



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

Committee Draft 03 / Public Review Draft 01

19 March 2009

Specification URIs:

This Version:

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

Previous Version:

<http://www.oasis-open.org/committees/download.php/31093/sca-ccni-1.1-spec-cd02.doc>
[\(Authoritative\)](http://www.oasis-open.org/committees/download.php/31737/sca-ccni-1.1-spec-cd02.pdf)

Latest Version:

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

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>
<http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/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 component implemented in C gets access to services and calls their operations.

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

Status:

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

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

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

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

Notices

Copyright © OASIS® 2007, 2009. All Rights Reserved.

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

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

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

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

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

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

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

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

Table of Contents

1	Introduction.....	8
1.1	Terminology	8
1.2	Normative References	8
1.3	Conventions.....	9
1.3.1	Naming Conventions	9
1.3.2	Typographic Conventions.....	9
2	Basic Component Implementation Model.....	10
2.1	Implementing a Service	10
2.1.1	Implementing a Remotable Service.....	11
2.1.2	AllowsPassByReference	11
2.1.3	Implementing a Local Service.....	12
2.2	Component and Implementation Scopes.....	12
2.2.1	Stateless scope	13
2.2.2	Composite scope.....	13
2.3	Implementing a Configuration Property	13
2.4	Component Type and Component.....	14
2.4.1	Interface.c.....	15
2.4.2	Function and CallbackFunction	15
2.4.3	Implementation.c	16
2.4.4	Implementation Function	17
2.5	Implementing a Service with a Program	17
3	Basic Client Model.....	19
3.1	Accessing Services from Component Implementations.....	19
3.2	Accessing Services from non-SCA component implementations	20
3.3	Calling Service Operations	20
3.3.1	Proxy Functions.....	20
3.4	Long Running Request-Response Operations	21
3.4.1	Asynchronous Invocation	21
3.4.2	Polling Invocation	22
3.4.3	Synchronous Invocation	23
4	Asynchronous Programming	25
4.1	Non-blocking Calls.....	25
4.2	Callbacks	25
4.2.1	Using Callbacks.....	26
4.2.2	Callback Instance Management	27
4.2.3	Implementing Multiple Bidirectional Interfaces.....	27
5	Error Handling	28
6	C API.....	29
6.1	SCA Programming Interface.....	29
6.1.1	SCAGetReference.....	31
6.1.2	SCAGetReferences	32
6.1.3	SCAInvoke.....	32
6.1.4	SCAProperty<T>	33

6.1.5	SCAGetReplyMessage.....	34
6.1.6	SCAGetFaultMessage.....	34
6.1.7	SCASetFaultMessage	35
6.1.8	SCASelf.....	35
6.1.9	SCAGetCallback.....	36
6.1.10	SCAResponse.....	36
6.1.11	SCAInvokeAsync.....	37
6.1.12	SCAInvokePoll	37
6.1.13	SCACheckResponse.....	38
6.1.14	SCACancelInvoke	38
6.1.15	SCAEntryPoint	38
6.2	Program-Based Implementation Support.....	39
6.2.1	SCAService	39
6.2.2	SCAOperation	39
6.2.3	SCAMessageIn.....	40
6.2.4	SCAMessageOut.....	40
7	C Contributions.....	41
7.1	Executable files.....	41
7.1.1	Executable in contribution	41
7.1.2	Executable shared with other contribution(s) (Export)	41
7.1.3	Executable outside of contribution (Import)	42
7.2	componentType files.....	42
7.3	C Contribution Extensions	43
7.3.1	Export.c	43
7.3.2	Import.c	43
8	Types Supported in Service Interfaces.....	45
8.1	Local service.....	45
8.2	Remotable service	45
9	Restrictions on C header files.....	46
10	WSDL to C and C to WSDL Mapping.....	47
10.1	Interpretations for WSDL to C Mapping	47
10.1.1	Definitions	47
10.1.2	PortType.....	47
10.1.3	Operations.....	48
10.1.4	Types	48
10.1.5	Fault	48
10.1.6	Service and Port.....	49
10.1.7	XML Names	49
10.2	Interpretations for C to WSDL Mapping	49
10.2.1	Package	49
10.2.2	Class	49
10.2.3	Interface	49
10.2.4	Method	49
10.2.5	Method Parameters and Return Type	50
10.2.6	Service Specific Exception	50

10.2.7	Generics.....	50
10.2.8	Service and Ports.....	51
10.3	Data Binding	51
10.3.1	Simple Content Binding.....	51
10.3.2	Complex Content Binding.....	56
11	Conformance.....	61
11.1	Conformance Targets.....	61
11.2	SCA Implementations.....	61
11.3	SCA Documents	61
11.4	C Files	62
11.5	WSDL Files.....	62
A	C SCA Annotations	63
A.1	Application of Annotations to C Program Elements.....	63
A.2	Interface Header Annotations.....	63
A.2.1	@Interface	63
A.2.2	@Operation.....	64
A.2.3	@Remotable	65
A.2.4	@Callback.....	65
A.2.5	@OneWay	66
A.3	Implementation Annotations.....	66
A.3.1	@ComponentType	66
A.3.2	@Service	67
A.3.3	@Reference.....	67
A.3.4	@Property.....	68
A.3.5	@Scope	68
A.3.6	@Init.....	69
A.3.7	@Destroy	69
A.3.8	@EagerInit.....	70
A.3.9	@AllowsPassByReference	70
A.4	Base Annotation Grammar	70
B	C SCA Policy Annotations.....	72
B.1	General Intent Annotations.....	72
B.2	Specific Intent Annotations	73
B.2.1	Security Interaction	74
B.2.2	Security Implementation.....	74
B.2.3	Reliable Messaging.....	74
B.2.4	Transactions	74
B.2.5	Miscellaneous	75
B.3	Application of Intent Annotations.....	75
B.4	Policy Annotation Scope	75
B.5	Relationship of Declarative And Annotated Intents	77
B.6	Policy Set Annotations	77
B.7	Policy Annotation Grammar Additions	77
B.8	Annotation Constants	78
C	C WSDL Annotations.....	79

C.1 Interface Header Annotations	79
C.1.1 @WebService.....	79
C.1.2 @WebFunction	80
C.1.3 @WebOperation	82
C.1.4 @OneWay	84
C.1.5 @WebParam	85
C.1.6 @WebResult.....	87
C.1.7 @SOAPBinding	89
C.1.8 @WebFault.....	90
C.1.9 @WebThrows	92
D C WSDL Mapping Extensions	93
D.1 <sca-c:bindings>.....	93
D.2 <sca-c:prefix>	93
D.3 <sca-c:enableWrapperStyle>.....	94
D.4 <sca-c:function>.....	96
D.5 <sca-c:struct>	97
D.6 <sca-c:parameter>	99
D.7 JAX-WS WSDL Extensions.....	100
D.8 WSDL Extensions Schema	101
E XML Schemas	102
E.1 sca-interface-c-1.1.xsd.....	102
E.2 sca-implementation-c-1.1.xsd	102
E.3 sca-contribution-c-1.1.xsd	103
F Conformance Items	105
F.1 JAX-WS Conformance	110
F.1.1 Ignored Conformance Statements.....	112
G Migration.....	114
G.1 Implementation.c attributes.....	114
G.2 SCALocate and SCALocateMultiple	114
H Acknowledgements	115
I Revision History	116

1 **Introduction**

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

10 **1.1 Terminology**

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

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

16
17 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 "	Defined by the SCA specifications
sca-c	" http://docs.oasis-open.org/ns/opencsa/sca-c-c/200901 "	

18 **1.2 Normative References**

- 19 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF
20 RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>.
- 21 **[ASSEMBLY]** OASIS Committee Draft 03, *Service Component Architecture Assembly Model Specification Version 1.1*, March 2009. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.pdf>
- 22 **[POLICY]** OASIS Committee Draft 02, *SCA Policy Framework Version 1.1*, March 2009.
23 <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec.pdf>
- 24 **[SDO21]** OSOA, *Service Data Objects For C Specification*, September 2007.
25 http://www.osoa.org/download/attachments/36/SDO_Specification_C_V2.1.pdf
- 26 **[WSDL11]** World Wide Web Consortium, *Web Service Description Language (WSDL)*,
27 March 2001. <http://www.w3.org/TR/wsdl>
- 28 **[XSD]** World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*,
29 October 2004. <http://www.w3.org/TR/xmlschema-2/>
- 30 **[JAXWS21]** Doug. Kohlert and Arun Gupta, *The Java API for XML-Based Web Services (JAX-WS) 2.1*, JSR, JCP, May 2007.
31 <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>

35 **1.3 Conventions**

36 **1.3.1 Naming Conventions**

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

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

48 **1.3.2 Typographic Conventions**

49 This specification follows some typographic conventions for some specific constructs

- 50 • XML attributes are identified in text as `@attribute`
- 51 • Language identifiers used in text are in `courier`
- 52 • Literals in text are in *italics*

53 2 Basic Component Implementation Model

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

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

59 2.1 Implementing a Service

60 A component implementation based on a set of C functions (a **C implementation**) provides one or more
61 services.

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

- 64 • the declaration of the C functions implementing the services
- 65 • a WSDL 1.1 portType **[WSDL11]**

66 If function declarations are used to define the interface, they will typically be placed in a separate header
67 file. A C implementation MUST implement all of the operation(s) of the service interface(s) of its
68 componentType. **[C20001]**

69

70 The following snippets show the C service interface and the C functions of a C implementation.

71

72 Service interface.

73

```
74    /* LoanService interface */  
75    char approveLoan(long customerNumber, long loanAmount);
```

76

77 Implementation.

```
78    #include "LoanService.h"  
79  
80    char approveLoan(long customerNumber, long loanAmount)  
81    {  
82       ...  
83    }
```

84

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

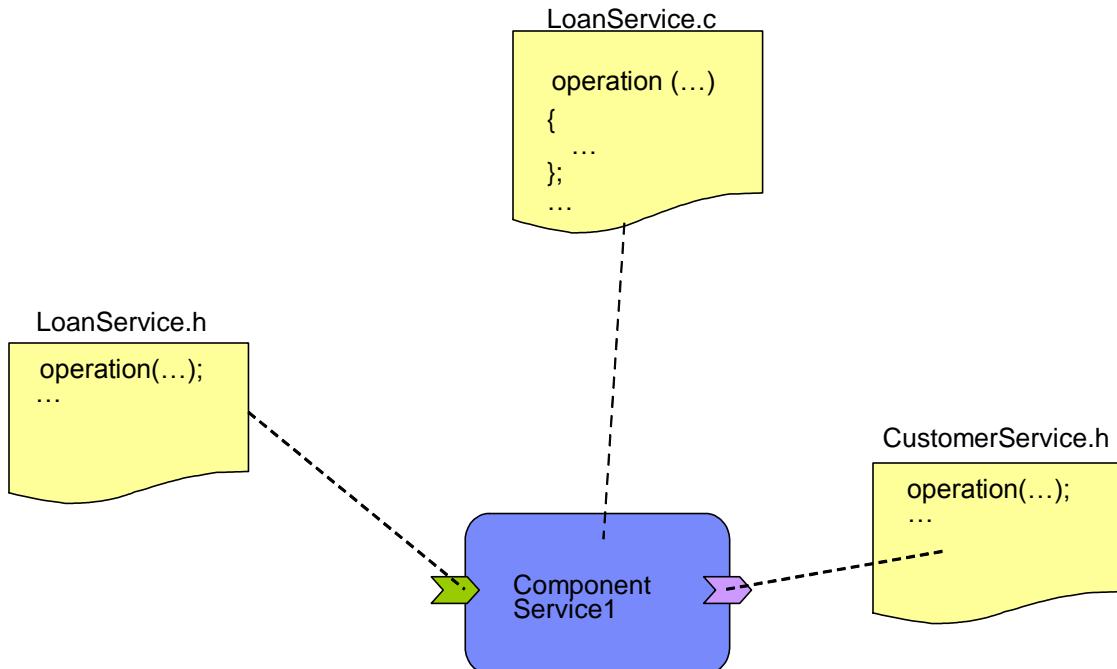
86

```
87    <?xml version="1.0" encoding="ASCII"?>  
88    <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
89       <service name="LoanService">  
90           <interface.c header="LoanService.h"/>  
91           </service>  
92       </componentType>
```

93

94 The following picture shows the relationship between the C header files and implementation files for a
95 component that has a single service and a single reference.

96



97

98 2.1.1 Implementing a Remotable Service

99 A `@remotable="true"` attribute on an `interface.c` element indicates that the interface is **remotable** as
100 described in the Assembly Specification **[ASSEMBLY]**. The following snippet shows the component type
101 for a remotable service:

102

```

103 <?xml version="1.0" encoding="ASCII"?>
104 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
105     <service name="LoanService">
106         <interface.c header="LoanService.h" remotable="true"/>
107     </service>
108 </componentType>

```

109

2.1.2 AllowsPassByReference

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

120

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

127 address space. See Implementation.c and Implementation Function for a description of the
128 @allowsPassByReference attribute and how it is used.

129 **2.1.2.1 Marking services and references as “allows pass by reference”**

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

- 132 • Function execution will not modify any input parameter before the function returns.
- 133 • The service implementation will not retain a pointer to any by-reference input parameter, return value
134 or fault data after the function returns.
- 135 • The function will observe “allows pass by value” client semantics (see below) for any callbacks that it
136 makes.

