet 6-8: ServiceUnavailableException Class Definition

6.9 MultipleServicesException

The ~~following shows the~~ `MultipleServicesException` interface~~.~~ definition is:

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

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 ~~some~~ additional considerations for executable and componentType files~~ discussed in the following sections~~.

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.

~~The following~~Snippet 7-1 shows a contribution containing a DLL.

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

Snippet 7-1: Contribution Containing a DLL

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

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

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

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 ~~the following snippet~~Snippet 7-3.

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

Snippet 7-3: Exporting a Contribution

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

```
META-INF/
   sca-contribution.xml
bin/
   rates.dll
RateUtilities/
   RateUtilities.composite
   RateUtilitiesService/
          RateUtils.h
          RateUtilsImpl.componentType
```

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

An export of the form:

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

Snippet 7-5: Exporting a Subdirectory of a Contribution

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

7.1.3 Executable outside of contribution (Import)

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

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

Snippet 7-6: Contribution with an Import

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

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

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

If the executable is located in the file system, the *@location* attribute identifies the location in the files system used as the root of the import as shown in ~~this snippet~~Snippet 7-8.

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

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

7.2 componentType files

As stated in ~~section 2.5~~Component Type and Component, each component implemented in C++ has a corresponding componentType file. This componentType file is, by default, located in the root directory of the composite containing the component or a subdirectory of the composite root with the name *<implementation class>.componentType*, as shown in ~~the following example~~Snippet 7-9.

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

Snippet 7-9: Contribution with ComponentType

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

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

Snippet 7-10: Component Definition with Local ComponentType

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

```
<component name="UnderwritingService">
   <implementation.cpp library="rates" path="rates:bin/"
         class="UnderwritingImpl” componentType="rates:types/UnderwritingImpl"
/>
</component>
```

Snippet 7-11: Component Definition with Imported ComponentType

7.3 C++ Contribution Extensions

7.3.1 Export.cpp

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

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

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

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

- ***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. [CPP70001]
- ***path : string (0..1)*** – path of the exported executable relative to the root of the contribution. If not present, the entire contribution is exported.

7.3.2 Import.cpp

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

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

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

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

- ***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. [CPP70002]
- ***location : string (1..1)*** – either the QName of a export or a file system location. If the value does not match an export name it is taken as an absolute file system path.

8 C++ Interfaces

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

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

~~7.4~~8.1 Types Supported in Service Interfaces

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

~~1.2 Local service~~

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

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

~~1.3 Remotable service~~

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

8.1.1 Local Service

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

8.1.2 Remotable Service

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

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

- 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. [CPP80002]

~~7.5~~8.2 ~~Unless the interface is marked as allowing pass by reference semantics, the behavior of the following are not defined:~~Header Files

- ~~Pointers.~~
- ~~References.~~

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

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

- A C++ header file used to define an interface MUST declare at least one class with:
  - At least one public member function.
  - All public member functions are pure virtual. ~~[CPP90001]~~

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

- ~~Macros~~
- ~~Inline member functions~~
- ~~Friend classes [CPP90002~~CPP80003]

~~8~~9 WSDL to C++ and C++ to WSDL Mapping

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

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

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

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

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

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

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

Major divergences from JAX-WS:

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

~~8.1~~9.1 Augmentations for WSDL to C++ Mapping

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

~~8.1.1~~9.1.1 Mapping WSDL targetNamespace to a C++ namespace

Since C++ does not define a standard convention for the use of namespaces, the SCA specification does not define an implicit mapping of WSDL targetNamespaces to C++ namespaces. 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. [CPP100001]

~~8.1.2~~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 wsd: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.

~~8.1.2.1~~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]

~~8.1.3~~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. [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. [CPP100004]

~~8.2~~9.2 Augmentations for C++ to WSDL Mapping

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

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. [CPP100010]

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

~~8.2.1~~9.2.1 Mapping C++ namespaces to WSDL namespaces

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

~~8.2.2~~9.2.2 Parameter and return type classification

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

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. [CPP100006]

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

~~8.2.3~~9.2.3 C++ to WSDL Type Conversion

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

~~8.2.4~~9.2.4 Service-specific Exceptions

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

~~8.3~~9.3 SDO Data Binding

~~8.3.1~~9.3.1 Simple Content Binding

The translation of XSD simple content types to C++ types follows the convention defined in the SDO specification. ~~The following table~~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

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

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

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

Table ~~2~~9-2: C++ type to XSD type mapping

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

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

~~8.3.2~~9.3.2 Complex Content Binding

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

~~9~~10 Conformance

The XML schema pointed to by the RDDL document at the SCA namespace URI, defined by the Assembly specification **[ASSEMBLY]** ~~[CPP110001]~~

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

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

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

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

[CPP110001]

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

[CPP110002]

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

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

~~9.1~~10.1 Conformance Targets

The conformance targets of this specification are:

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

~~9.2~~10.2 SCA Implementations

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

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

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

~~9.3~~10.3 SCA Documents

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

~~9.~~1. It MUST conform to the SCA Assembly Model Specification **[ASSEMBLY]** and, if appropriate, the SCA Policy Framework **[POLICY]**.

~~10.~~2. If it is a composite document, it MUST conform to the http://docs.oasis-open.org/opencsa/sca/~~200903~~200912/sca-interface-cpp-1.1.xsd and http://docs.oasis-open.org/opencsa/sca/~~200903~~200912/sca-implementation-cpp-1.1.xsd schema and MUST comply with the additional constraints on the document contents as defined in Table F-1.

If it is a componentType ~~or constrainingType~~ document, it MUST conforms to the http://docs.oasis-open.org/opencsa/sca/~~200903~~200912/sca-interface-cpp-1.1.xsd schema and MUST comply with the additional constraints on the document contents as defined in Table F-1.

If it is a contribution document, it MUST conforms to the http://docs.oasis-open.org/opencsa/sca/~~200903~~200912/sca-contribution-cpp-1.1.xsd schema and MUST comply with the additional constraints on the document contents as defined in Table F-1.

~~9.4~~10.4 C++ Files

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

1. It MUST comply with all statements in Table F-1, Table F-2 and Table F-4 related to C++ contents and annotations, notably all mandatory statements have to be satisfied.

~~9.5~~10.5 WSDL Files

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

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

A C++ SCA Annotations

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

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

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

Snippet A-1: Example Annotation

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

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

- Class

  The annotation immediately precedes the class.

  Example:

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

Snippet A-2: Example Class Annotation

- Member function

  The annotation immediately precedes the member function.

  Example:

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

Snippet A-3: Example Member Function Annotation

- Data Member

  The annotation immediately precedes the data member.

  Example:

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

Snippet A-4: Example Data Member Annotation

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

~~A.1~~A.2 Interface Header Annotations

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

~~A.1.1~~A.2.1 @Interface

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

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

Format:

```
// @Interface
```

Snippet A-5: @Interface Annotation Format

Applies to: `Class`

Example:

Interface header:

```
// @Interface
class LoanService {
...
};
```

Service definition:

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

Snippet A-6: Example of @Interface Annotation

~~A.1.2~~A.2.2 @Remotable

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

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

Format:

```
// @Remotable
```

Snippet A-7: @Remotable Annotation Format

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

Applies to: Class

Example:

Interface header:

```
// @Remotable
class LoanService {
...
};
```

Service definition:

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

Snippet A-8: Example of @Remotable Annotation

~~A.1.3~~A.2.3 @Callback

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

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

Format:

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

Snippet A-9: @Callback Annotation Format

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.

Applies to: `Class`

Example:

Interface header:

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

Service definition:

```
<service name="MyService">
   <interface.cpp header="MyService.h"
          callbackHeader="MyServiceCallback.h"
          callbackClass="MyServiceCallback"  />
</service>
```

Snippet A-10: Example of @Callback Annotation

~~A.1.4~~A.2.4 @OneWay

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

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

Format:

```
// @OneWay
```

Snippet A-11: @OneWay Annotation Format

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

Applies to: Member function

Example:

Interface header:

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

Service definition:

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

Snippet A-12: Example of @OneWay Annotation

A.2.5 @Function

Annotation on a interface member function to modify its representation in an SCA interface. 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 valued of the @Function annotation. [CPPA0004]

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

Format:

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

Snippet A-13: @Function Annotation Format

where

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

Applies to: Member function

Example:

Interface header:

```
class LoanService
{
public:
    …
// @Function(exclude="true")
   virtual void reportEvent(int eventId) = 0;
…
};
```

Service definition:

```
<service name="LoanService">
   <interface.cpp header="LoanService.h">
          <function name="reportEvent" exclude="true" />
   </interface.cpp>
</service>
```

Snippet A-14: Example of @Function Annotation

~~A.2~~A.3 Implementation Header Annotations

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

~~A.2.1~~A.3.1 @ComponentType

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

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

Format:

```
// @ComponentType
```

Snippet A-15: @ComponentType Annotation Format

Applies to: Class

Example:

Implementation header:

```
// @ComponentType
class LoanServiceImpl : public LoanService {
...
};
```

Component definition:

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

Snippet A-16: Example of @ComponentType Annotation

~~A.2.2~~A.3.2 @Scope

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

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

Format:

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

Snippet A-17: @Scope Annotation Format

where

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

Applies to: Class

Example:

Implementation header:

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

Component definition:

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

Snippet A-18: Example of @Scope Annotation

~~A.2.3~~A.3.3 @EagerInit

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

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

Format:

```
// @EagerInit
```

Snippet A-19: @EagerInit Annotation Format

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

Applies to: Class

Example:

Implementation header:

```
// @EagerInit
```

```
class LoanServiceImpl : public LoanService {
...
};
```

Component definition:

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

Snippet A-20: Example of @EagerInit Annotation

~~A.2.4~~A.3.4 @AllowsPassByReference

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

Corresponds to: *@allowsPassByReference=”true”* attribute of an *implementation.cpp* element or a *function* child element of an *implementation.cpp* element.

Format:

```
// @AllowsPassByReference
```

Snippet A-21: @AllowsPassByReference Annotation Format

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

Applies to: Class or Member function

Example:

Implementation header:

```
// @AllowsPassByReference
class LoanService {
...
};
```

Component definition:

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

Snippet A-22: Example of @AllowsPassByReference Annotation

~~A.2.5~~A.3.5 @Property

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

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

Format:

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

Snippet A-23: @Property Annotation Format

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*,

Applies to: DataMember

Example:

Implementation:

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

Component Type definition:

```
<componentType … >
   <service … />
   <property name="loanType" type="xsd:int" />
</componentType>
```

Snippet A-24: Example of @Property Annotation

~~A.2.6~~A.3.6 @Reference

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

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

Format:

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

Snippet A-25: @Reference Annotation Format

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.*

If the annotated data member is a std::~~list~~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.

Applies to: Data Member

Example:

Implementation:

```
// @Reference(interfaceHeader="LoanService.h" required="true")
LoanService*ProxyPtr loanService;

// @Reference(interfaceHeader="LoanService.h" required="false")
std::list<LoanService*vector<LoanServiceProxyPtr> loanServices;
```

Component Type definition:

```
<componentType … >
  <service … />
  <reference name="loanService" multiplicity="1..1">
        <interface.cpp header="LoanService.h" class="LoanService" />
  </reference>
  <reference name="loanServices" multiplicity="0..n">
        <interface.cpp header="LoanService.h" class="LoanService" />
  </reference>
</componentType>
```

Snippet A-26: Example of @Reference Annotation

~~A.3~~A.4 Base Annotation Grammar

```
<annotation> ::= // @<baseAnnotation>

<baseAnnotation> ::= <name> [(<params>)]

<params> ::= <paramNameValue>[, <paramNameValue>]* |
             <paramValue>[, <paramValue>]*

<paramNameValue> ::= <name>=”<value>”

<paramValue> ::= “<value>”

<name> ::= NCName

<value> ::= string
```

Snippet A-27: Base Annotation Grammar

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

B C++ SCA Policy Annotations

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

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

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

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

B.1 General Intent Annotations

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

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

Each intent is expressed as a string. Intents are XML QNames, which consist of a Namespace URI followed by the name of the Intent. The precise form used is~~ as follows:~~:

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

Snippet B-1: Intent Format

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

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

```
// @Define SCA_PREFIX "{http://docs.oasis-open.org/ns/opencsa/sca/~~200903~~200912}"
// @Define CONFIDENTIALITY SCA_PREFIX ## "confidentiality"
// @Define CONFIDENTIALITY_MESSAGE CONFIDENTIALITY ## ".message"
```

Snippet B-2: Example Intent Constants

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

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

Corresponds to: *@requires* attribute of ~~a *service, reference, operation*~~an *interface.cpp, inplemenation.cpp, function* or ~~*property*~~*callbackFunction* element.

Format:

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

where

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

Snippet B-3: @Requires Annotation Format

Applies to: Class, Member Function

Examples:

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

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

Snippet B-4: Example @Requires Annotation

A reference requiring support for confidentiality:

```
class Foo {
   …
// @Requires(CONFIDENTIALITY)
// @Reference(interfaceHeader="SetBar.h")
   void setBar(Bar* bar);
   …
}
```

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

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

```
class Foo {
   …
// @Requires(SCA_PREFIX "confidentiality ")
// @Reference(interfaceHeader="SetBar.h")
```

```
    void setBar(Bar* bar);
    …
}
```

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

B.2 Specific Intent Annotations

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

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

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

```
// @Integrity
```

Snippet B-7: Example Specific Intent Annotation

Corresponds to: *@requires="<Intent>"* attribute of ~~*a service, reference, operation*~~an *interface.cpp, implementation,cpp, function* or ~~*property*~~*callbackFunction* element.

Format:

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

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

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

Snippet B-8: @<Intent> Annotation Format

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

Example:

```
// @ClientAuthentication( {“message”, “transport”} )
```

Snippet B-9: Example @<Intent> Annotation

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

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

B.2.1 Security Interaction

Intent	Annotation
~~authentication~~clientAuthentication	@ClientAuthentication
serverAuthentication	@ServerAuthentication
mutualAuthentication	@MutualAuthentication
confidentiality	@Confidentiality
integrity	@Integrity

Table B-1: Security Interaction Intent Annotations

These three intents can be qualified with

- transport
- message

B.2.2 Security Implementation

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

~~In addition to allow roles to defined, an SCA runtime MAY use the following annotation~~
~~@DeclareRoles(<comma separated list of roles>")~~

Intent	**Annotation**	**Qualifiers**
authorization	@Authorization	fine_grain

Table B-2: Security Implementation Intent Annotations

B.2.3 Reliable Messaging

Intent	Annotation
atLeastOnce	@AtLeastOnce
atMostOnce	@AtMostOnce
~~Ordered~~ordered	@Ordered
exactlyOnce	@ExactlyOnce

Table B-3: Reliable Messaging Intent Annotations

B.2.4 Transactions

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

Table B-4: Transaction Intent Annotations

B.2.5 Miscellaneous

Intent	Annotation	Qualifiers
SOAP	@SOAP	v1_1~~_1~~ v1_2
~~JMS~~	~~@JMS~~	

~~A.2 Application of Intent Annotations~~

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

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

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

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

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

~~A.3 Inheritance and Intent Annotations~~

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

```
// @Remotable
// @Integrity("transport")
```

```
// @Authentication
class HelloService {
public:
// @Integrity
// @Authentication("message")
   wchar_t* hello(wchar_t* message) {...}

// @Integrity
// @Authentication("transport")
   wchar_t* helloThere() {...}
}

// @Remotable
// @Confidentiality("message")
class HelloChildService : public HelloService {
public:
// @Confidentiality("transport")
   wchar_t* hello(wchar_t* message) {...}
// @Authentication
   wchar_t* helloWorld(){...}
}
```

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

- ~~The effective intent annotation on the helloWorld member function is @Integrity("transport"), @Authentication, and @Confidentiality("message").~~
- ~~The effective intent annotation on the hello member function of the HelloChildService is @Integrity("transport"), @Authentication, and @Confidentiality("transport"),~~
- ~~The effective intent annotation on the helloThere member function of the HelloChildService is @Integrity and @Authentication("transport"), the same as in HelloService class.~~
- ~~The effective intent annotation on the hello member function of the HelloService is @Integrity and @Authentication("message")~~

~~The listing below contains the equivalent declarative security interaction policy of the HelloService and HelloChildService implementation corresponding to the C++ classes shown in Example 1a.~~

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

<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
                name="HelloServiceComposite" >
   ...