137 Marking a client as “allows pass by reference” asserts that the client observe the following restrictions for
138 all references’ functions:

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

144 **2.1.2.2 Using “allows pass by reference” to optimize remotable calls**

145 The SCA runtime MAY use by-reference semantics when passing input parameters, return values or
146 exceptions on calls to remotable services within the same system address space if both the service
147 function implementation and the client are marked “allows pass by reference”. [C20016]

148 The SCA runtime MUST use by-value semantics when passing input parameters, return values and
149 exceptions on calls to remotable services within the same system address space if the service function
150 implementation is not marked “allows pass by reference” or the client is not marked “allows pass by
151 reference”. [C20017]

152 **2.1.3 Implementing a Local Service**

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

154 **2.2 Component and Implementation Scopes**

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

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

160 When a scope is not specified in an implementation file, the SCA runtime will interpret the implementation
161 scope as **stateless**.

162 An SCA runtime MUST support these scopes; **stateless** and **composite**. Additional scopes MAY be
163 provided by SCA runtimes. [C20003]

164 The following snippet shows the component type for a composite scoped component:

```
166 <component name="LoanService">
167   <implementation.c module="loan" componentType="LoanService"
168     scope="composite"/>
169 </component>
```

171 Certain scoped implementations potentially also specify **lifecycle functions** which are called when an
172 implementation is instantiated or the scope is expired. An implementation is either instantiated eagerly
173 when the scope is started (specified by `@scope="composite" @eagerInit="true"`), or lazily when the first
174 client request is received. Lazy instantiation is the default for all scopes. The C implementation uses the
175 `@init="true"` attribute of an implementation function element to denote the function to be called upon
176 initialization and the `@destroy="true"` attribute for the function to be called when the scope ends. A C
177 implementation MUST only designate functions with no arguments and a void return type as lifecycle
178 functions. [C20004]

179 **2.2.1 Stateless scope**

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

182 The concurrency model for the stateless scope is single threaded. An SCA runtime MUST ensure that a
183 stateless scoped implementation instance object is only ever dispatched on one thread at any one time.
184 In addition, within the SCA lifecycle of an instance, an SCA runtime MUST only make a single invocation
185 of one business method. [C20014]

186 Lifecycle functions are not defined for stateless implementations.

187 **2.2.2 Composite scope**

188 All service requests are dispatched to the same implementation instance for the lifetime of the containing
189 composite, i.e. the binary implementing the component is loaded into memory once and all requests are
190 processed by this single instance. The lifetime of the containing composite is defined as the time it
191 becomes active in the runtime to the time it is deactivated, either normally or abnormally.

192 A composite scoped implementation can also specify eager initialization using the `@eagerInit="true"`
193 attribute on the *implementation.c* element of a component definition. When marked for eager initialization,
194 the composite scoped instance will be created when its containing component is started.

195 The concurrency model for the composite scope is multi-threaded. An SCA runtime MAY run multiple
196 threads in a single composite scoped implementation instance object and it MUST NOT perform any
197 synchronization. [C20015]

198 Composite scope supports both `@init="true"` and `@destroy="true"` functions.

199 **2.3 Implementing a Configuration Property**

200 Component implementations can be configured through properties. The properties and their types (not
201 their values) are defined in the component type. The C component can retrieve properties values using
202 the `SCAProperty<PropertyType>()` functions, for example `SCAPropertyInt()` to access an Int
203 type property..

204 The following code extract shows how to get the property values.

```
205 #include "SCA.h"  
206  
207 void clientFunction()  
208 {  
209     ...  
210  
211     int32_t loanRating;  
212     int compCode, reason;  
213  
214     ...  
215  
216     SCAPropertyInt(L"maxLoanValue", &loanRating, &compCode, &reason);  
217     ...  
218 }  
219  
220 }
```

222 If the property is many valued, an array of the appropriate type is used as the second parameter, and the
223 third parameter would point to an int that would receive the number of values. The type for the property
224 SHOULD NOT allow more values to be defined than the size of the array in the implementation.

225 **2.4 Component Type and Component**

226 For a C component implementation, a component type is specified in a side file. By default, the
227 componentType side file is in the root directory of the composite containing the component or some
228 subdirectory of the composite root directory with a name specified on the @componentType attribute.
229 The location can be modified as described below.

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

232 The following snippets show a C service interface and a C implementation of a service.

233

```
234     /* LoanService interface */  
235     char approveLoan(long customerNumber, long loanAmount);
```

236

237 Implementation.

238

```
239     #include "LoanService.h"  
240  
241     char approveLoan(long customerNumber, long loanAmount)  
242     {  
243         ...  
244     }
```

245

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

247

```
248 <?xml version="1.0" encoding="ASCII"?>  
249 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
250     <service name="LoanService">  
251         <interface.c header="LoanService.h" />  
252     </service>  
253 </componentType>
```

254

255 The following snippet shows the component using the implementation.

256

```
257 <?xml version="1.0" encoding="ASCII"?>  
258 <composite      xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
259  
260     name="LoanComposite" >  
261  
262     ...  
263  
264     <component name="LoanService">  
265         <implementation.c module="loan" componentType="LoanService" />  
266     </component>  
267  
268     ...  
269 </composite>
```

271 **2.4.1 Interface.c**

272 The following snippet shows the schema for the C interface element used to type services and references
273 of component types.

274

```
275 <?xml version="1.0" encoding="ASCII"?>
276 <!-- interface.c schema snippet -->
277 <interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
278     header="string" remotable="boolean"? callbackHeader="string"? >
279
280     <function ... />*
281     <callbackFunction ... />*
282
283 </interface.c>
```

284

285 The *interface.c* element has the following **attributes**:

- **header : string (1..1)** – full name of the header file, including either a full path, or its equivalent, or a relative path from the composite root. This header file describes the interface.
- **callbackHeader : string (0..1)** – full name of the header file that describes the callback interface, including either a full path, or its equivalent, or a relative path from the composite root.
- **remotable : boolean (0..1)** – indicates whether the service is remotable or local. The default is local.
See Implementing a Remotable Service

292 The *interface.c* element has the following **child elements**:

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

295 **2.4.2 Function and CallbackFunction**

296 Some functions of an interface have behavioral characteristics, which will be described later, that need to
297 be identified. This is done using a *function* or *callbackFunction* child element of *interface.c*. These child
298 elements are also used when not all functions in a header file are part of the interface or when the
299 interface is implemented by a program.

- If the header file identified by the @header attribute of an *<interface.c/>* element contains function declarations that are not operations of the interface, then the functions that define operations of the interface MUST be identified using *<function/>* child elements of the *<interface.c/>* element. [C20006]
- If the header file identified by the @callbackHeader attribute of an *<interface.c/>* element contains function declarations that are not operations of the callback interface, then the functions that define operations of the callback interface MUST be identified using *<callbackFunction/>* child elements of the *<interface.c/>* element. [C20007]
- If the header file identified by the @header or @callbackHeader attribute of an *<interface.c/>* element defines the operations of the interface (callback interface) using message formats, then all functions of the interface (callback interface) MUST be identified using *<function/>* (*<callbackFunction/>*) child elements of the *<interface.c/>* element. [C20008]

311 The following snippet shows the *interface.c* schema with the schema for the *function* and
312 *callbackFunction* child elements:

313

```
314 <?xml version="1.0" encoding="ASCII"?>
315 <!-- interface.c schema snippet -->
316 <interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"... >
317
318     <function name="NCName" requires="listOfQNames"? oneWay="Boolean"?
319         input="NCName"? output="NCNAME"? />*
320     <callbackFunction name="NCName" requires="listOfQNames"? oneWay="Boolean"?
```

```

321         input="NCName"? output="NCName"? /*
322
323     </interface.c>
324

```

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

- **name : NCName (1..1)** – name of the function being decorated or included in the interface. The @name attribute of a **<function/>** child element of a **<interface.c/>** MUST be unique amongst the **<function/>** elements of that **<interface.c/>**. [C20009]
- 329 The @name attribute of a **<callbackFunction/>** child element of a **<interface.c/>** MUST
330 be unique amongst the **<callbackFunction/>** elements of that **<interface.c/>**. [C20010]
- 331 • **requires : listOfQNames (0..1)** – list of intents [POLICY] needed by this function.
- 332 • **oneWay : boolean (0..1)** – see Non-blocking Calls
- 333 • **input : NCNAME (0..1)** – If the header file identified by the @header or @callbackHeader attribute of
334 an **<interface.c/>** element defines the operations of the interface (callback interface) using message
335 formats, then the struct defining the input message format MUST be identified using an @input
336 attribute. [C20011] (See Implementing a Service with a Program)
- 337 • **output : NCNAME (0..1)** – If the header file identified by the @header or @callbackHeader attribute
338 of an **<interface.c/>** element defines the operations of the interface (callback interface) using
339 message formats, then the struct defining the output message format MUST be identified using an
340 @input attribute. [C20012]

341 2.4.3 Implementation.c

342 The following snippet shows the schema for the C implementation element used to define the
343 implementation of a component.

```

344
345 <?xml version="1.0" encoding="ASCII"?>
346 <!-- implementation.c schema snippet -->
347 <implementation.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
348   module="NCName" library="boolean"? path="string"?
349   scope="scope"? componentType="string" allowsPassByReference="Boolean"?
350   eagerInit="Boolean"? init="Boolean"? destroy="Boolean"? >
351
352   <function ... />*
353
354 </implementation.c>

```

- 355
- 356 The **implementation.c** element has the following **attributes**:
- 357 • **module : NCName (1..1)** – name of the binary executable for the service component. This is the root
358 name of the module.
 - 359 • **library : boolean (0..1)** – indicates whether the service is implemented as a library or a program. The
360 default is library. See Implementing a Service with a Program
 - 361 • **path : string (0..1)** – path to the module which is either relative to the root of the contribution
362 containing the composite or is prefixed with a contribution import name and is relative to the root of
363 the import. See C Contributions.
 - 364 • **scope : CImplementationScope (0..1)** – indicates the scope of the component implementation. The
365 default is stateless. Component and Implementation Scopes
 - 366 • **componentType : string (1..1)** – name of the componentType file. A “.componentType” extention
367 will be appended. A path to the componentType file which is relative to the root of the contribution
368 containing the composite or is prefixed with a contribution import name and is relative to the root of
369 the import (see C Contributions) can be included.

- **allowsPassByReference : boolean (0..1)** – indicates the implementation allows pass by reference data exchange semantics on calls to it or from it. These semantics apply to all services provided by and references used by an implementation. See AllowsPassByReference
 - **eagerInit : boolean (0..1)** – indicates a composite scoped implementation is to be initialized when it is loaded. See Composite scope
 - **init : boolean (0..1)** – indicates program is to be called with an initialize flag to initialize the implementation. See Component and Implementation Scopes
 - **destroy : boolean (0..1)** – indicates is to be called with a destroy flag to cleanup the implementation. See Component and Implementation Scopes
- The *interface.c* element has the following **child element**:
- **function : CImplementationFunction (0..n)** – see Implementation Function

2.4.4 Implementation Function

Some functions of an implementation have operational characteristics that need to be identified. This is done using a *function* child element of *implementation.c*

The following snippet shows the *implementation.c* schema with the schema for a *function* child element:

```

<?xml version="1.0" encoding="ASCII"?>
<!-- ImplementationFunction schema snippet -->
<implementation.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
    <function name="NCName" requires="listOfQNames"?
        allowsPassByReference="Boolean"?
        init="Boolean"? destroy="Boolean"? />*
</implementation.c>
```

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

- **name : NCName (1..1)** – name of the function being decorated. The @name attribute of a *<function>* child element of a *<implementation.c>* MUST be unique amongst the *<function>* elements of that *<implementation.c>*. [C20013]
- **requires : listOfQNames (0..1)** – list of intents [POLICY] needed by this function.
- **allowsPassByReference : boolean" (0..1)** – indicates the function allows pass by reference data exchange semantics. See AllowsPassByReference
- **init : boolean (0..1)** – indicates this function is to be called to initialize the implementation. See Component and Implementation Scopes
- **destroy : boolean (0..1)** – indicates this function is to be called to cleanup the implementation. See Component and Implementation Scopes

2.5 Implementing a Service with a Program

Depending on the execution platform, services might be implemented in libraries, programs, or a combination of both libraries and programs. Services implemented as subroutines in a library are called directly by the runtime. Input and messages are passed as parameters, and output messages can either be additional parameters or a return value. Both local and remoteable interfaces are easily supported by this style of implementation.

For services implemented as programs, the SCA runtime uses normal platform functions to invoke the program. Accordingly, a service implemented as a program will run in its own address space and in its own process and its interface is most appropriately marked as remotable. A service implemented in a program will have either stateless scope. Local services implemented as subroutines used by a service implemented in a program can run in the address space and process of the program.

418 Since a program can implement multiple services and often will implement multiple operations, the
419 program has to query the runtime to determine which service and operation caused the program to be
420 invoked. This is done using `SCAService()` and `SCAOperation()`. Once the specific service and
421 operation is known, the proper input message can be retrieved using `SCAMessageIn()`. Once the logic
422 of the operation is finished `SCAMessageOut()` is used to provide the return data to the runtime to be
423 marshalled.