   <component name="HelloServiceComponent">
         <service name="HelloService" requires="integrity/transport
                authentication">
                …
         </service>
         <implementation.cpp library="HelloService.dll"
                class="HellowServiceImpl">
                <function name="hello" requires="integrity
                      authentication/message"/>
                <function name="helloThere" requires="integrity
                      authentication/transport"/>
         </implementation.cpp>
   </component>
   <component name="HelloChildServiceComponent">
         <service name="HelloChildService" requires="integrity/transport
                authentication confidentiality/message">
         …
```

```
            </service>
            <implementation.cpp library="HelloChildService.dll"
                   class="HelloChildServiceImpl">
                   <function name="hello" requires="confidentiality/transport"/>
                   <function name="helloThere" requires="integrity/transport
                          authentication"/>
                   <function name="helloWorld" requires="authentication"/>
            </implementation.cpp>
    </component>

    ...

</composite>
```

~~Example 1b. Declaratives intents equivalent to annotated intents in Example 1a.~~

~~A.4 Relationship of Declarative and Annotated Intents~~

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

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

Table B-5: Miscellaneous Intent Annotations

B.3 Policy Set Annotations

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

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

Corresponds to: *@policySets* attribute of ~~a *service, reference, operation*~~an *interface.cpp, implementation.cpp, function* or ~~*property*~~*callbackFuncion* element.

Format:

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

Snippet B-10: @PolicySets Annotation Format

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

Applies to: Class, Member Function,

Example:

```
// @Reference(name="helloService", interfaceHeader="helloService.h",
//              required="true")
// @PolicySets({ MY_NS “WS_Encryption_Policy",
//                 MY_NS "WS_Authentication_Policy"})
   HelloService* helloService;
```

```
…
```

Snippet B-11: Example @PolicySets Annotation

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

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

B.4 Policy Annotation Grammar Additions

```
<annotation> ::= // @<baseAnnotation> | @<requiresAnnotation> |
                     @<intentAnnotation> | @<policySetAnnotation>

<requiresAnnotation> ::= Requires(<intents>)

<intents> ::= “<qualifiedIntent>” |
              {“<qualifiedIntent>”[, “<qualifiedIntent>”]*})

<qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |
                      <intentName>.<qualifier>.qualifier>

<intentName> ::= {aAnyURI}NCName

<intentAnnotation> ::= <intent>[(<qualifiers>)]

<intent> ::= NCName [(param)]

<qualifiers> ::= “<qualifier>” | {“<qualifier>”[, “<qualifier>”]*}

<qualifier> ::= NCName | NCName/<qualifier>

<policySetAnnotation> ::= policySets(<policysets>)

<policySets> ::= “<policySetName>” | {“<policySetName>”[, “<policySetName>”]*}

<policySetName> ::= {aAnyURI}NCName
```

Snippet B-12: Annotation Grammar Additions for Policy Annotations

- anyURI is as defined by XML schema **[XSD]**

B.5 Annotation Constants

```
<annotationConstant> ::= // @Define <identifier> <token string>

<identifier> ::= token

<token string> ::= “string” | “string”[ ## <token string>]
```

Snippet B-13: Annotation Constants Grammar

- Constants are immediately expanded

C C++ WSDL Mapping Annotations

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

C.1 Interface Header Annotations

C.1.1 @WebService

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

[CPPC0002]

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

Format:

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

Snippet C-1: @WebService Annotation Format

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". ~~The~~
- ***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 ~~associated binding is also determined by the serviceName. In the case of a SOAP binding, the binding name~~WSDL port is the name of the ~~service suffixed~~portType suffiexed with "~~SoapBinding".~~Port". See [CPPF0042]
- ~~***portName : NCName (0..1) –*** specifies the name to be used for the associated WSDL port for the service. If a @WebService does not have a **portName** element, an SCA implementation MUST use the value associated with the **name** element, suffixed with *"Port"*. [CPPC0003]~~

Applies to: Class

Example:

Input C++ source file:

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

```
class StockQuoteService {
};
```

Generated 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>

   <binding name="StockQuoteServiceSoapBinding">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
   </binding>

   <service name="StockQuoteService">
      <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
      </port>
   </service>
</definitions>
```

Snippet C-2: Example @WebService Annotation

C.1.2 @WebFunction

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

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

Format:

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

Snippet C-3: @WebFunction Annotation Format

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"*.

Applies to: Member function.

Example:

Input C++ source file:

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

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

   // @WebFunction(exclude=true)
   void setLastTradePrice(const std::string& tickerSymbol, float value);
};
```

Generated 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/">

   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tns="http://www.example.org/"
         attributeFormDefault="unqualified"
         elementFormDefault="unqualified"
         targetNamespace="http://www.example.org/">
      <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
      <xs:element name="GetLastTradePriceResponse"
            type="tns:GetLastTradePriceResponse"/>
      <xs:complexType name="GetLastTradePrice">
         <xs:sequence>
            <xs:element name="tickerSymbol" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLastTradePriceResponse">
         <xs:sequence>
            <xs:element name="return" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
   </xs:schema>

   <message name="GetLastTradePrice">
      <part name="parameters" element="tns:GetLastTradePrice">
      </part>
   </message>

   <message name="GetLastTradePriceResponse">
      <part name="parameters" element="tns:GetLastTradePriceResponse">
      </part>
   </message>

   <portType name="StockQuote">
      <cpp:bindings>
         <cpp:class name="StockQuoteService"/>
      </cpp:bindings>
      <operation name="GetLastTradePrice">
         <cpp:bindings>
            <cpp:memberFunction name="getLastTradePrice"/>
         </cpp:bindings>
         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
         </input>
         <output name="GetLastTradePriceResponse"
```

```
                 message="tns:GetLastTradePriceResponse">
         </output>
      </operation>
   </portType>

   <binding name="StockQuoteServiceSoapBinding">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="GetLastTradePrice">
         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
         <wsdl:input name="GetLastTradePrice">
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="GetLastTradePriceResponse">
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </binding>

   <service name="StockQuoteService">
      <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
      </port>
   </service>
</definitions>
```

Snippet C-4: Example @WebFunction Annotation

C.1.3 @OneWay

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

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

Format:

```
// @OneWay
```

Snippet C-5: @OneWay Annotation Format

Applies to: Member function.

Example:

Input C++ source file:

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

   // @WebFunction(operationName="SetTradePrice",
   //       action="urn:SetTradePrice")
   // @OneWay
   void setTradePrice(const std::string& tickerSymbol, float price);
};
```

Generated 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/">

   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tns="http://www.example.org/"
         attributeFormDefault="unqualified"
         elementFormDefault="unqualified"
         targetNamespace="http://www.example.org/">
      <xs:element name="SetTradePrice" type="tns:SetTradePrice"/>
      <xs:complexType name="SetTradePrice">
         <xs:sequence>
            <xs:element name="tickerSymbol" type="xs:string"/>
            <xs:element name="price" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
   </xs:schema>

   <message name="SetTradePrice">
      <part name="parameters" element="tns:SetTradePrice">
      </part>
   </message>

   <portType name="StockQuote">
      <cpp:bindings>
         <cpp:class name="StockQuoteService"/>
      </cpp:bindings>
      <operation name="SetTradePrice">
         <cpp:bindings>
            <cpp:memberFunction name="setTradePrice"/>
         </cpp:bindings>
         <input name="SetTradePrice" message="tns:SetTradePrice">
         </input>
      </operation>
   </portType>

   <binding name="StockQuoteServiceSoapBinding">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="SetTradePrice">
         <soap:operation soapAction="urn:SetTradePrice" style="document"/>
         <wsdl:input name="SetTradePrice">
            <soap:body use="literal"/>
         </wsdl:input>
      </wsdl:operation>
   </binding>

   <service name="StockQuoteService">
      <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
      </port>
   </service>
</definitions>
```

Snippet C-6: Example @OneWay Annotation

C.1.4 @WebParam

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

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

Format:

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

Snippet C-7: @WebParam Annotation Format

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 he 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*".
- ***partName : NCName (0..1) –*** specifies the name of the WSDL part associated with this item. The default value is the value of name.
- ***type : NCName (0..1) –*** specifies the XML Schema type of the WSDL part or element associated with this parameter. 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. [CPPC0005] The default type is determined by the mapping defined in Simple Content Binding.

Applies to: Member function parameter.

Example:

Input C++ source 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);
};
```

Generated 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/">

   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
         xmlns:tns="http://www.example.org/"
         attributeFormDefault="unqualified"
         elementFormDefault="unqualified"
         targetNamespace="http://www.example.org/">
      <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
      <xs:element name="GetLastTradePriceResponse"
            type="tns:GetLastTradePriceResponse"/>
      <xs:complexType name="GetLastTradePrice">
         <xs:sequence>
            <xs:element name="symbol" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLastTradePriceResponse">
         <xs:sequence>
            <xs:element name="return" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
   </xs:schema>

   <message name="GetLastTradePrice">
      <part name="parameters" element="tns:GetLastTradePrice">
      </part>
   </message>

   <message name="GetLastTradePriceResponse">
      <part name="parameters" element="tns:GetLastTradePriceResponse">
      </part>
   </message>

   <portType name="StockQuote">
      <cpp:bindings>
         <cpp:class name="StockQuoteService"/>
      </cpp:bindings>
      <operation name="GetLastTradePrice">
         <cpp:bindings>
            <cpp:memberFunction name="getLastTradePrice"/>
            <cpp:parameter name="tickerSymbol"
                  part="tns:GetLastTradePrice/parameter"
                  childElementName="symbol"/>
         </cpp:bindings>
         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
         </input>
         <output name="GetLastTradePriceResponse"
               message="tns:GetLastTradePriceResponse">
         </output>
      </operation>
   </portType>

   <binding name="StockQuoteServiceSoapBinding">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="GetLastTradePrice">
         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
         <wsdl:input name="GetLastTradePrice">
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="GetLastTradePriceResponse">
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </binding>

   <service name="StockQuoteService">
      <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
```

```
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
    </service>
</definitions>
```

Snippet C-8: Example @WebParam Annotation

C.1.5 @WebResult

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

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

Format:

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

Snippet C-9: @WebResult Annotation Format

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.
- ***header : boolean (0..1) –*** specifies whether the result is associated with a SOAP header element. The default value is "*false*".
- ***partName : NCName (0..1) –*** specifies the name of the WSDL part associated with this item. The default value is the value of name.
- ***type : NCName (0..1) –*** specifies the XML Schema type of the WSDL part or element associated with this parameter. 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. [CPPC0006] The default type is determined by the mapping defined in Simple Content Binding.

Applies to: Member function return value.

Example:

Input C++ source file:

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

   // @WebFunction(operationName="GetLastTradePrice",
   //        action="urn:GetLastTradePrice")
   // @WebResult(name="price")
   float getLastTradePrice(const std::string& tickerSymbol);
};
```

Generated 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/">

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://www.example.org/"
      attributeFormDefault="unqualified"
      elementFormDefault="unqualified"
      targetNamespace="http://www.example.org/">
   <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
   <xs:element name="GetLastTradePriceResponse"
         type="tns:GetLastTradePriceResponse"/>
   <xs:complexType name="GetLastTradePrice">
      <xs:sequence>
         <xs:element name="tickerSymbol" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="GetLastTradePriceResponse">
      <xs:sequence>
         <xs:element name="price" type="xs:float"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>

<message name="GetLastTradePrice">
   <part name="parameters" element="tns:GetLastTradePrice">
   </part>
</message>

<message name="GetLastTradePriceResponse">
   <part name="parameters" element="tns:GetLastTradePriceResponse">
   </part>
</message>

<portType name="StockQuote">
   <cpp:bindings>
      <cpp:class name="StockQuoteService"/>
   </cpp:bindings>
   <operation name="GetLastTradePrice">
      <cpp:bindings>
         <cpp:memberFunction name="getLastTradePrice"/>
      </cpp:bindings>
      <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
      </input>
      <output name="GetLastTradePriceResponse"
            message="tns:GetLastTradePriceResponse">
      </output>
   </operation>
</portType>

<binding name="StockQuoteServiceSoapBinding">
   <soap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="GetLastTradePrice">
      <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
      <wsdl:input name="GetLastTradePrice">
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="GetLastTradePriceResponse">
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
</binding>
```

```
    <service name="StockQuoteService">
       <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
          <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
       </port>
    </service>
</definitions>
```

Snippet C-10: Example @WebResult Annotation

C.1.6 @SOAPBinding

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

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

Format:

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

Snippet C-11: @SOAPBinding Annotation Format

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*".

Applies to: Class, Member function.

Example:

Input C++ source file:

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

Generated 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>

   <binding name="StockQuoteServiceSoapBinding">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
   </binding>

   <service name="StockQuoteService">
```

```
      <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
      </port>
   </service>
</definitions>
```

Snippet C-12: Example @SOAPBinding Annotation

C.1.7 @WebFault

Annotation on a C++ exception class indicating that it might be thrown as a fault by a web service function. 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. [CPPC0007]

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

Format:

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

Snippet C-13: @WebFault Annotation Format

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.

Applies to: Class.

Example:

Input C++ source file:

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

   std:string getFaultInfo() const;
};

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

   // @WebFunction(operationName="GetLastTradePrice",
   //       action="urn:GetLastTradePrice")
   // @WebThrows(faults="UnknownSymbol")
   float getLastTradePrice(const std::string& tickerSymbol);
};
```

Generated 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/">

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://www.example.org/"
      attributeFormDefault="unqualified"
      elementFormDefault="unqualified"
      targetNamespace="http://www.example.org/">
   <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
   <xs:element name="GetLastTradePriceResponse"
         type="tns:GetLastTradePriceResponse"/>
   <xs:complexType name="GetLastTradePrice">
      <xs:sequence>
         <xs:element name="tickerSymbol" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="GetLastTradePriceResponse">
      <xs:sequence>
         <xs:element name="return" type="xs:float"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="UnknownSymbolFault" type="xs:string"/>
</xs:schema>

<message name="GetLastTradePrice">
   <part name="parameters" element="tns:GetLastTradePrice">
   </part>
</message>

<message name="GetLastTradePriceResponse">
   <part name="parameters" element="tns:GetLastTradePriceResponse">
   </part>
</message>

<message name="UnknownSymbol">
   <part name="parameters" element="tns:UnknownSymbolFault">
   </part>
</message>

<portType name="StockQuote">
   <cpp:bindings>
      <cpp:class name="StockQuoteService"/>
   </cpp:bindings>
   <operation name="GetLastTradePrice">
      <cpp:bindings>
         <cpp:memberFunction name="getLastTradePrice"/>
      </cpp:bindings>
      <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
      </input>
      <output name="GetLastTradePriceResponse"
            message="tns:GetLastTradePriceResponse">
      </output>
      <fault name="UnknownSymbol" message="tns:UnknownSymbol">
      </fault>
   </operation>
</portType>

<binding name="StockQuoteServiceSoapBinding">
   <soap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="GetLastTradePrice">
      <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
      <wsdl:input name="GetLastTradePrice">
         <soap:body use="literal"/>
      </wsdl:input>
```