424 Since a program does not have a specific prototype for each operation of each service it implements, a C
425 interface definition for the service identifies the operation names and the input and output message
426 formats using functions elements, with input and output attributes, in an *interface.c* element. Alternatively,
427 an external interface definition, such as a WSDL document, is used to describe the operations and
428 message formats.

429 The following shows a program implementing a service using these support functions.
430

```
431 #include "SCA.h"
432 #include "myInterface.h"
433 main () {
434     wchar_t myService [255];
435     wchar_t myOperation [255];
436     int compCode, reason;
437     struct FirstInputMsg myFirstIn;
438     struct FirstOutputMsg myFirstOut;
439
440
441     SCAService(myService, &compCode, &reason);
442
443     SCAOperation(myOperation, &compCode, &reason);
444
445     if (wstrcmp(myOperation,L"myFirstOperation")==0) {
446         SCAMessageIn(myService, myOperation,
447                     sizeof(struct FirstInputMsg), (void *)&myFirstIn,
448                     &compCode, &reason);
449         ...
450         SCAMessageOut(myService, myOperation,
451                     sizeof(struct FirstOutputMsg), (void *)&myFirstOut,
452                     &compCode, &reason);
453     }
454     else
455     {
456         ...
457     }
458 }
```

459 3 Basic Client Model

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

462 3.1 Accessing Services from Component Implementations

463 A service can get access to another service using a reference of the current component

464

465 The following shows the `SCAGetReference()` function used for this.

466

```
467       void SCAGetReference(wchar_t *referenceName, SCAREF *referenceToken,
468                              int *compCode, int *reason);
469       void SCAInvoke(SCAREF referenceToken, wchar_t *operationName,
470                              int inputMsgLen, void *inputMsg,
471                              int outputMsgLen, void *outputMsg, int *compCode, int *reason);
```

472

473 The following shows a sample of how a service is called in a C component implementation.

474

```
475       #include "SCA.h"
476
477       void clientFunction()
478       {
479
480           SCAREF serviceToken;
481           int compCode, reason;
482           long custNum = 1234;
483           short rating;
484
485           ...
486           SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
487           SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),
488                              (void *)&custNum, sizeof(rating), (void *)&rating,
489                              &compCode, &reason);
490
491       }
```

492

493 If a reference has multiple targets, the client has to use `SCAGetReferences()` to retrieve tokens for
494 each of the tokens and then invoke the operation(s) for each target. For example:

495

```
496       SCAREF *tokens;
497       int num_targets;
498       ...
499       myFunction(...) {
500           int compCode, reason;
501           ...
502           SCAGetReferences(L"myReference", &tokens, &num_targets, &compCode,
503                              &reason);
504           for (i = 0; i < num_targets; i++)
505           {
506               SCAInvoke(tokens[i], L"myOperation", sizeof(inputMsg),
507                              (void *)&inputMsg, 0, NULL, &compCode, &reason);
508           };
509       };
```

510 3.2 Accessing Services from non-SCA component implementations

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

513

514 The following shows a sample of how a service is called in non-SCA C code.

515

```
516 #include "SCA.h"
517
518 void externalFunction()
519 {
520     SCAREF serviceToken;
521     int compCode, reason;
522     long custNum = 1234;
523     short rating;
524
525     SCAEntryPoint(L"customerService", L"http://example.com/mydomain",
526                   &serviceToken, &compCode, &reason);
527     SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),
528               (void *)&custNum, sizeof(rating), (void *)&rating,
529               &compCode, &reason);
530 }
```

531

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

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

- 535 • componentName/serviceName/bindingName

536 3.3 Calling Service Operations

537 The previous sections show the various options for getting access to a service and using SCAInvoke ()
538 to invoke operations of that service.

539 If you have access to a service whose interface is marked as remotable, then on calls to operations of
540 that service you will experience remote semantics. Arguments and return values are passed by-value and
541 it is possible to get a SCA_SERVICE_UNAVAILABLE reason code which is a Runtime error.

542 3.3.1 Proxy Functions

543 It is more natural to use specific function calls than the generic SCAInvoke() API for invoking operations.
544 An SCA runtime typically needs to be involved when a client invokes on operation, particularly if the
545 service is remote. Proxy functions provide a mechanism for using specific function calls and still allow the
546 necessary SCA runtime processing. However, proxies require generated code and managing additional
547 source files, so use of proxies is not always desirable.

548 For SCA, proxy functions have the form:

```
549 <functionReturn> SCA_<functionName>(<SCAREF referenceToken,>
550                                     <functionParameters> )
```

551 where:

- 552 • **<functionName>** is the name of interface function
- 553 • **<functionParameters>** are the parameters of the interface function
- 554 • **<functionReturn>** is the return type of the interface function

555

556 Proxy functions can set `errno` to one of the following values:

557 • ENOENT if a remote service is unavailable
 558 • EFAULT if a fault is returned by the operation
 559
 560 The following shows a sample of using a proxy function.
 561
 562 #include "SCA.h"
 563
 564 void clientFunction()
 565 {
 566
 567 SCAREF serviceToken;
 568 int compCode, reason;
 569 long custNum = 1234;
 570 short rating;
 571
 572 ...
 573 SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
 574 errno = 0;
 575 rating = SCA_getCreditRating(serviceToken, custNum);
 576 if (errno) {
 577 /* handle error or fault */
 578 }
 579 else {
 580 ...
 581 }
 582 ...
 583 }
 584
 585 An SCA implementation MAY support proxy functions. [C30001]

586 3.4 Long Running Request-Response Operations

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

596 3.4.1 Asynchronous Invocation

597 The asynchronous style of invocation uses `SCAIInvokeAsync()` which has the same signature as
 598 `SCAIInvoke()` without the `outputMsgLen` or `outputMsg` parameters but with a parameter taking the
 599 address of a haneler function. This API sends the operation request. The handler function has the
 600 signature

```
601       void <handler>(short responseType);
```

602 and is called when the response is ready. The response type indicates if the response is a reply
 603 message or a fault message. The implementation of the handler uses `SCAGetReplyMessage()` or
 604 `SCAGetFaultMessage()` to retrieve the data.

605 For program-based component implementations, the handler parameter is set to an empty string and
 606 when the SCA runtime starts the program to process the response, a call to `SCAService()` returns the
 607 name of the reference and a call to `SCAOperation()` returns the name of the reference operation.

```

608 If proxy functions are supported, for a service operation with signature
609     <return type> <function name>(<parameters>);
610 the asynchronous invocation style includes a proxy function
611     void SCA_<function name>Async(SCAREF, <in_parameters>, void (*)(short));
612 which will set errno to EBUSY if one request is outstanding and another is attempted.
613 The following shows a sample of how the asynchronous invocation style is used in a C component
614 implementation.

615
616 #include "SCA.h"
617 #include "TravelService.h"
618
619 SCAREF serviceToken;
620 int compCode, reason;
621
622 void makeReservationsHandler(short rspType)
623 {
624     struct confirmationData cd;
625     wchar_t *fault, *faultDetails;
626
627     if (rspType == SCA_REPLY_MESSAGE {
628         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
629         ...
630     }
631     else {
632         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
633                             &faultDetails, &compCode, &reason);
634         if (wstrcmp(*fault, L"noFlight") {
635             ...
636         }
637         else {
638             ...
639         }
640     }
641
642     return;
643 }
644
645 void clientFunction()
646 {
647
648     struct itineraryData id;
649
650     ...
651
652     void (*ah)(short) = &makeReservationsHandler;
653
654     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
655
656     SCAInvokeAsync(serviceToken, L"makeReservations", sizeof(itineraryData),
657                   ah, &compCode, &reason);
658
659     return;
660 }

```

661 3.4.2 Polling Invocation

662 The polling style of invocation uses `SCAInvokePoll()` which has the same signature as `SCAInvoke()`
 663 but without the `outputMsgLen` or `outputMsg` parameters. This API sends the operation request. After
 664 the request is sent the client can check to see if a response has been received by using
 665 `SCACheckResponse()` or cancel the request with `SCACancelInvoke()`.

```

666 If proxy functions are supported, for a service operation with signature
667     <return type> <function name>(<parameters>);
668 the polling invocation style includes a proxy function
669     void SCA_<function name>Poll(SCAREF, <in parameters>);
670 which will set errno to EBUSY if one request is outstanding and another is attempted.
671 The following shows a sample of how the polling invocation style is used in a C component
672 implementation.

673
674 #include "SCA.h"
675 #include "TravelService.h"
676
677 void pollingClientFunction()
678 {
679     SCAREF serviceToken;
680     int compCode, reason;
681     short rspType;
682
683     struct itineraryData id;
684     struct confirmationData cd;
685     wchar_t *fault, *faultDetails;
686
687     ...
688
689     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
690
691     SCAInvokePoll(serviceToken, L"makeReservations", sizeof(itineraryData),
692                   &compCode, &reason);
693
694     SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
695     while (!rspType) {
696         // do something, then wait for some time...
697         SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
698     }
699     if (rspType == SCA_REPLY_MESSAGE {
700         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
701         ...
702     }
703     else {
704         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
705                             &faultDetails, &compCode, &reason);
706         if (wstrcmp(*fault, L"noFlight") {
707             ...
708         }
709         else {
710             ...
711         }
712     }
713
714     return;
715 }

```

716 3.4.3 Synchronous Invocation

717 In this style the client uses API SCAInvoke() but the implementation of this API suspends the main
718 thread after the request is made, and in an implementation-dependent manner receives the response,
719 resumes the main thread and returns from the member function call. If proxy functions are supported, the
720 client can call SCA_<function name>() as normal, and again the implementation handles the
721 asynchronous aspects.

722

723 The following shows a sample of how the synchronous invocation style is used in a C component
724 implementation.

```
725
726     #include "SCA.h"
727     #include "TravelService.h"
728
729     void synchronousClientFunction()
730     {
731         SCAREF serviceToken;
732         int compCode, reason;
733
734         struct itineraryData id;
735         struct confirmationData *cd;
736         wchar_t *fault, *faultDetails;
737
738         ...
739
740         SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
741
742         SCAInvoke(serviceToken, L"makeReservations", sizeof(itineraryData),
743                 (void *)&id, sizeof(confirmationData), (void *)&cd,
744                 &compCode, &reason);
745         if (compCode == SCA_FAULT) {
746             ...
747         }
748         else {
749             SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
750                                 &faultDetails, &compCode, &reason);
751             if (wstrcmp(*fault, L"noFlight")) {
752                 ...
753             }
754             else {
755                 ...
756             }
757         }
758
759         return;
760     }
```

761 4 Asynchronous Programming

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

769 4.1 Non-blocking Calls

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

773 Any function that returns `void` and has only by-value parameters can be marked with the
774 `@oneWay="true"` attribute in the interface definition of the service. An operation marked as `oneWay` is
775 considered non-blocking and the SCA runtime MAY use a binding that buffers the requests to the function
776 and sends them at some time after they are made. [C40001]

777 The following snippet shows the component type for a service with the `reportEvent()` function
778 declared as a one-way operation:

```
779
780 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
781   <service name="LoanService">
782     <interface.c header="LoanService.h">
783       <function name="reportEvent" oneWay="true" />
784     </interface.c>
785   </service>
786 </componentType>
```

787

788 SCA does not currently define a mechanism for making non-blocking calls to functions that return values.
789 It is considered to be a best practice that service designers define one-way operations as often as
790 possible, in order to give the greatest degree of binding flexibility to deployers.

791 4.2 Callbacks

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

794 A callback interface is declared by the `@callbackHeader` and `@callbackFunctions` attributes in the
795 interface definition of the service. The following snippet shows the component type for a service
796 `MyService` with the interface defined in `MyService.h` and the interface for callbacks defined in
797 `MyServiceCallback.h`,

```
798
799 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" >
800   <service name="MyService">
801     <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h"/>
802   </service>
803 </componentType>
```

804 **4.2.1 Using Callbacks**

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

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

814

```
815   double requestQuotation(char *productCode,int quantity);  
816  
817   char *getState();  
818   char *getZipCode();  
819   char *getCreditRating();
```

820

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

827 The following code snippet illustrates a possible implementation of the example service.

828

```
829   #include "QuotationCallback.h"  
830   #include "SCA.h"  
831  
832   double requestQuotation(char *productCode,int quantity) {  
833     double price, discount = 0;  
834     char state[3], creditRating[4];  
835     SCAREF callbackRef;  
836     int compCode, reason;  
837  
838     price = getPrice(productQuote, quantity);  
839  
840     SCAGetCallback(L"", &callbackRef, &compCode, &reason);  
841     SCAIInvoke(callbackRef, L"getState", 0, NULL, sizeof(state), state,  
842                         &compCode, &reason);  
843     if (quantity > 1000 && strcmp(state,"FL") == 0)  
844         discount = 0.05;  
845     SCAIInvoke(callbackRef, L"getCreditRating", 0, NULL, sizeof(creditRating),  
846                         creditRating, &compCode, &reason);  
847     if (quantity > 10000 && creditRating[0] == 'A')  
848         discount += 0.05;  
849     SCAResponseCallback(callbackRef, &compCode, &reason);  
850     return price * (1-discount);  
851 }
```

852

853 The code snippet below is taken from the client of this example service. The client's service
854 implementation class implements the functions of the `QuotationCallback` interface as well as those of its
855 own service interface `ClientService`.