```
         <wsdl:output name="GetLastTradePriceResponse">
            <soap:body use="literal"/>
         </wsdl:output>
         <wsdl:fault>
            <soap:fault name="UnknownSymbol" use="literal"/>
         </wsdl:fault>
      </wsdl:operation>
   </binding>

   <service name="StockQuoteService">
      <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
      </port>
   </service>
</definitions>
```

Snippet C-14: Example @WebFault Annotation

C.1.8 @WebThrows

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

Corresponds to: No equivalent in JAX-WS.

Format:

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

Snippet C-15: @WebThrows Annotation Format

where:

- ***faults : NMTOKEN (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]

Applies to: Member function.

Example:

See ~~@WebFault~~@WebFault.

D WSDL C++ Mapping Extensions

~~The following~~ 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. [CPPD0001]

D.1 <cpp:bindings>

<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.

D.2 <cpp:class>

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

Format:

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

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

where:

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

Applicable WSDL element(s):

- wsdl:portType
- wsdl:fault

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

Example:

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>
```

Generated C++ file:

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

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

D.3 <cpp:enableWrapperStyle>

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

Format:

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

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

where:

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

Applicable WSDL element(s):

- wsdl:definitions
- wsdl:portType – overrides a binding applied to wsdl:definitions
- wsdl:portType/wsdl:operation – overrides a binding applied to wsdl:definitions or the enclosing wsdl:portType

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

Example:

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/">

   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tns="http://www.example.org/"
         attributeFormDefault="unqualified"
         elementFormDefault="unqualified"
         targetNamespace="http://www.example.org/">
      <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
      <xs:element name="GetLastTradePriceResponse"
            type="tns:GetLastTradePriceResponse"/>
      <xs:complexType name="GetLastTradePrice">
         <xs:sequence>
            <xs:element name="tickerSymbol" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLastTradePriceResponse">
         <xs:sequence>
            <xs:element name="return" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
```

```
   </xs:schema>

   <message name="GetLastTradePrice">
      <part name="parameters" element="tns:GetLastTradePrice">
      </part>
   </message>

   <message name="GetLastTradePriceResponse">
      <part name="parameters" element="tns:GetLastTradePriceResponse">
      </part>
   </message>

   <portType name="StockQuote">
      <cpp:bindings>
         <cpp:class name="StockQuoteService"/>
         <cpp:enableWrapperStyle>false</cpp:enableWrapperStyle>
      <cpp:bindings>
      <operation name="GetLastTradePrice">
         <cpp:bindings>
            <cpp:memberFunction name="getLastTradePrice"/>
         </cpp:bindings>
         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
         </input>
         <output name="GetLastTradePriceResponse"
               message="tns:GetLastTradePriceResponse">
         </output>
      </operation>
   </portType>
</definitions>
```

Generated C++ file:

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

   // @WebFunction(operationName="GetLastTradePrice",
   //       action="urn:GetLastTradePrice")
   commonj::sdo::DataObjectPtr
         getLastTradePrice(commonj::sdo::DataObjectPtr parameters);
};
```

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

D.4 <cpp:namespace>

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

Format:

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

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

where:

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

Applicable WSDL element(s):

- wsdl:definitions

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

Example:

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/">
   <cpp:bindings>
      <cpp:namespace name="stock"/>
   </cpp:bindings>

   <portType name="StockQuote">
      <cpp:bindings>
         <cpp:class name="StockQuoteService"/>
      </cpp:bindings>
   </portType>
</definitions>
```

Generated C++ file:

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

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

D.5 <cpp:memberFunction>

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

Format:

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

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

where:

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

Applicable WSDL element(s):

- wsdl:portType/wsdl:operation

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

Example:

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/">

   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tns="http://www.example.org/"
         attributeFormDefault="unqualified"
         elementFormDefault="unqualified"
         targetNamespace="http://www.example.org/">
      <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
      <xs:element name="GetLastTradePriceResponse"
            type="tns:GetLastTradePriceResponse"/>
      <xs:complexType name="GetLastTradePrice">
         <xs:sequence>
            <xs:element name="tickerSymbol" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLastTradePriceResponse">
         <xs:sequence>
            <xs:element name="return" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
   </xs:schema>

   <message name="GetLastTradePrice">
      <part name="parameters" element="tns:GetLastTradePrice">
      </part>
   </message>

   <message name="GetLastTradePriceResponse">
      <part name="parameters" element="tns:GetLastTradePriceResponse">
      </part>
   </message>

   <portType name="StockQuote">
      <cpp:bindings>
         <cpp:class name="StockQuoteService"/>
      </cpp:bindings>
      <operation name="GetLastTradePrice">
         <cpp:bindings>
            <cpp:memberFunction name="getTradePrice"/>
         </cpp:bindings>
         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
         </input>
         <output name="GetLastTradePriceResponse"
               message="tns:GetLastTradePriceResponse">
         </output>
      </operation>
   </portType>
</definitions>
```

Generated C++ file:

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

   // @WebFunction(operationName="GetLastTradePrice",
   //       action="urn:GetLastTradePrice")
```

```
    float getTradePrice(const std::string& tickerSymbol);
};
```

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

~~A.5~~D.6 <cpp:parameter>

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

Format:

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

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

where:

- ***parameter/@name : NCName (1..1)*** – specifies the name of the C++ member function parameter associated with this WSDL operation. "*return*" is used to denote the return value.
- ***parameter/@part : string (1..1)*** - an XPath expression identifying the wsdl:part of a wsdl:message.
- ***parameter/@childElementName : QName (1..1)*** – specifies the qualified name of a child element of the global element identified by parameter/@part.
- ***parameter/@type : ~~NCName~~string (0..1) –*** specifies the type of the parameter or struct member or return type. The @type attribute of a <cpp:parameter/> element MUST be a C++ type specified in Simple Content Binding. [CPPD0006] The default type is determined by the mapping defined in Simple Content Binding.

Applicable WSDL element(s):

- wsdl:portType/wsdl:operation

Example:

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/">

   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:tns="http://www.example.org/"
         attributeFormDefault="unqualified"
         elementFormDefault="unqualified"
         targetNamespace="http://www.example.org/">
      <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
      <xs:element name="GetLastTradePriceResponse"
            type="tns:GetLastTradePriceResponse"/>
      <xs:complexType name="GetLastTradePrice">
         <xs:sequence>
            <xs:element name="symbol" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetLastTradePriceResponse">
         <xs:sequence>
            <xs:element name="return" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
```

```
    </xs:schema>

    <message name="GetLastTradePrice">
       <part name="parameters" element="tns:GetLastTradePrice">
       </part>
    </message>

    <message name="GetLastTradePriceResponse">
       <part name="parameters" element="tns:GetLastTradePriceResponse">
       </part>
    </message>

    <portType name="StockQuote">
       <cpp:bindings>
          <cpp:class name="StockQuoteService"/>
       </cpp:bindings>
       <operation name="GetLastTradePrice">
          <cpp:bindings>
             <cpp:memberFunction name="getLastTradePrice"/>
             <cpp:parameter name="tickerSymbol"
                   part="tns:GetLastTradePrice/parameter"
                   childElementName="symbol"/>
          </cpp:bindings>
          <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
          </input>
          <output name="GetLastTradePriceResponse"
                message="tns:GetLastTradePriceResponse">
          </output>
       </operation>
    </portType>

    <binding name="StockQuoteServiceSoapBinding">
       <soap:binding style="document"
             transport="http://schemas.xmlsoap.org/soap/http"/>
       <wsdl:operation name="GetLastTradePrice">
          <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
          <wsdl:input name="GetLastTradePrice">
             <soap:body use="literal"/>
          </wsdl:input>
          <wsdl:output name="GetLastTradePriceResponse">
             <soap:body use="literal"/>
          </wsdl:output>
       </wsdl:operation>
    </binding>

    <service name="StockQuoteService">
       <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
          <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
       </port>
    </service>
</definitions>
```