856

```
857   #include "QuotationCallback.h"  
858   #include "SCA.h"
```

```

859     char state[3] = "TX", zipCode[6] = "78746", creditRating[3] = "AA";
860
861     aClientFunction() {
862         SCAREF serviceToken;
863         int compCode, reason;
864
865         SCAGetReference(L"quotationService", &serviceToken, &compCode, &reason);
866
867         SCA_requestQuotation(serviceToken, "AB123", 2000);
868     }
869
870     char *getState() {
871         return state;
872     }
873     char *getZipCode() {
874         return zipCode;
875     }
876     char *getCreditRating() {
877         return creditRating;
878     }
879 }
880

```

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

885 4.2.2 Callback Instance Management

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

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

897 4.2.3 Implementing Multiple Bidirectional Interfaces

898 Since it is possible for a single component to implement multiple services, it is also possible for callbacks
 899 to be defined for each of the services that it implements. The service name parameter of
 900 SCAGetCallback() identifies the service for which the callback is to be obtained.

901 5 Error Handling

902 Clients calling service operations will experience business logic errors, and SCA runtime errors.
903 Business logic errors are generated by the implementation of the called service operation. They are
904 handled by client the invoking the operation of the service.
905 SCA runtime errors are generated by the SCA runtime and signal problems in the management of the
906 execution of components, and in the interaction with remote services. The SCA C API includes two return
907 codes on every function, a completion code and a reason code. The reason code is used to provide
908 more detailed information if a function does not complete successfully. Currently the following SCA codes
909 are defined:

910

```
911     /* Completion Codes */  
912     #define SCACC_OK          0  
913     #define SCACC_WARNING      1  
914     #define SCACC_FAULT        2  
915     #define SCACC_ERROR        3  
916  
917     /* Reason Codes */  
918     #define SCA_SERVICE_UNAVAILABLE 1  
919     #define SCA_MULTIPLE_SERVICES 2  
920     #define SCA_DATA_TRUNCATED   3  
921     #define SCA_BUSY            4  
922  
923     /* Response Types */  
924     #define SCA_NO_RESPONSE    0  
925     #define SCA_REPLY_MESSAGE   1  
926     #define SCA_FAULT_MESSAGE   2
```

927

928 Reason codes between 0 and 100 are reserved for use by this specification. Vendor defined reason
929 codes SHOULD start at 101. [C50001]

930 6 C API

931 6.1 SCA Programming Interface

932 The following shows the C interface declarations for synchronous programming.

933

```
934     typedef void *SCAREF;
935
936     void SCAGetReference(wchar_t *referenceName,
937                           SCAREF *referenceToken,
938                           int *compCode,
939                           int *reason);
940
941     void SCAGetReferences(wchar_t *referenceName,
942                           SCAREF **referenceTokens,
943                           int *num_targets,
944                           int *CompCode,
945                           int *Reason);
946
947     void SCAInvoke(SCAREF token,
948                     wchar_t *operationName,
949                     int inputMsgLen,
950                     void *inputMsg,
951                     int outputMsgLen,
952                     void *outputMsg,
953                     int *compCode,
954                     int *reason);
955
956     void SCAPropertyBoolean(wchar_t *propertyName,
957                             char *value,
958                             int *compCode,
959                             int *reason);
960
961     void SCAPropertyByte(wchar_t *propertyName,
962                           int8_t *value,
963                           int *compCode,
964                           int *reason);
965
966     void SCAPropertyBytes(wchar_t *propertyName,
967                           int8_t **value,
968                           int *size,
969                           int *compCode,
970                           int *reason);
971
972     void SCAPropertyChar(wchar_t *propertyName,
973                           wchar_t *value,
974                           int *compCode,
975                           int *reason);
976
977     void SCAPropertyChars(wchar_t *propertyName,
978                           wchar_t **value,
979                           int *size,
980                           int *compCode,
981                           int *reason);
982
983     void SCAPropertyCChar(wchar_t *propertyName,
984                           char *value,
985                           int *compCode,
986                           int *reason);
```

```

987 void SCAPropertyCChars(wchar_t *propertyName,
988                         char **value,
989                         int *size,
990                         int *compCode,
991                         int *reason);
992
993 void SCAPropertyShort(wchar_t *propertyName,
994                         int16_t *value,
995                         int *compCode,
996                         int *reason);
997
998 void SCAPropertyInt(wchar_t *propertyName,
1000                         int32_t *value,
1001                         int *compCode,
1002                         int *reason);
1003
1004 void SCAPropertyLong(wchar_t *propertyName,
1005                         int64_t *value,
1006                         int *compCode,
1007                         int *reason);
1008
1009 void SCAPropertyFloat(wchar_t *propertyName,
1010                         float *value,
1011                         int *compCode,
1012                         int *reason);
1013
1014 void SCAPropertyDouble(wchar_t *propertyName,
1015                         double *value,
1016                         int *compCode,
1017                         int *reason);
1018
1019 void SCAPropertyString(wchar_t *propertyName,
1020                         wchar_t **value,
1021                         int *size,
1022                         int *compCode,
1023                         int *reason);
1024
1025 void SCAPropertyCString(wchar_t *propertyName,
1026                         char **value,
1027                         int *size,
1028                         int *compCode,
1029                         int *reason);
1030
1031 void SCAPropertyStruct(wchar_t *propertyName,
1032                         void **value,
1033                         int *compCode,
1034                         int *reason);
1035
1036 void SCAGetReplyMessage(SCAREF token,
1037                         int *bufferLen,
1038                         char *buffer,
1039                         int *compCode,
1040                         int *reason);
1041
1042 void SCAGetFaultMessage(SCAREF token,
1043                         int *bufferLen,
1044                         wchar_t **faultName,
1045                         char *buffer,
1046                         int *compCode,
1047                         int *reason);
1048
1049 void SCASetFaultMessage(wchar_t *serviceName,
1050                         wchar_t *operationName,

```

```

1051                 wchar_t *faultName,
1052                     int bufferLen,
1053                         char *buffer,
1054                             int *compCode,
1055                                 int *reason);
1056
1057     void SCASelf(wchar_t *serviceName,
1058                     SCAREF *serviceToken,
1059                         int *compCode,
1060                             int *reason);
1061
1062     void SCAGetCallback(wchar_t *serviceName,
1063                     SCAREF *serviceToken,
1064                         int *compCode,
1065                             int *reason);
1066
1067     void SCAResponseCallback(SCAREF serviceToken,
1068                     int *compCode,
1069                             int *reason);
1070
1071     void SCAInvokeAsync(SCAREF token,
1072                     wchar_t *operationName,
1073                         int inputMsgLen,
1074                             void *inputMsg,
1075                                 void (*handler)(short);
1076                                     int *compCode,
1077                                         int *reason);
1078
1079     void SCAInvokePoll(SCAREF token,
1080                     wchar_t *operationName,
1081                         int inputMsgLen,
1082                             void *inputMsg,
1083                                 int *compCode,
1084                                     int *reason);
1085
1086     void SCACheckResponse(SCAREF token,
1087                     short *responseType
1088                         int *compCode,
1089                             int *reason);
1090
1091     void SCACancelInvoke(SCAREF token,
1092                     int *compCode,
1093                         int *reason);
1094
1095     void SCAEntryPoint(wchar_t *serviceURI,
1096                     wchar_t *domainURI,
1097                         SCAREF *serviceToken,
1098                             int *compCode,
1099                                 int *reason);

```

1100

1101 The C synchronous programming interface has the following functions:

1102 **6.1.1 SCAGetReference**

1103 A C component implementation uses `SCAGetReference()` to initialize a Reference before invoking any
1104 operations of the Reference.

Precondition	C component instance is running	
Input Parameter	referenceName	Name of the Reference to initialize
Output Parameters	referenceToken	Token to be used in subsequent <code>SCAInvoke()</code> calls. This will be NULL if <code>referenceName</code> is not defined for

		the component.
	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_SERVICE_UNAVAILABLE if no suitable service exists in the domain SCA_MULTIPLE_SERVICES if the reference is bound to multiple services
Post Condition	If an operational Service exists for the reference, the component instance has a valid token to use for subsequent runtime calls.	

1105 6.1.2 SCAGetReferences

1106 A C component implementation uses `SCAGetReferences()` to initialize a Reference that might be
 1107 bound to multiple Services before invoking any operations of the Reference.

Precondition	C component instance is running	
Input Parameter	referenceName	Name of the Reference to initialize
Output Parameters	referenceTokens	Array of tokens to be used in subsequent <code>SCAInvoke()</code> calls. These will all be NULL if <code>referenceName</code> is not defined for the component. Operations need to be invoked on each token in the array.
	num_targets	Number of tokens returned in the array.
	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_SERVICE_UNAVAILABLE if no suitable service exists in the domain
Post Condition	If operational Services exist for the reference, the component instance has a valid token to use for subsequent runtime calls.	

1108 6.1.3 SCAInvoke

1109 A C component implementation uses `SCAInvoke()` to invoke an operation of an interface.

Precondition	C component instance is running and has a valid token	
Input Parameters	token	Token returned by prior <code>SCAGetReference()</code> or <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
	operationName	Name of the operation to invoke
	inputMsgLen	Length of the request message buffer
	inputMsg	Request message
In/Out Parameter	outputMsgLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message

Output Parameters	outputMsg	Response message
	compCode	<p>SCACC_OK, if the call is successful</p> <p>SCACC_WARNING, if the response data was truncated. The buffer size needs to be increased and SCAGetReplyMessage() called with the larger buffer.</p> <p>SCACC_FAULT, if the operation returned a business fault. SCAGetFaultMessage() needs to be called to get the fault details.</p> <p>SCACC_ERROR, otherwise – see reason for details</p>
	Reason	<p>SCA_DATA_TRUNCATED if the response data was truncated</p> <p>SCA_PARAMETER_ERROR if the operationName is not defined for the interface</p> <p>SCA_SERVICE_UNAVAILABLE if the provider for the interface is no longer operational</p>
Post Condition	Unless a SCA_SERVICE_UNAVAILABLE reason is returned, the token remains valid for subsequent calls.	

1110 6.1.4 SCAProperty<T>

1111 A C component implementation uses SCAProperty<T>() to get the configured value for a Property.
 1112 This API is available for Boolean, Byte, Bytes, Char, Chars, CChar, CChars, Short, Int, Long, Float,
 1113 Double, String, CString and Struct. The Char, Chars, and String variants return wchar_t based data while
 1114 the CChar, CChars, and CString variants return char based data. The Bytes, Chars, and CChars variants
 1115 return a buffer of data. The String and CString variants return a null terminated string.
 1116 An SCA runtime MAY additionally provide a DataObject variant of this API for handling properties with
 1117 complex XML types. The type of the value parameter in this variant is DATAOBJECT. [C60002]
 1118 If <T> is one of: Boolean, Byte, Char, CChar, Short, Int, Long, Float, Double or Struct

Precondition	C component instance is running	
Input Parameter	propertyName	Name of the Property value to obtain
Output Parameters	value	Configured value of the property
	compCode	<p>SCACC_OK, if the call is successful</p> <p>SCACC_ERROR, otherwise – see reason for details</p>
	reason	SCA_PARAMETER_ERROR if the propertyName is not defined for the component or its type is incompatible with <T>
Post Condition	The configured value of the Property is loaded into the appropriate variable.	

1119
 1120 If <T> is one of: Bytes, Chars, CChars, String or CString

Precondition	C component instance is running	
Input Parameter	propertyName	Name of the Property value to obtain
In/Out Parameter	size	Input: Maximum number of bytes or characters that can

		be returned Output: Actual number of bytes or characters returned or size needed to hold entire value
Output Parameters	value	Configured value of the property
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer. SCACC_ERROR, otherwise – see reason for details
	reason	SCACC_WARNING, if the data was truncated SCA_PARAMETER_ERROR if the propertyName is not defined for the component or its type is incompatible with <T>
Post Condition	The configured value of the Property is loaded into the appropriate variable.	

1121 6.1.5 SCAGetReplyMessage

1122 A C component implementation uses `SCAGetReplyMessage()` to retrieve the reply message of an
1123 operation invocation if the length of the message exceeded the buffer size provided on `SCAInvoke()`.

Precondition	C component instance is running, has a valid token and an <code>SCAInvoke()</code> returned a <code>SCACC_WARNING</code> compCode or has a valid serviceToken and an <code>SCACallback()</code> returned a <code>SCACC_WARNING</code> compCode	
Input Parameter	token	Token returned by prior <code>SCAGetReference()</code> , <code>SCAGetReferences()</code> , <code>SCASelf()</code> , or <code>SCAGetCallback()</code> call.
In/Out Parameter	bufferLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	buffer	Response message
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the fault data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer. SCACC_ERROR, otherwise – see reason for details
	reason	<code>SCA_DATA_TRUNCATED</code> if the fault data was truncated.
Post Condition	The <code>referenceToken</code> remains valid for subsequent calls.	

1124 6.1.6 SCAGetFaultMessage

1125 A C component implementation uses `SCAGetFaultMessage()` to retrieve the details of a business fault
1126 received in response to an operation invocation.

Precondition	C component instance is running, has a valid token and an <code>SCAInvoke()</code> returned a <code>SCACC_FAULT</code> compCode
--------------	---

Input Parameter	token	Token returned by prior SCAGetReference(), SCAGetReferences(), SCASelf() or SCAGetCallback() call.
In/Out Parameter	bufferLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	faultName	Name of the business fault
	Buffer	Fault message
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the fault data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer. SCACC_ERROR, otherwise – see reason for details
	Reason	SCA_DATA_TRUNCATED if the fault data was truncated. SCA_PARAMETER_ERROR if the last operation invoked on the Reference did return a business fault
Post Condition	The referenceToken remains valid for subsequent calls.	

1127 **6.1.7 SCASetFaultMessage**

1128 A C component implementation uses SCASetFaultMessage() to return a business fault in response to
1129 a request.

Precondition	C component instance is running	
Input Parameters	serviceName	Name of the Service of the component for which the fault is being returned
	operationName	Name of the operation of the Service for which the fault is being returned
	faultName	Name of the business fault
	bufferLen	Length of the fault message buffer
	buffer	Fault message
Output Parameters	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_PARAMETER_ERROR if the serviceName is not defined for the component, operationName is not defined for the Service or the faultName is not defined for the operation
Post Condition	No change	

1130 **6.1.8 SCASelf**

1131 A C component implementation uses SCASelf() to access a Service it provides.

Precondition	C component instance is running	
Input Parameter	serviceName	Name of the Service to access. If a component only provides one service, this string can be empty.
Output Parameters	serviceToken	Token to be used in subsequent <code>SCAIInvoke()</code> calls. This will be NULL if <code>serviceName</code> is not defined for the component.
	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_PARAMETER_ERROR if the <code>serviceName</code> is not defined for the component
Post Condition	The component instance has a valid token to use for subsequent calls.	

1132 6.1.9 SCAGetCallback

1133 A C component implementation uses `SCAGetCallback()` to initialize a Service before invoking any callback operations of the Service.

Precondition	C component instance is running	
Input Parameter	serviceName	Name of the Service to initialize. If a component only provides one service, this string can be empty.
Output Parameters	serviceToken	Token to be used in subsequent <code>SCAIinvoke()</code> calls. This will be NULL if <code>serviceName</code> is not defined for the component.
	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_SERVICE_UNAVAILABLE if client is no longer available in the domain
Post Condition	If callback interface is defined for the Service, the component instance has a valid token to use for subsequent callbacks.	

1135 6.1.10 SCAResleaseCallback

1136 A C component implementation uses `SCAResleaseCallback()` to tell the SCA runtime it has completed callback processing and the EndPointReference can be released.

Precondition	C component instance is running and has a valid serviceToken	
Input Parameter	serviceToken	Token returned by prior <code>SCAGetCallback()</code> call.
Output Parameters	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_PARAMETER_ERROR if the <code>serviceToken</code> is not valid
Post Condition	The token becomes invalid for subsequent calls.	

1138 **6.1.11 SCAInvokeAsync**

1139 A C component implementation uses `SCAInvokeAsync()` to invoke a long running operation of an
 1140 interface using the asynchronous style.

Precondition	C component instance is running and has a valid token	
Input Parameters	token	Token returned by prior <code>SCAGetReference()</code> , <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
	operationName	Name of the operation to invoke
	inputMsgLen	Length of the request message buffer
	inputMsg	Request message
	handler	Address of the function to handle the asynchronous response.
Output Parameters	compCode	<code>SCACC_OK</code> , if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
	reason	<code>SCA_BUSY</code> if an operation is already outstanding for this Reference or Callback <code>SCA_PARAMETER_ERROR</code> if the <code>operationName</code> is not defined for the interface <code>SCA_SERVICE_UNAVAILABLE</code> if for the provider of the interface is no longer operational
Post Condition	Unless a <code>SCA_SERVICE_UNAVAILABLE</code> reason is returned, the token remains valid for subsequent calls.	

1141 **6.1.12 SCAInvokePoll**

1142 A C component implementation uses `SCAInvokePoll()` to invoke a long running operation of a
 1143 Reference using the polling style.

Precondition	C component instance is running and has a valid token	
Input Parameters	token	Token returned by prior <code>SCAGetReference()</code> , <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
	operationName	Name of the operation to invoke
	inputMsgLen	Length of the request message buffer
	inputMsg	Request message
	compCode	<code>SCACC_OK</code> , if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
Output Parameters	reason	<code>SCA_BUSY</code> if an operation is already outstanding for this Reference or Callback <code>SCA_PARAMETER_ERROR</code> if the <code>operationName</code> is not defined for the interface <code>SCA_SERVICE_UNAVAILABLE</code> if provider of the

		interface is no longer operational
Post Condition	Unless a <code>SCA_SERVICE_UNAVAILABLE</code> reason is returned, the token remains valid for subsequent calls.	

1144 6.1.13 SCACheckResponse

1145 A C component implementation uses `SCACheckResponse()` to determine if a response to a long
1146 running operation request has been received.

Precondition	C component instance is running, has a valid token and has made a <code>SCAInvokePoll()</code> but has not received a response.	
Input Parameter	token	Token returned by prior <code>SCALocate()</code> , <code>SCALocateMultiple()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
Output Parameters	responseType	Type of response received
	compCode	<code>SCACC_OK</code> if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
	reason	<code>SCA_PARAMETER_ERROR</code> if there is no outstanding operation for this Reference or Callback
Post Condition	No change	

1147 6.1.14 SCACancelInvoke

1148 A C component implementation uses `SCACancelInvoke()` to cancel a long running operation request.

Precondition	C component instance is running, has a valid token and has made a <code>SCAInvokeAsync()</code> or <code>SCAInvokePoll()</code> but has not received a response.	
Input Parameter	token	Token returned by prior <code>SCALocate()</code> , <code>SCALocateMultiple()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
Output Parameters	compCode	<code>SCACC_OK</code> , if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
	reason	<code>SCA_PARAMETER_ERROR</code> if there is no outstanding operation for this Reference or Callback
Post Condition	If a response is subsequently received for the operation, it will be discarded.	

1149 6.1.15 SCAEntryPoint

1150 Non-SCA C code uses `SCAEntryPoint()` to access a Service before invoking any operations of the
1151 Service.

Precondition	None	
Input Parameter	serviceURI	URI of the Service to access
	domainURI	URI of the SCA domain
Output Parameters	serviceToken	Token to be used in subsequent <code>SCAInvoke()</code> calls.

		This will be NULL if the Service cannot be found.
	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_SERVICE_UNAVAILABLE if the domain does not exist or the service does not exist in the domain
Post Condition	If the Service exists in the domain, the client has a valid token to use for subsequent runtime calls.	

6.2 Program-Based Implementation Support

A SCA runtime MAY provide the functions `SCAService()`, `SCAOperation()`, `SCAMessageIn()` and `SCAMessageOut()` to support C implementations in programs. [C60003]

```

1155
1156     void SCAService(wchar_t *serviceName, int *compCode, int *reason);
1157
1158     void SCAOperation(wchar_t *operationName, int *compCode, int *reason);
1159
1160     void SCAMessageIn(wchar_t *serviceName,
1161                         wchar_t *operationName,
1162                         int *bufferLen,
1163                         void *buffer,
1164                         int *compCode,
1165                         int *reason);
1166
1167     void SCAMessageOut(wchar_t *serviceName,
1168                         wchar_t *operationName,
1169                         int bufferLen,
1170                         void *buffer,
1171                         int *CompCode,
1172                         int *Reason);

```

1173

1174 The C program-based implementation support has the following functions:

6.2.1 SCAService

1176 A program-based C component implementation uses `SCAService()` to determine which service was
1177 used to invoke it.

Precondition	C component instance is running	
Output Parameters	serviceName	Name of the service used to invoke the component
	compCode	SCACC_OK
	reason	
Post Condition	No change	

6.2.2 SCAOperation

1179 A program-based C component implementation uses `SCAOperation()` to determine which operation of
1180 a Service was used to invoke it.

Precondition	C component instance is running
--------------	---------------------------------

Output Parameters	operationName	Name of the operation used to invoke the component
	compCode	SCACC_OK
	reason	
Post Condition	Component has sufficient information to select proper processing branch.	

1181 **6.2.3 SCAMessageIn**

1182 A program-based C component implementation uses `SCAMessageIn()` to retrieve its request message.

Precondition	C component instance is running, and has determined its invocation Service and operation	
Input Parameters	serviceName	Name returned by <code>SCAService()</code> .
	operationName	Name returned by <code>SCAOperation()</code> .
In/Out Parameter	bufferLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	buffer	Request message
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the request data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer.
	reason	SCA_DATA_TRUNCATED if the request data was truncated.
Post Condition	The component is ready to begin processing.	

1183 **6.2.4 SCAMessageOut**

1184 A program-based C component implementation uses `SCAMessageOut()` to return a reply message.

Precondition	C component instance is running	
Input Parameters	serviceName	Name returned by <code>SCAService()</code> .
	operationName	Name returned by <code>SCAOperation()</code> .
	bufferLen	Length of the reply message buffer
	buffer	Reply message
Output Parameters	compCode	SCACC_OK
	reason	
Post Condition	The component normally ends processing.	

1185 7 C Contributions

1186 Contributions are defined in the Assembly specification [ASSEMBLY] C contributions are typically, but
1187 not necessarily contained in .zip files. In addition to SCDL and potentially WSDL artifacts, C contributions
1188 include binary executable files, componentType files and potentially C interface headers. No additional
1189 discussion is needed for header files, but here are some additional considerations for executable and
1190 componentType files discussed in the following sections.

1191 7.1 Executable files

1192 Executable files containing the C implementations for a contribution can be contained in the contribution,
1193 contained in another contribution or external to any contribution. In some cases, it could be desirable to
1194 have contributions share an executable. In other cases, an implementation deployment policy might
1195 dictate that executables are placed in specific directories in a file system.

1196 7.1.1 Executable in contribution

1197 When the executable file containing a C implementation is in the same contribution, the `@path` attribute of
1198 the `implementation.c` element is used to specify the location of the executable. The specific location of an
1199 executable within a contribution is not defined by this specification.

1200 The following shows a contribution containing a DLL.

```
1201
1202     META-INF/
1203         sca-contribution.xml
1204     bin/
1205         autoinsurance.dll
1206     AutoInsurance/
1207         AutoInsurancecomposite
1208         AutoInsuranceService/
1209             AutoInsurance.h
1210             AutoInsurance.componentType
1211         include/
1212             Customers.h
1213             Underwriting.h
1214             RateUtils.h
```

1215 The SCDL for the AutoInsuranceService component is:

```
1216
1217
1218 <component name="AutoInsuranceService">
1219     <implementation.c module="autoinsurance" path="bin/">
1220         componentType="AutoInsurance" />
1221 </component>
```

1222 7.1.2 Executable shared with other contribution(s) (Export)

1223 If a contribution contains an executable that also implements C components found in other contributions,
1224 the contribution has to export the executable. An executable in a contribution is made visible to other
1225 contributions by adding an `export.c` element to the contribution definition as shown in the following
1226 snippet.

```
1227
1228 <contribution>
1229     <deployable composite="myNS:RateUtilities">
1230         <export.c name="contribNS:rates" >
1231     </contribution>
```

1232 It is also possible to export only a subtree of a contribution. If a contribution contains the following:

```
1233  
1234     META-INF/  
1235         sca-contribution.xml  
1236     bin/  
1237         rates.dll  
1238     RateUtilities/  
1239         RateUtilities.composite  
1240         RateUtilitiesService/  
1241             RateUtils.h  
1242             RateUtils.componentType
```

1243

1244 An export of the form:

1245

```
1246     <contribution>  
1247         <deployable composite="myNS:RateUtilities"  
1248             <export.c name="contribNS:ratesbin" path="bin/" >  
1249     </contribution>
```

1250

1251 only makes the contents of the bin directory visible to other contributions. By placing all of the executable
1252 files of a contribution in a single directory and exporting only that directory, the amount of information
1253 contribution that uses the exported executable files is limited. This is considered a best practice.

1254 7.1.3 Executable outside of contribution (Import)

1255 When the executable that implements a C component is located outside of a contribution, the contribution
1256 MUST import the executable. If the executable is located in another contribution, the **import.c** element of
1257 the contribution definition uses a *@location* attribute that identifies the name of the export as defined in
1258 the contribution that defined the export as shown in the following snippet.

1259

```
1260     <contribution>  
1261         <deployable composite="myNS:Underwriting"  
1262             <import.c name="rates" location="contribNS:rates">  
1263     </contribution>
```

1264

1265 The SCDL for the UnderwritingService component is:

1266

```
1267     <component name="UnderwritingService">  
1268         <implementation.c module="rates" path="rates:bin/"  
1269             componentType="Underwriting" />  
1270     </component>
```

1271

1272 If the executable is located in the file system, the *@location* attribute identifies the location in the files
1273 system used as the root of the import as shown in this snippet.