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);
};
```

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

~~D.6~~D.7 JAX-WS WSDL Extensions

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.

[CPPD0007] ~~The following is a list of JAX-WS WSDL extensions that MAY be recognized, and their corresponding SCA WSDL extension.~~

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

~~A.6 WSDL Extensions Schema~~

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

Table D-1: Allowed JAX-WS Extensions

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

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-
cpp/cpp/200901"
        xmlns:cpp="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/cpp/200901"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

        <element name="bindings" type="cpp:BindingsType" />
        <complexType name="BindingsType">
                <choice minOccurs="0" maxOccurs="unbounded">
                        <element ref="cpp:namespace" />
                        <element ref="cpp:class" />
                        <element ref="cpp:enableWrapperStyle" />
                        <element ref="cpp:memberFunction" />
                        <element ref="cpp:parameter" />
                </choice>
        </complexType>

        <element name="namespace" type="cpp:NamespaceType" />
        <complexType name="NamespaceType">
                <attribute name="name" type="xsd:anyURI" use="required" />
        </complexType>

        <element name="class" type="cpp:ClassType" />
        <complexType name="ClassType">
```

```
                <attribute name="name" type="xsd:NCName" use="required" />
        </complexType>

        <element name="memberFunction" type="cpp:MemberFunctionType" />
        <complexType name="MemberFunctionType">
                <attribute name="name" type="xsd:NCName" use="required" />
        </complexType>

        <element name="parameter" type="cpp:ParameterType" />
        <complexType name="ParameterType">
                <attribute name="part" type="xsd:string" use="required" />
                <attribute name="childElementName" type="xsd:QName"
                        use="required" />
                <attribute name="name" type="xsd:NCName" use="required" />
                <attribute name="type" type="xsd:string" use="optional" />
        </complexType>

        <element name="enableWrapperStyle" type="xsd:boolean" />
</schema>
```

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

E XML Schemas

~~Three XML schemas are defined to support the use of C++ for implementation and definition of interfaces.~~

~~The XML schema pointed to by the RDDL document at the SCA namespace URI, defined by the Assembly specification **[ASSEMBLY]** and extended by this specification, are considered to be authoritative and take precedence over the XML schema in this appendix.~~

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

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/~~200903~~200912"
      xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/~~200903~~200912"
      elementFormDefault="qualified">

   <include schemaLocation="sca-core.xsd"/>

   <element name="interface.cpp" type="sca:CPPInterface"
         substitutionGroup="sca:interface"/>

   <complexType name="CPPInterface">
      <complexContent>
         <extension base="sca:Interface">
            <sequence>
               <element name="function" type="sca:CPPFunction"
                     minOccurs="0" maxOccurs="unbounded" />
               <element name="callbackFunction" type="sca:CPPFunction"
                     minOccurs="0" maxOccurs="unbounded" />
               <any namespace="##other" processContents="lax"
                     minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="header" type="string" use="required"/>
            <attribute name="class" type="Name" use="required"/>
            <attribute name="callbackHeader" type="string" use="optional"/>
            <attribute name="callbackClass" type="Name" use="optional"/>
~~            <anyAttribute namespace="##other" processContents="lax"/>~~
         </extension>
      </complexContent>
   </complexType>

   <complexType name="CPPFunction">
      <sequence>
         <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="sca:requires"/>
            <element ref="sca:policySetAttachment"/>
         </choice>
         <any namespace="##other" processContents="lax" minOccurs="0"
               maxOccurs="unbounded" />
      </sequence>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="requires" type="sca:listOfQNames" use="optional"/>
      <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
      <attribute name="oneWay" type="boolean" use="optional"/>
      <attribute name="exclude" type="boolean" use="optional"/>
      <anyAttribute namespace="##other" processContents="lax"/>
   </complexType>

```

```
</schema>
```

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<schema  xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
      xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
      elementFormDefault="qualified">

   <include schemaLocation="sca-core.xsd"/>

   <element name="implementation.cpp" type="sca:CPPImplementation"
         substitutionGroup="sca:implementation" />
   <complexType name="CPPImplementation">
      <complexContent>
         <extension base="sca:Implementation">
            <sequence>
               <element name="function" type="sca:CPPImplementationFunction"
                     minOccurs="0" maxOccurs="unbounded" />
               <any namespace="##other" processContents="lax"
                     minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="library" type="NCName" use="required"/>
            <attribute name="header" type="NCName" use="required"/>
            <attribute name="path" type="string" use="optional"/>
            <attribute name="class" type="Name" use="optional"/>
            <attribute name="componentType" type="string" use="optional"/>
            <attribute name="scope" type="sca:CPPImplementationScope"
                  use="optional"/>
            <attribute name="eagerInit" type="boolean" use="optional"/>
            <attribute name="allowsPassByReference" type="boolean"
                  use="optional"/>
            <anyAttribute namespace="##other" processContents="lax"/>
         </extension>
      </complexContent>
   </complexType>

   <simpleType name="CPPImplementationScope">
      <restriction base="string">
         <enumeration value="stateless"/>
         <enumeration value="composite"/>
      </restriction>
   </simpleType>

   <complexType name="CPPImplementationFunction">
      <sequence>
         <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="sca:requires"/>
            <element ref="sca:policySetAttachment"/>
         </choice>
         <any namespace="##other" processContents="lax" minOccurs="0"
               maxOccurs="unbounded" />
      </sequence>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="requires" type="sca:listOfQNames" use="optional"/>
      <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
      <attribute name="allowsPassByReference" type="boolean"
         use="optional"/>
      <anyAttribute namespace="##other" processContents="lax"/>
   </complexType>
```

```
</schema>
```

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
      xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
      elementFormDefault="qualified">

   <include schemaLocation="sca-contributions.xsd"/>

   <element name="export.cpp" type="sca:CPPExport"
         substitutionGroup="sca:Export"/>

   <complexType name="CPPExport">
      <complexContent>
         <attribute name="name" type="QName" use="required"/>
         <attribute name="path" type="string" use="optional"/>
      </complexContent>
   </complexType>

   <element name="import.cpp" type="sca:CPPImport"
         substitutionGroup="sca:Import"/>

   <complexType name="CPPImport">
      <complexContent>
         <attribute name="name" type="QName" use="required"/>
         <attribute name="location" type="string" use="required"/>
      </complexContent>
   </complexType>