1274

```
1275     <contribution>  
1276         <deployable composite="myNS:CustomerUtilities"  
1277             <import.c name="usr-bin" location="/usr/bin/" >  
1278     </contribution>
```

1279 7.2 componentType files

1280 As stated in section 2.5, each component implemented in C has a corresponding componentType file.
1281 This componentType file is, by default, located in the root directory of the composite containing the

1282 component or a subdirectory of the composite root with a name specified on the `@componentType`
1283 attribute as shown in the following example.

1284

```
1285     META-INF/
1286         sca-contribution.xml
1287     bin/
1288         autoinsurance.dll
1289     AutoInsurance/
1290         AutoInsurancecomposite
1291         AutoInsuranceService/
1292             AutoInsurance.h
1293             AutoInsurance.componentType
```

1294

1295 The SCDL for the AutoInsuranceService component is:

1296

```
1297 <component name="AutoInsuranceService">
1298     <implementation.c module="autoinsurance" path="bin/">
1299         componentType="AutoInsurance" />
1300 </component>
```

1301

1302 Since there is a one-to-one correspondence between implementations and componentTypes, when an
1303 implementation is shared between contributions, it is desirable to also share the componentType file.
1304 ComponentType files can be exported and imported in the same manner as executable files. The
1305 location of a `.componentType` file can be specified using the `@componentType` attribute of the
1306 `implementation.c` element.

1307

```
1308 <component name="UnderwritingService">
1309     <implementation.c library="rates" path="rates:bin/">
1310         componentType="rates:types/Underwriting" />
1311 </component>
```

1312 7.3 C Contribution Extensions

1313 7.3.1 Export.c

1314 The following snippet shows the schema for the C export element used to make an executable or
1315 componentType file visible outside of a contribution.

1316

```
1317 <?xml version="1.0" encoding="ASCII"?>
1318 <!-- export.c schema snippet -->
1319 <export.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
1320     name="QName" path="string"? >
```

1321

1322 The `export.c` element has the following **attributes**:

- **name : QName (1..1)** – name of the export. The `@name` attribute of a `<export.c/>` element MUST be unique amongst the `<export.c/>` elements in a domain. [C70001]
- **path : string (0..1)** – path of the exported executable relative to the root of the contribution. If not present, the entire contribution is exported.

1327 7.3.2 Import.c

1328 The following snippet shows the schema for the C import element used to reference an executable or
1329 componentType file that is outside of a contribution.

```
1330     <?xml version="1.0" encoding="ASCII"?>
1331     <!-- import.c schema snippet -->
1332     <import.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1333         name="QName" location="string" >
```

1334

1335 The ***import.c*** element has the following ***attributes***:

- ***name : QName (1..1)*** – name of the import. The @name attribute of a ***<import.c>*** child element of a ***<contribution>*** MUST be unique amongst the ***<import.c>*** elements in of that contribution. [C70002]
- ***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.

1340 8 Types Supported in Service Interfaces

1341 A service interface can support a restricted set of the types available to a C programmer. This section
1342 summarizes the valid types that can be used.

1343 8.1 Local service

1344 The return type and types of the parameters of a function of a local service interface MUST be one of:

- 1345 • Any of the C primitive types (for example, int, short, char). In this case the data will be passed by
1346 value as is normal for C.
- 1347 • Pointers to any of the C primitive types (for example, int *, short *, char *).
- 1348 • DATAOBJECT. An SDO handle. [C80001]

1349 8.2 Remotable service

1350 For a remotable service being called by another service the data exchange semantics is by-value. The
1351 return type and types of the parameters of a function of a remotable service interface MUST be one of:

- 1352 • Any of the C primitive types (for example, int, short, char). This will be copied.
 - 1353 • DATAOBJECT. An SDO handle. The SDO will be copied and passed to the destination. [C80002]
- 1354 Unless the interface is marked as allowing pass by reference semantics, the behavior of the following are
1355 not defined:
- 1356 • Pointers.

1357

9 Restrictions on C header files

1358 A C header file that is used to describe an interface has some restrictions.

1359 A C header file used to define an interface MUST:

- Declare at least one function or message format struct [C90001]

1361 A C header file used to define an interface MUST NOT use the following constructs:

- 1362 • Macros [C90002]

1363 10 WSDL to C and C to WSDL Mapping

1364 The SCA Client and Implementation Model for C applies the principles of the WSDL to Java and Java to
1365 WSDL mapping rules (augmented and interpreted for C as detailed in the following section) defined in the
1366 JAX-WS specification [JAXWS21] for generating remotable C interfaces from WSDL portTypes and vice
1367 versa. Use of the JAX-WS specification as a guideline for WSDL to C and C to WSDL mappings does not
1368 imply that any support for the Java language is mandated by this specification.

1369 For the mapping from C types to XML schema types SCA supports the SDO 2.1 [SDO21] mapping. A
1370 detailed mapping of C to WSDL types and WSDL to C types is covered in Data Binding.

1371 The following general rules apply to the application of JAX-WS to C:

- 1372 • References to Java are considered references to C.
- 1373 • References to Java classes are considered references to a collection of C functions that
1374 implement an interface.
- 1375 • References to Java methods are considered references to C functions.
- 1376 • References to Java interfaces are considered references to a collection of C function declarations
1377 used to define an interface.
- 1378 • For the purposes of the C-to-WSDL mapping algorithm, a C header file with containing function
1379 declarations and no annotations is treated as if it had a @WebService annotation. All default
1380 values are assumed for the @WebService annotation.

1381 10.1 Interpretations for WSDL to C Mapping

1382 External binding files are not supported.

1383 For dispatching functions or invoking programs and marshalling data, an implementation can choose to
1384 interpret the WSDL document, possibly containing mapping customizations, at runtime or interpret the
1385 document as part of the deployment process generating implementation specific artifacts that represent
1386 the mapping.

1387 10.1.1 Definitions

1388 Since C has no namespace or package construct, the targetNamespace of a WSDL document is ignored
1389 by the mapping.

1390 MIME binding is not supported.

1391 10.1.2 PortType

1392 A portType maps to a set of declarations that form the C interface for the service. The form of these
1393 declarations depends on the type of the service implementation.

1394 If the implementation is a library, the declarations are one or more function declarations and potentially
1395 any necessary struct declarations corresponding to any complex XML schema types needed by
1396 messages used by operations of the portType. See Complex Content Binding for options for complex
1397 type mapping.

1398 If the implementation is contained in a program, the declarations are all struct declarations. See the next
1399 section for details.

1400 In the absence of customizations, an SCA implementation SHOULD map each portType to separate
1401 header file. An SCA implementation MAY use any sca-c:prefix binding declarations to control this
1402 mapping. [C100001] For example, all portTypes in a WSDL document with a common sca-c:prefix binding
1403 declaration could be mapped to a single header file..

1404 Header file naming is implementation dependent.

1405 10.1.3 Operations

1406 Asynchronous mapping is not supported.

1407 10.1.3.1 Operation Names

1408 WSDL operation names are only guaranteed to be unique with a portType. C requires function and struct
1409 names loaded into an address space to be distinct. The mapping of operation names to function or struct
1410 names have to take this into account.

1411 For components implemented in libraries, in the absence of customizations, an SCA implementation
1412 MUST concatenate the portType name, with the first character converted to lower case, and the operation
1413 name, with the first character converted to upper case, to form the function. [C100002]

1414 An application can customize this mapping using the sca-c:prefix and/or sca-c:function binding
1415 declarations.

1416 For program-based service implementations:

- If the number of **In** parameters plus the number of **In/Out** parameters is greater than one there will be
1418 a request struct.

1419 • If the number of **Out** parameters plus the number of **In/Out** parameters is greater than one there will
1420 be a response struct.

1421 For components implemented in a program, in the absence of customizations, an SCA implementation
1422 MUST concatenate the portType name, with the first character converted to lower case, and the operation
1423 name, with the first character converted to upper case, to form the request struct name. Additionally an
1424 SCA implementation MUST append “*Response*” to the request struct name to form the response struct
1425 name. [C100005]

1426 An application can customize this mapping using the sca-c:prefix and/or sca-c:struct binding declarations.

1427 10.1.3.2 Message and Part

1428 In the absence of any customizations for a WSDL operation that does not meet the requirements for the
1429 wrapped style, the name of a mapped function parameter or struct member MUST be the value of the
1430 name attribute of the wsdl:part element with the first character converted to lower case. [C100003]

1431 In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped
1432 style, the name of a mapped function parameter or struct member MUST be the value of the local name
1433 of the wrapper child with the first character converted to lower case. [C100004]

1434 An application can customize this mapping using the sca-c:parameter binding declaration.

1435 For library-based service implementations, an SCA implementation MUST map **In** parameters as pass
1436 by-value and **In/Out** and **Out** parameters as pass via pointers. [C100019]

1437 For program-based service implementations, an SCA implementation MUST map all values in the input
1438 message as pass by-value and the updated values for **In/Out** parameters and all **Out** parameters in the
1439 response message as pass by-value. [C100020]

1440 10.1.4 Types

1441 As per section Data Binding (based on SDO type mapping).

1442 MTOM/XOP content processing is left to the application.

1443 10.1.5 Fault

1444 C has no exceptions so an API is provided for getting and setting fault messages (see
1445 SCAGetFaultMessage and SCASetFaultMessage). Fault messages are mapped in same manner as
1446 input and output messages.

1447 In the absence of customizations, an SCA implementation MUST map the name of the message element
1448 referred to by a fault element to name of the struct describing the fault message content. If necessary, to

1449 avoid name collisions, an implementation MAY append “*Fault*” to the name of the message element when
1450 mapping to the struct name. [C100006]

1451 An application can customize this mapping using the sca-c:struct binding declaration.

1452 **10.1.6 Service and Port**

1453 This mapping does not define generation of client side code.

1454 **10.1.7 XML Names**

1455 See comments in Operations

1456 **10.2 Interpretations for C to WSDL Mapping**

1457 **10.2.1 Package**

1458 Not relevant.

1459 An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be
1460 configurable. [C100007]

1461 **10.2.2 Class**

1462 Not relevant since mapping is only based on declarations.

1463 **10.2.3 Interface**

1464 The declarations in a header file are used to define an interface. A header file can be used to define an
1465 interface if it satisfies either (for components implemented in libraries):

- 1466 • Contains one or more function declarations
 - 1467 • Any of these functions declarations might carry a @WebFunction annotation
 - 1468 • The parameters and return types of these function declarations are compatible with the C to XML
1469 Schema mapping in Data Binding
- 1470 or (for components implemented in programs):
- 1471 • Contains one request message struct declarations
 - 1472 • Any of the request message struct declarations might carry a @WebOperation annotation
 - 1473 • Any of the request message struct declarations can have a corresponding response message struct,
1474 identified by either having a name with “Response” appended to the request message struct name or
1475 identified in a @WebOperation annotation
 - 1476 • Members of these struct declarations are compatible with the C to XML Schema mapping in Data
1477 Binding

1478 In the absence of customizations, an SCA implementation MUST map the header file name to the
1479 portType name. An implementation MAY append “*PortType*” to the header file name in the mapping to
1480 the portType name. [C100008]

1481 An application can customize this mapping using the @WebService annotation.

1482 **10.2.4 Method**

1483 For components implemented in libraries, functions map to operations.

1484 In the absence of customizations, an SCA implementation MUST map the function name to the operation
1485 name, stripping the portType name, if present, and any namespace prefix from the function name from
1486 the front of function name before mapping it to the operation name. [C100009]

1487 An application can customize function to operation mapping or exclude a function from an interface using
1488 the @WebFunction annotation.
1489 For components implemented in programs, operations are mapped from request structs.
1490 In the absence of customizations, a struct with a name that does not end in “Response” or “Fault” is
1491 considered to be a request message struct and an SCA implementation MUST map the struct name to
1492 the operation name, stripping the portType name, if present, and any namespace prefix from the front of
1493 the struct name before mapping it to the operation name. [C100010]
1494 An application can customize stuct to operation mapping or exclude a struct from an interface using the
1495 @WebOperation annotation.

1496 **10.2.5 Method Parameters and Return Type**

1497 For components implemented in libraries, function parameters and return type map to either message or
1498 global element components.
1499 In the absence of customizations, an SCA implementation MUST map the parameter name, if present, to
1500 the part or global element component name. If the parameter does not have a name the SCA
1501 implementation MUST use argN as the part or global element child name. [C100011]
1502 An application can customize parameter to message or global element component mapping using the
1503 @WebParam annotation.
1504 In the absence of customizations, an SCA implementation MUST map the return type to a part or global
1505 element child named “return”. [C100012]
1506 An application can customize return type to message or global element component mapping using the
1507 @WebReturn annotation.
1508 An SCA implementation MUST map:
1509

- a function’s return value as an **out** parameter.
- by-value and const parameters as **in** parameters.
- in the absence of customizations, pointer parameters as **in/out** parameters. [C100017]

1512 An application can customize parameter classification using the @WebParam annotation.
1513 Program based implementation SHOULD use the Document-Literal style and encoding. [C100013]
1514 In the absence of customizations, an SCA implementation MUST map the struct member name to the
1515 part or global element child name. [C100014]
1516 An application can customize struct member to message or global element component mapping using the
1517 @WebParam annotation.
1518

- Members of the request struct that are not members of the response struct are **in** parameters
- Members of the response struct that are not members of the request struct are **out** parameters
- Members of both the request and response structs are **in/out** parameters. Matching is done by
1521 member name. An SCA implementation MUST ensure that **in/out** parameters have the same type in
1522 the request and response structs. [C100015]

1523 **10.2.6 Service Specific Exception**

1524 C has no exceptions. A struct can be annotated as a fault message type. A function or operation
1525 declaration can be annotated to indicate that it potentially generates a specific fault.
1526 An application can define a fault message format using the @WebFault annotation.
1527 An application can indicate that a WSDL fault might be generated by a function or operation using the
1528 @WebThrows annotation.

1529 **10.2.7 Generics**

1530 Not relevant.

1531 **10.2.8 Service and Ports**

1532 An SCA runtime invokes function (or programs) as a result of receiving an operation request. No
1533 mapping to Service or Ports is defined by this specification.

1534 **10.3 Data Binding**

1535 The data in wsdl:parts or wrapper children is mapped to and from C function parameters and return
1536 values (for library-based component implementations), or struct members (for program-based component
1537 implementations and fault messages).

1538 **10.3.1 Simple Content Binding**

1539 The mapping between XSD simple content types and C types follows the convention defined in the SDO
1540 specification **[SDO21]**. The following table summarizes that mapping as it applies to SCA services.

1541

XSD Schema Type →	C Type	→ XSD Schema Type
anySimpleType	wchar_t *	string
anyType	DATAOBJECT	anyType
anyURI	wchar_t *	string
base64Binary	char *	string
boolean	char	string
byte	int8_t	byte
date	wchar_t *	string
dateTime	wchar_t *	string
decimal	wchar_t *	string
double	double	double
duration	wchar_t *	string
ENTITIES	wchar_t *	string
ENTITY	wchar_t *	string
float	float	float
gDay	wchar_t *	string
gMonth	wchar_t *	string
gMonthDay	wchar_t *	string
gYear	wchar_t *	string
gYearMonth	wchar_t *	string
hexBinary	char *	string
ID	wchar_t *	string
IDREF	wchar_t *	string

IDREFS	wchar_t *	string
int	int32_t	int
integer	wchar_t *	string
language	wchar_t *	string
long	int64_t	long
Name	wchar_t *	string
NCName	wchar_t *	string
negativeInteger	wchar_t *	string
NMTOKEN	wchar_t *	string
NMTOKENS	wchar_t *	string
nonNegativeInteger	wchar_t *	string
nonPositiveInteger	wchar_t *	string
normalizedString	wchar_t *	string
NOTATION	wchar_t *	string
positiveInteger	wchar_t *	string
QName	wchar_t *	string
short	int16_t	short
string	wchar_t *	string
time	wchar_t *	string
token	wchar_t *	string
unsignedByte	uint8_t	unsignedByte
unsignedInt	uint32_t	unsignedInt
unsignedLong	uint64_t	unsignedLong
unsignedShort	uint16_t	unsignedShort

1542 *Table 1: XSD simple type to C type mapping*

1543

C Type →	XSD Schema Type
_Bool	boolean
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	time
struct tm	dateTime

1544 *Table 2: C type to XSD type mapping*

1545

1546 The C standard does not define value ranges for integer types so it is possible that on a platform
 1547 parameters or return values could have values that are out of range for the default XSD schema type. In
 1548 these circumstances, the mapping would need to be customized, using @WebParam or @WebResult if
 1549 supported, or some other implementation-specific mechanism.

1550 An SCA implementation MUST map simple types as defined in Table 1 and Table 2 by default. [C100021]

1551 An SCA implementation MAY map boolean to _Bool by default. [C100022]

1552 10.3.1.1 WSDL to C Mapping Details

1553 In general, when xsd:string and types derived from xsd:string map to a struct member, the
 1554 mapping is to a combination of a wchar_t * and a separately allocated data array. If either the length
 1555 or maxLength facet is used, then a wchar_t[] is used. If the pattern facet is used, this might allow
 1556 the use of char and/or also constrain the length.

1557 Example:

1558 <xsd:element name="myString" type="xsd:string"/>
 1559 maps to:

1560 wchar_t *myString;
 1561 /* this points to a dynamically allocated buffer with the data */
 1562
 1563 <xsd:simpleType name="boundedString25">
 1564 <xsd:restriction base="xsd:string">
 1565 <xsd:length value="25"/>
 1566 </xsd:restriction>
 1567 </xsd:simpletype>
 1568 ...
 1569 <xsd:element name="myString" type="boundedString25"/>

1570 maps to:

1571 wchar_t myString[26];

- 1572
- When unbounded binary data maps to a struct member, the mapping is to a char * that points to the location where the actual data is located. Like strings, if the binary data is bounded in length, a char[] is used.

1576

1577 Examples:

1578 <xsd:element name="myData" type="xsd:hexBinary"/>

1579 maps to:

1580 char *myData;

1581 /* this points to a dynamically allocated buffer with the data */

1582

1583 <xsd:simpleType name="boundedData25">

1584 <xsd:restriction base="xsd:hexBinary">

1585 <xsd:length value="25"/>

1586 </xsd:restriction>

1587 </xsd:simpleType>

1588 ...

1589 <xsd:element name="myData" type="boundedData25"/>

1590 maps to:

1591 char myData[26];

1592

- Since C does not have a way of representing unset values, when elements with `minOccurs != maxOccurs` and lists with `minLength != maxLength`, which have a variable, but bounded, number of instances, map to a struct, the mapping is to a count of the number of occurrences and an array. If the count is 0, then the contents of the array is undefined.

1593

1598 Examples:

1599 <xsd:element name="counts" type="xsd:int" maxOccurs="5"/>

1600 maps to:

1601 size_t counts_num;

1602 int counts[5];

1603

1604 <xsd:simpleType name="lineNumList">

1605 <xsd:list itemType="xsd:int"/>

1606 </xsd:simpleType>

1607 <xsd:simpleType name="lineNumList6">

1608 <xsd:restriction base="lineNumList ">

1609 <xsd:minLength value="1"/>

1610 <xsd:maxLength value="6"/>

1611 </xsd:restriction>

1612 </xsd:simpleType>

1613 ...

1614 <xsd:element name="lineNums" type="lineNumList6"/>

1615 maps to:

1616 size_t lineNums_num;

1617 long lineNums[6];

- Since C does not allow for unbounded arrays, when elements with `maxOccurs = unbounded` and lists without a defined `length` or `maxLength`, map to a struct, the mapping is to a count of the number of occurrences and a pointer to the location where the actual data is located as an array

1621 Examples:

1622 <xsd:element name="counts" type="xsd:int" maxOccurs="unbounded"/>

1623 maps to:

1624 size_t counts_num;

1625 int *counts;

```

1626     /* this points to a dynamically allocated array of struct tm's */
1627     <xsd:simpleType name="lineNumList">
1628         <xsd:list itemType="xsd:int"/>
1629     </xsd:simpleType>
1630 ...
1631     <xsd:element name="lineNums" type="lineNumList"/>
1632 
```

maps to:

```

1633     size_t lineNums_num;
1634     long *lineNums;
1635     /* this points to a dynamically allocated array of longs */
1636 
```

- 1637 • Union Types are not supported.

1638 10.3.1.2 C to WSDL Mapping Details

- 1639 • wchar_t[] and char[] map to xsd:string with a maxLength facet.
- 1640 • C arrays map as normal elements but with multiplicity allowed via the minOccurs and maxOccurs facets.

1642 Example:

```

1643     long myFunction(char* name, int idList[], double value);
1644 
```

maps to:

```

1645     <xsd:element name="myFunction">
1646         <xsd:complexType>
1647             <xsd:sequence>
1648                 <xsd:element name="name" type="xsd:string"/>
1649                 <xsd:element name="idList" type="xsd:short"
1650                     minOccurs="0" maxOccurs="unbounded"/>
1651                 <xsd:element name="value" type="xsd:double"/>
1652             </xsd:sequence>
1653         </xsd:complexType>
1654     </xsd:element>
1655 
```

- 1656 • Multi-dimensional arrays map into nested elements.

1657 Example:

```

1658     long myFunction(int multiIdArray[][4][2]);
1659 
```

maps to:

```

1660     <xsd:element name="myFunction">
1661         <xsd:complexType>
1662             <xsd:sequence>
1663                 <xsd:element name="multiIdArray"
1664                     minOccurs="0" maxOccurs="unbounded"/>
1665                 <xsd:complexType>
1666                     <xsd:sequence>
1667                         <xsd:element name="multiIdArray"
1668                             minOccurs="4" maxOccurs="4"/>
1669                         <xsd:complexType>
1670                             <xsd:sequence>
1671                                 <xsd:element name="multiIdArray" type="xsd:short"
1672                                     minOccurs="2" maxOccurs="2" />
1673                             </xsd:sequence>
1674                         </xsd:complexType>
1675                     </xsd:element>
1676                 </xsd:sequence>
1677             </xsd:complexType>
1678         </xsd:element>
1679     </xsd:sequence>
1680 
```

```
1677     </xsd:complexType>
1678     </xsd:element>
1679     </xsd:sequence>
1680     </xsd:complexType>
1681 </xsd:element>
1682
```

- Except as detailed in the table above, pointers do not affect the type mapping, only the classification as in, out, or in/out.

1685 **10.3.2 Complex Content Binding**

1686 When mapping between XSD complex content types and C, either instances of SDO DataObjects or
1687 structs are used. An SCA implementation MUST support mapping message parts or global elements with
1688 complex types and parameters, return types, and struct members with a type defined by a struct. The
1689 mapping from WSDL MAY be to DataObjects and/or structs. The mapping to and from structs MUST
1690 follow the rules defined in WSDL to C Mapping Details. [C100016]

1691 **10.3.2.1 WSDL to C Mapping Details**

- Complex types and groups mapped to static DataObjects follow the rules defined in [SDO21].
- Complex types and groups mapped to structs have the attributes and elements of the type mapped to members of the struct.
 - The name of the struct is the name of the type or group.
 - Attributes appear in the struct before elements.
 - Simple types are mapped to members as described above.
 - The same rules for variable number of instances of a simple type element apply to complex type elements.
 - A sequence group is mapped as either a simple type or a complex type as appropriate.

1701 Example:

```
1702 <xsd:complexType name="myType">
1703   <xsd:sequence>
1704     <xsd:element name="name">
1705       <xsd:simpleType>
1706         <xsd:restriction base="xsd:string">
1707           <xsd:length value="25"/>
1708         </xsd:restriction>
1709       </xsd:simpleType>
1710     </xsd:element>
1711     <xsd:element name="idList" type="xsd:int"
1712                   minOccurs="0" maxOccurs="unbounded"/>
1713     <xsd:element name="value" type="xsd:double"/>
1714   </xsd:sequence>
1715 </xsd:complexType>
```

1716 maps to:

```
1717 struct myType {
1718   wchar_t name[26];
1719   size_t idList_num;
1720   long *idList;
1721   /* this points to a dynamically allocated array of longs */
1722   double value;
1723 };
```

- 1725 • While XML Schema allow the elements of an `all` group to appear in any order, the order is fixed in
 1726 the C mapping. Each child of an `all` group is mapped as pointer to the value and value itself. If the
 1727 child is not present, the pointer is NULL and the value is undefined.

1728 Example:

```
1729 <xsd:element name="myVariable">
1730   <xsd:complexType name="myType">
1731     <xsd:all>
1732       <xsd:element name="name" type="xsd:string"/>
1733       <xsd:element name="idList" type="xsd:int"
1734         minOccurs="0" maxOccurs="unbounded"/>
1735       <xsd:element name="value" type="xsd:double"/>
1736     </xsd:all>
1737   </xsd:complexType>
1738 </xsd:element>
```

1739 maps to:

```
1740 struct myType {
1741   wchar_t *name;
1742   /* this points to a dynamically allocated string */
1743   size_t idList_num;
1744   long *idList;
1745   /* this points to a dynamically allocated array of longs */
1746   double *value;
1747   /* this points to a dynamically allocated long */
1748 } *pmyVariable, myVariable;
```

- 1750 • Handing of `choice` groups is not defined by this mapping, and is implementation dependent. For
 1751 portability, `choice` groups are discouraged in service interfaces.
- 1752 • Nillable elements are mapped to a pointer to the value and the value itself. If the element is not
 1753 present, the pointer is NULL and the value is undefined.

1754 Example:

```
1755 <xsd:element name="priority" type="xsd:short" nillable="true"/>
```

1756 maps to:

```
1757 int16_t *pprioiry, priority;
```

- 1759 • Mixed content and open content (Any Attribute and Any Element) is supported via DataObjects.

10.3.2.2 C to WSDL Mapping Details

- 1761 • C structs that contain types that can be mapped, are themselves mapped to complex types.

1762 Example:

```
1763 char *myFunction(struct DataStruct data, int id);
```

1764 with the `DataStruct` type defined as a struct holding mappable types:

```
1765 struct DataStruct {
1766   char *name;
1767   double value;
1768 };
```

1769 maps to:

```
1770 <xsd:element name="myFunction">
1771   <xsd:complexType>
1772     <xsd:sequence>
1773       <xsd:element name="data" type="DataStruct" />
1774       <xsd:element name="id" type="xsd:int"/>
```

```

1775      </xsd:sequence>
1776    </xsd:complexType>
1777  </xsd:element>
1778
1779  <xsd:complexType name="DataStruct">
1780    <xsd:sequence>
1781      <xsd:element name="name" type="xsd:string"/>
1782      <xsd:element name="value" type="xsd:double"/>
1783    </xsd:sequence>
1784  </xsd:complexType>
1785

```

- **char** and **wchar_t** arrays inside of structs are mapped to a restricted subtype **xsd:string** that limits the length the space allowed in the array.