</schema>
```

~~B Conformance Items~~

~~This section contains a list of conformance items for the SCA C++ Client and Implementation Model specification.~~

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

F Normative Statement Summary

This section contains a list of normative statements for this specification.

Conformance ID	Description
[CPP20001]	A C++ implementation MUST implement all of the operation(s) of the service interface(s) of its componentType.
[CPP20003]	An SCA runtime MUST support these scopes; **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 ~~method.~~member function.
[CPP20013]	An SCA runtime MAY run multiple threads in a single composite scoped implementation instance object~~ and it MUST NOT perform any synchronization.~~.
[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 *@header* attribute of an *<interface.cpp/>* 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 *<function/>* child elements of the *<interface.cpp/>* element with *@exclude="true"*.
[CPP20017]	If the header file identified by the *@callbackHeader* attribute of an *<interface.cpp/>* 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 *<callbackFunction/>* child elements of the *<interface.cpp/>* element with *@exclude="true"*.
[CPP20018]	An SCA runtime MUST NOT perform any synchronization of access to component implementations.
[CPP30001]	If a remotable interface is defined with a C++ class, an SCA implementation SHOULD map the interface definition to WSDL before generating the proxy for the interface.
[CPP30002]	For each reference of a component, an SCA implementation MUST generate a service proxy derived from `ServiceProxy` that contains the operations of the reference's interface definition.
[CPP30003]	An SCA runtime MUST include an asynchronous invocation member function for every operation of a reference interface with a *@requires="asyncInvocation"* intent applied either to the operation or the reference as a whole.
[CPP30004]	An SCA runtime MUST include a response class for every response message of a reference interface that can be returned by an operation of the interface with a *@requires="asyncInvocation"* intent applied either to the operation of the reference as a whole.
[CPP40001]	An operation marked as oneWay is considered non-blocking and the SCA runtime MAY use a binding that buffers the requests to the member function and sends them at some time after they are made.
[CPP40002]	For each service of a component that includes a bidirectional interface, an SCA implementation MUST generate a service proxy derived from `ServiceProxy` that contains the operations of the reference's callback interface definition.
[CPP40003]	If a service of a component that has a callback interface contains operations with a *@requires="asyncInvocation"* intent applied either to the operation of the reference as a whole, an SCA implementation MUST include asynchronous invocation member functions and response classes as described in Long Running Request-Response Operations.
[CPP70001]	The*@name* attribute of a *<export.cpp/>* element MUST be unique amongst the *<export.cpp/>* elements in a domain.
[CPP70002]	The*@name* attribute of a *<import.cpp/>* child element of a *<contribution/>* MUST be unique amongst the *<import.cpp/>* elements in of that contribution.

Conformance ID	Description
[CPP80001]	~~The return type and types of the parameters of a member function of a local service interface MUST be one of:~~ • ~~Any of the C++ primitive types (for example, `int`, `short`, `char`). In this case the data will be passed by value as is normal for C++.~~ • ~~Pointers to any of the C++ primitive types (for example, `int *`, `short *`, `char *`).~~ • ~~The `const` keyword can be used for any pointer to a C++ primitive type (for example `const char *`). If this is used on a parameter then the destination can not change the value.~~ • ~~C++ `class`. The class will be passed by value as is normal for C++.~~ • ~~Pointer to a C++ `class`. A pointer will be passed to the destination which can then modify the original contents.~~ • ~~`DataObjectPtr`. An SDO pointer. This will be passed by reference.~~ ~~References to C++ classes (passed by reference).~~An SCA implementation MUST translate a class to tokens as part of conversion to WSDL or compatibility testing.
[CPP80002]	The return type and types of the parameters of a member function of a remotable service interface MUST be one of: • ~~Any of the C++ primitive types (for example, `int`, `short`, `char`). This will~~Any of the C++ types specified in Simple Content Binding. These types may be ~~copied.~~ • ~~`DataObjectPtr`. An SDO pointer. The SDO will be copied and~~ passed ~~to the destination.~~by-value, by-reference, or by-pointer. Unless the member function and client indicate that they allow by-reference semantics (see AllowsPassByReference), a copy will be explicitly created by the runtime for any parameters passed by-reference or by-pointer. • 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]	A C++ header file used to define an interface MUST~~:~~ ~~Declare~~ declare at least one class with: • At least one public member function. • All public member functions ~~MUST be~~are pure virtual~~ (virtual with no implementation)~~.
~~[CPP90002]~~	~~A C++ header file used to define an interface MUST NOT use the following constructs:~~ • ~~Macros~~ • ~~Inline member functions~~ • ~~Friend classes~~
[CPP100001]	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.

Conformance ID	Description
[CPP100002]	If multiple operations within the same portType indicate that they throw faults that reference the same global element, an SCA implementation MUST generate a single C++ exception class with each C++ member function referencing this class in its @WebThrows annotation.
[CPP100003]	• 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.
[CPP100004]	• 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.
[CPP100005]	An SCA implementation SHOULD provide a mechanism for overriding the default targetNamespace.
[CPP100006]	An SCA implementation MUST map a method's return type as an **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.
[CPP100008]	~~An SCA implementation MUST ensure each class that is referenced from an @WebThrows annotation MUST itself have a @WebFault annotation that associates the fault with a particular global element that will be associated with the fault message.~~An SCA implementation MUST map simple types as defined in Table 9~~-~~1 and Table 9~~-~~2 by default.
[CPP100009]	~~An SCA implementation MUST map simple types as defined in Table 1 and Table 2 by default.~~An SCA implementation MUST map a WSDL portType to a remotable C++ interface definition.
[CPP100010]	~~An SCA implementation MUST reject a composite file that does not conform to http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-cpp-1.1.xsd or http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-cpp-1.1.xsd.~~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 componentType or constraining type file that does not conform to~~ An SCA implementation MUST reject a composite file that does not conform to http://docs.oasis-open.org/opencsa/sca/~~200903~~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 contribution file that does not conform to~~ An SCA implementation MUST reject a componentType file that does not conform to http://docs.oasis-open.org/opencsa/sca/~~200903~~200912/sca-~~contribution~~interface-cpp-1.1.xsd~~.~~.

Conformance ID	Description
[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-wsdlext-cpp-1.1.xsd.

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

F.1 Annotation Normative Statement Summary

This section contains a list of normative statements related to source file annotations for this specification.

Conformance ID	**Description**
[CPPA0001]	If annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as described. The SCA runtime MUST only process the SCDL files and not the annotations.
[CPPA0002]	If multiple annotations apply to a program element, all of the annotations SHOULD be in the same comment block.
[CPPA0003]	An SCA implementation MUST treat a class with an @WebService annotation specified as if ~~an @Interface~~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 valued of the @Function annotation.
[CPPC0001]	If WSDL mapping annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to WSDL as described.
[CPPC0002]	An SCA implementation MUST treat any instance of a @~~Interface~~Remotable annotation and without an explicit @WebService annotation as if a @WebService annotation with no parameters was specified.
[CPPC0004]	~~If a @WebService does not have a **portName** element, an SCA implementation MUST use the value associated with the **name** element, suffixed with *"Port"*.~~The value of the paramName of a @WebParam annotation MUST be the name of a parameter of the member function the annotation is applied to.
~~[CPPC0004]~~	~~Only named parameters MAY be referenced by a @WebParam annotation.~~
[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.
[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.

Conformance ID	Description
[CPPC0009]	An SCA implementation MUST treat a member function annotated with an @Function annotation and without an explicit @WebFunction annotation as if a @WebFunction annotation with with an operationName value equal to the name value of the @Function annotation, an exclude value equal to the exclude value of the @Operation annotation and no other parameters was specified.

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

F.2 WSDL Extention Normative Statement Summary

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

Conformance ID	Description
[CPPD0001]	If WSDL extensions are supported by an implementation, all the extensions defined here MUST be supported and MUST be mapped to C++ as described.
[CPPD0002]	A <cpp:bindings/> element MUST NOT have more than one <cpp:class/> child element.
[CPPD0003]	A <cpp:bindings/> element MUST NOT have more than one <cpp:enableWrapperStyle/> child element.
[CPPD0004]	A <cpp:bindings/> element MUST NOT have more than one <cpp:namespace/> child element.
[CPPD0005]	A <cpp:bindings/> element MUST NOT have more than one <cpp:memberFunction/> child element.
[CPPD0006]	The @type attribute of a <cpp:parameter/> element MUST be a ~~valid~~ C++ type~~.~~ specified in Simple Content Binding.
[CPPD0007]	An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present. Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted and their corresponding SCA WSDL extensions.

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

~~E.4~~F.3 JAX-WS ~~Conformance~~Normative Statements

The JAX-WS 2.1 specification **[JAXWS21]** defines ~~conformance~~normative statements for various requirements defined by that specification. ~~The following table~~Table F-4 outlines those ~~conformance statements, and describes whether the conformance statement applies~~normative statemetns which apply to the WSDL ~~binding~~mapping described in this specification.

~~Section~~Number	Conformance ~~Statement~~Point	Notes	Conformance ID
2.1	WSDL 1.1 support	[A]	[CPPF0001]
2.2	Customization required	[CPPD0001] The reference to the JAX-WS binding language ~~are~~is treated as a reference to the C++ WSDL extensions defined in section WSDL C++ Mapping Extensions.	

~~Section~~Number	Conformance ~~Statement~~Point	Notes	Conformance ID
2.3	Annotations on generated classes		[CPPF0002]
2.~~1~~4	Definitions mapping	[CPP100001]	
2.~~1~~5	WSDL and XML Schema import directives		[CPPF0003]
2.~~1.1~~6	Optional WSDL extensions		[CPPF0004]
2.~~2~~7	SEI naming		[CPPF0005]
2.~~2~~8	javax.jws.WebService required	[B] References to javax.jws.WebService in the conformance statement are treated as the C++ annotation @WebService.	[CPPF0006]
2.~~3~~10	Method naming		[CPPF0007]
2.~~3~~11	javax.jws.WebMethod required	[A], [B] References to javax.jws.WebMethod in the conformance statement are treated as the C++ annotation @WebFunction.	[CPPF0008]
2.~~3~~12	Transmission primitive support		[CPPF0009]
2.~~3~~13	Using javax.jws.OneWay	[A], [B] References to javax.jws.OneWay in the conformance statement are treated as the C++ annotation @OneWay.	[CPPF0010]
2.~~3.1~~14	Using javax.jws.SOAPBinding	[A], [B] References to javax.jws.SOAPBinding in the conformance statement are treated as the C++ annotation @SOAPBinding.	[CPPF0011]
2.~~3.1~~15	Using javax.jws.WebParam	[A], [B] References to javax.jws.WebParam in the conformance statement are treated as the C++ annotation @WebParam.	[CPPF0012]

~~Section~~Number	Conformance ~~Statement~~Point	Notes	Conformance ID
2.~~3.1~~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.~~3.1.1~~18	Non-wrapped parameter naming		[CPPF0014]
2.~~3.1.2~~19	Default mapping mode		[CPPF0015]
2.~~3.1.2~~20	Disabling wrapper style	[B] References to jaxws:enableWrapperStyle in the conformance statement are treated as the WSDL extension cpp:enableWrapperStyle.	[CPPF0016]
2.~~3.1.2~~21	Wrapped parameter naming		[CPPF0017]
2.~~3.1.2~~22	Parameter name clash	[A]	[CPPF0018]
2.~~5~~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.~~5~~39	Exception naming		[CPPF0020]
2.~~5~~40	Fault equivalence	~~[A]~~[CPP100002]	~~[CPPF0021]~~
2.~~6~~42	Required WSDL extensions	MIME Binding not necessary	[CPPF0022]
2.~~6.1~~43	Unbound message parts	[A]	[CPPF0023]
2.~~6.2.1~~44	Duplicate headers in binding		[CPPF0024]
2.~~6.2.1~~45	Duplicate headers in message		[CPPF0025]
3.1	WSDL 1.1 support	[A]	[CPPF0026]
3.2	Standard annotations	[A] [CPPC0001]	
3.~~1~~3	Java identifier mapping	[A]	[CPPF0027]

~~Section~~Number	Conformance ~~Statement~~Point	Notes	Conformance ID
3.~~1.1~~4	Method name disambiguation	[A] References to javax.jws.WebMethod in the conformance statement are treated as the C++ annotation @WebFunction.	[CPPF0028]
3.~~2~~6	WSDL and XML Schema import directives		[CPPF0029]
3.~~4~~8	portType naming		[CPPF0030]
3.~~4.1~~9	Inheritance flattening	[A]	[CPPF0044]
3.~~4.1~~10	Inherited interface mapping		[CPPF0045]
3.~~5~~11	Operation naming		[CPPF0031]
3.~~5.1~~12	One-way mapping	[B] References to javax.jws.OneWay in the conformance statement are treated as the C++ annotation @OneWay.	[CPPF0032]
3.~~5.1~~13	One-way mapping errors		[CPPF0033]
3.~~6.1~~15	Parameter classification	[CPP100006]	
3.~~6.1~~16	Parameter naming		[CPPF0035]
3.~~6.1~~17	Result naming		[CPPF0036]
3.~~6.1~~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]

~~Section~~Number	Conformance ~~Statement~~Point	Notes	Conformance ID
3.~~7~~27	~~Exception naming~~Binding selection	~~[A]~~ ~~References to javax.jws.WebFault in the conformance statement are treated as the C++ annotation @WebFault.~~References to the BindingType annotation are treated as references to SOAP related intents defined by **[POLICY]**.	~~[CPPF0038]~~[CPPF0039]
3.~~8~~28	~~Binding selection~~SOAP binding support	~~References to the BindingType annotation are treated as references to SOAP related intents defined by **[POLICY]**.~~[A]	~~[CPPF0039]~~[CPPF0040]
3.~~10~~29	SOAP binding ~~support~~style required	~~[A]~~	~~[CPPF0040]~~[CPPF0041]
3.~~10.1~~31	~~SOAP binding style required~~Port selection		~~[CPPF0041]~~[CPPF0042]
3.~~11~~32	Port ~~selection~~binding	References to the BindingType annotation are treated as references to SOAP related intents defined by **[POLICY]**.	~~[CPPF0042]~~[CPPF0043]
~~3.11~~	~~Port binding~~	~~References to the BindingType annotation are treated as references to SOAP related intents defined by **[POLICY]**.~~	~~[CPPF0043]~~

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

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

~~B.1.1 Ignored Conformance Statements~~

~~Section~~	~~Conformance Statement~~	~~Notes~~
~~2.2~~	~~javax.xml.bind.XmlSeeAlso required~~	
~~2.3.1~~	~~use of JAXB annotations~~	
~~2.3.1.2~~	~~Using javax.xml.ws.RequestWrapper~~	
~~2.3.1.2~~	~~Using javax.xml.ws.ResponseWrapper~~	
~~2.3.3~~	~~Use of Holder~~	
~~2.3.4~~	~~Asynchronous mapping required~~	
~~2.3.4~~	~~Asynchronous mapping option~~	
~~2.3.4.2~~	~~Asynchronous method naming~~	
~~2.3.4.2~~	~~Asynchronous parameter naming~~	

~~2.3.4.2~~	~~Failed method invocation~~	
~~2.3.4.4~~	~~Response bean naming~~	
~~2.3.4.5~~	~~Asynchronous fault reporting~~	
~~2.3.4.5~~	~~Asychronous fault cause~~	
~~2.4~~	~~JAXB class mapping~~	
~~2.4~~	~~JAXB customization use~~	
~~2.4~~	~~JAXB customization clash~~	
~~2.4.1~~	~~javax.xml.ws.wsaddressing.W3CEndpointReference~~	
~~2.5~~	~~Fault Equivalence~~	
~~2.6.3.1~~	~~Use of MIME type information~~	
~~2.6.3.1~~	~~MIME type mismatch~~	
~~2.6.3.1~~	~~MIME part identification~~	
~~2.7~~	~~Service superclass required~~	
~~2.7~~	~~Service class naming~~	
~~2.7~~	~~javax.xml.ws.WebServiceClient required~~	
~~2.7~~	~~Default constructor required~~	
~~2.7~~	~~2 argument constructor required~~	
~~2.7~~	~~Failed getPort Method~~	
~~2.7~~	~~javax.xml.ws.WebEndpoint required~~	
~~3.2~~	~~Package name mapping~~	
~~3.3~~	~~Class mapping~~	
~~3.6~~	~~use of JAXB annotations~~	
~~3.6.2.1~~	~~Default wrapper bean names~~	
~~3.6.2.1~~	~~Default wrapper bean package~~	
~~3.6.2.3~~	~~Null Values in rpc/literal~~	
~~3.7~~	~~java.lang.RuntimeExceptions and java.rmi.RemoteExceptions~~	
~~3.7~~	~~Fault bean name clash~~	
~~3.11~~	~~Service creation~~	

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

F.3.1 Ignored Normative Statements

Number	**Conformance Point**

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
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	Asychronous 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

Number	Conformance Point
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
3.26	Fault bean name clash
3.30	Service creation

3900 *Table F-5: JAX-WS Normative Statements that Are Not Applicable to SCA C++*

~~F~~G Migration

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.

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

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.

~~G~~H Acknowledgements

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

Participants:

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

~~H~~I Revision History

[optional; should not be included in OASIS Standards]

Revision	Date	Editor	Changes Made
			•