Example:

```

1789 struct DataStruct {
1790   char name[256];
1791   double value;
1792 };
1793

```

maps to:

```

1794 <xsd:element name="myFunction">
1795   <xsd:complexType>
1796     <xsd:sequence>
1797       <xsd:element name="data" type="DataStruct" />
1798       <xsd:element name="id" type="xsd:int"/>
1799     </xsd:sequence>
1800   </xsd:complexType>
1801 </xsd:element>
1802
1803 <xsd:complexType name="DataStruct">
1804   <xsd:sequence>
1805     <xsd:element name="name">
1806       <xsd:simpleType>
1807         <xsd:restriction base="xsd:string">
1808           <xsd:maxLength value="255"/>
1809         </xsd:restriction>
1810       </xsd:simpleType>
1811     </xsd:element>
1812     <xsd:element name="value" type="xsd:double"/>
1813   </xsd:sequence>
1814 </xsd:complexType>
1815

```

- C enums define a list of named symbols that map to values. If a function uses an **enum** type, this is mapped to a restricted element in the WSDL schema.

Example:

```

1819 char *getValueFromType(enum ParameterType type);
1820

```

with the **ParameterType** type defined as an **enum**:

```

1821 enum ParameterType {
1822   UNSET = 1,
1823   TYPEA,
1824   TYPEB,
1825   TYPEC
1826 };
1827

```

maps to:

```

1828 <xsd:element name="getValueFromType">
1829   <xsd:complexType>
1830     <xsd:sequence>
1831       <xsd:element name="type" type="ParameterType"/>
1832     </xsd:sequence>
1833   </xsd:complexType>
1834 </xsd:element>
1835
1836 <xsd:simpleType name="ParameterType">
1837   <xsd:restriction base="xsd:int">
1838     <xs:minInclusive value="1"/>
1839     <xs:maxInclusive value="4"/>
1840   </xsd:restriction>
1841 </xsd:simpleType>
1842

```

1843 The restriction used will have to be appropriate to the values of the enum elements.

1844 Example:

```

1845 enum ParameterType {
1846   UNSET = 'u',
1847   TYPEA = 'A',
1848   TYPEB = 'B',
1849   TYPEC = 'C'
1850 };

```

1851 maps to:

```

1852 <xsd:simpleType name="ParameterType">
1853   <xsd:restriction base="xsd:int">
1854     <xsd:enumeration value="86"/> <!-- Character 'u' -->
1855     <xsd:enumeration value="65"/> <!-- Character 'A' -->
1856     <xsd:enumeration value="66"/> <!-- Character 'B' -->
1857     <xsd:enumeration value="67"/> <!-- Character 'C' -->
1858   </xsd:restriction>
1859 </xsd:simpleType>
1860

```

- If a struct or enum contains other structs or enums, the mapping rules are applied recursively.

1862 Example:

```
char *myFunction(struct DataStruct data);
```

1864 with types defined as follows:

```

1865 struct DataStruct {
1866   char name[30];
1867   double values[20];
1868   ParameterType type;
1869 };
1870
1871 enum ParameterType {
1872   UNSET = 1,
1873   TYPEA,
1874   TYPEB,
1875   TYPEC
1876 };

```

1877 maps to:

```

1878 <xsd:element name="myFunction">
1879   <xsd:complexType>
1880     <xsd:sequence>
```

```

1881     <xsd:element name="data" type="DataStruct"/>
1882     </xsd:sequence>
1883   </xsd:complexType>
1884 </xsd:element>
1885
1886 <xsd:complexType name="DataStruct">
1887   <xsd:sequence>
1888     <xsd:element name="name">
1889       <xsd:simpleType>
1890         <xsd:restriction base="xsd:string">
1891           <xsd:maxLength value="29"/>
1892         </xsd:restriction>
1893       </xsd:simpleType>
1894     </xsd:element>
1895     <xsd:element name="values" type="xsd:double" minOccurs=20
1896 maxOccurs=20/>
1897       <xsd:element name="type" type=" ParameterType"/>
1898     </xsd:sequence>
1899 </xsd:complexType>
1900
1901 <xsd:simpleType name="ParameterType">
1902   <xsd:restriction base="xsd:int">
1903     <xs:minInclusive value="1"/>
1904     <xs:maxInclusive value="4"/>
1905   </xsd:restriction>
1906 </xsd:simpleType>
1907

```

- 1908 • Handing of C unions is not defined by this mapping, and is implementation dependent. For
1909 unions are discouraged in service interfaces.
- 1910 • Typedefs are resolved when evaluating parameter and return types. Typedefs are resolved before
1911 the mapping to Schema is done.
- 1912 • Handling for pre-processor directives is not defined by this mapping, and support is implementation
1913 dependent. For portability pre-processor directives are discouraged in service interfaces.

1914 11 Conformance

- 1915 An SCA implementation MUST reject a composite file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> or <http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd>. [C110001]
- 1916
- 1917
- 1918 An SCA implementation MUST reject a componentType or constraining type file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd>. [C110002]
- 1919
- 1920 An SCA implementation MUST reject a contribution file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200903/sca-contribution-c-1.1.xsd>. [C110003]
- 1921
- 1922 An SCA implementation MUST reject a WSDL file that does not conform to <http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200901/sca-wsdlext-c-1.1.xsd>. [C110004]
- 1923

1924 11.1 Conformance Targets

1925 The conformance targets of this specification are:

- 1926 • **SCA implementations**, which provide a **runtime** for SCA components and potentially **tools** for authoring SCA artifacts, component descriptions and/or runtime operations.
- 1927
- 1928 • **SCA documents**, which describe SCA artifacts, and specific **elements** within these documents.
- 1929 • **C files**, which define SCA service interfaces and implementations.
- 1930 • **WSDL files**, which define SCA service interfaces.

1931 11.2 SCA Implementations

1932 An implementation conforms to this specification if it meets the following conditions:

- 1933 1. It MUST conform to the SCA Assembly Model Specification [**ASSEMBLY**] and the SCA Policy Framework [**POLICY**].
- 1934
- 1935 2. It MUST comply with all statements in Conformance Items related to an SCA implementation.
- 1936 3. It MUST implement the SCA C API defined in section C API.
- 1937 4. It MUST implement the mapping between C and WSDL 1.1 [**WSDL11**] defined in WSDL to C and C to WSDL Mapping.
- 1938
- 1939 5. It MUST support <interface.c/> and <implementation.c/> elements as defined in Component Type and Component in composite, componentType and constrainingType documents.
- 1940
- 1941 6. It MUST support <export.c/> and <import.c/> elements as defined in C Contributions in contribution documents.
- 1942
- 1943 7. It MAY support source file annotations as defined in C SCA Annotations, C SCA Policy Annotations and C WSDL Annotations.
- 1944
- 1945 8. It MAY support WSDL extenstions as defined in C WSDL Mapping Extensions.

1946 11.3 SCA Documents

1947 An SCA document conforms to this specification if it meets the following conditions:

- 1948 9. It MUST conform to the SCA Assembly Model Specification [**ASSEMBLY**] and, if appropriate, the SCA Policy Framework [**POLICY**].
- 1949
- 1950 10. If it is a composite document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> and <http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd> schema and MUST comply with the additional constraints on the document contents as defined in Conformance Items..
- 1951
- 1952
- 1953

1955 If it is a componentType or constrainingType document, it MUST conform to the
1956 <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> schema and
1957 MUST comply with the additional constraints on the document contents as defined in
1958 Conformance Items.

1959 If it is a contribution document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200903/sca-contribution-c-1.1.xsd> schema and MUST comply
1960 with the additional constraints on the document contents as defined in Conformance
1961 Items.
1962

1963 **11.4C Files**

1964 A C file conforms to this specification if it meets the following conditions:

- 1965 1. It MUST comply with all statements in Conformance Items related to C contents and annotations.

1966 **11.5 WSDL Files**

1967 A WSDL conforms to this specification if it meets the following conditions:

- 1968 1. It is a valid WSDL 1.1 [[WSDL11](#)] document.
- 1969 2. It MUST comply with all statements in Conformance Items related to WSDL contents and extensions.

1970 A C SCA Annotations

1971 To allow developers to define SCA related information directly in source files, without having to separately
1972 author SCDL files, a set of annotations is defined. If SCA annotations are supported by an
1973 implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as
1974 described. The SCA runtime MUST only process the SCDL files and not the annotations. [CA0001]

1975 A.1 Application of Annotations to C Program Elements

1976 In general an annotation immediately precedes the program element it applies to. If multiple annotations
1977 apply to a program element, all of the annotations SHOULD be in the same comment block. [CA0002]

- 1978 • Function or Function Prototype

1979 The annotation immediately precedes the function definition or declaration.

1980 Example:

```
1981 /* @OneWay */  
1982 reportEvent(int eventID);
```

- 1983 • Variable

1984 The annotation immediately precedes the variable definition.

1985 Example:

```
1986 /* @Property */  
1987 long loanType;
```

- 1988 • Set of Functions Implementing a Service

1989 A set of functions implementing a service begins with an @Service annotation. Any
1990 annotations applying to this service as a whole immediately precede the @Service
1991 annotation. These annotations SHOULD be in the same comment block as the @Service
1992 annotation.

1993 Example:

```
1994 /* @Scope("composite")  
1995 * @Service(name="LoanService", interfaceHeader="loan.h") */
```

- 1996 • Set of Function Prototypes Defining an Interface

1997 To avoid any ambiguity about the application of an annotation to a specific function or
1998 the set of functions defining an interface, if an annotation is to apply to the interface as
1999 a whole, then the @Interface annotation is used, even in the case where there is just
2000 one interface defined in a header file. Any annotations applying to the interface
2001 immediately precede the @Interface annotation.

```
2002 /* @Remoteable  
2003 * @Interface(name="LoanService" */
```

2004 A.2 Interface Header Annotations

2005 This section lists the annotations that can be used in the header file that defines a service interface.

2006 A.2.1 @Interface

2007 Annotation that indicates the start of a new interface definition. An SCA implementation MUST treat a file
2008 with a @WebService annotation specified as if @Interface was specified with the name value of the
2009 @WebService annotation used as the name value of the @Interface annotation. [CA0003]

2010

2011 **Corresponds to:** *interface.c* element

2012 **Format:**

2013 */* @Interface(name="serviceName") */*

2014 where

- 2015 • **name : NCName (1..1)** – specifies the name of the service.

2016 **Applies to:** Set of functions defining an interface.

2017 Function declarations following this annotation form the definition of this interface. This annotation also
2018 serves to bound the scope of the remaining annotations in this section,

2019 Example:

2020 Interface header:

2021 */* @Interface(name="LoanService") */*

2022 Service definition:

2023 *<service name="LoanService">*
2024 *<interface.c header="loans.h" />*
2025 *</service>*

2026 **A.2.2 @Operation**

2027 Annotation that indicates that a function defines an operation of a service. There are two formats for this
2028 annotation depending on if the service is implemented as a set of subroutines or in a program. An SCA
2029 implementation MUST treat a function with a @WebFunction annotation specified, unless the exclude
2030 value of the @WebFunction annotation is true, as if @Operation was specified with the operationName
2031 value of the @WebFunction annotation used as the name value of the @Operation annotation. [CA0004]
2032 An SCA implementation MUST treat a struct with a @WebOperation annotation specified, unless the
2033 exclude value of the @WebOperation annotation is true, as if @Operation was specified with the struct as
2034 the input value, the operationName value of the @WebOperation annotation used as the name value of
2035 the @Operation annotation and the response value of the @WebOperation annotation used as the output
2036 values of the @Operation annotation. [CA0005]

2037 **Corresponds to:** *function* child element of an *interface.c* element

2038 If the service is implemented as a set of subroutines, this format is used.

2039 **Format:**

2040 */* @Operation(name="operationName") */*

2041 where

- 2042 • **name : NCName (0..1)** – gives the operation a different name than the function name.

2043 **Applies to (library based implementations):** Function declaration

2044 The function declaration following this annotation defines an operation of the current service. If no
2045 @Operation annotation exists in an interface definition, all the function declarations in a header file or
2046 following an @Interface annotation define the operations of a service, otherwise only the annotated
2047 function declarations define operations for the service.

2048 Example:

2049 Interface header (loans.h):

2050 *short internalFcn(char *param1, short param2);*
2051
2052 */* @Operation(name="getRate") */*
2053 *void rateFcn(char *cust, float *rate);*

2054 Interface definition:

2055 *<interface.c header="loans.h">*
2056 *<operation name="getRate" />*
2057 *</interface.c>*

2058

2059 If the service is implemented in a program, the following format is used. In this format, all operations are
2060 be defined via annotations.

2061 **Format:**

```
2062 /* @Operation(name="operationName", input="inputStruct", output="outputStruct")  
2063 */
```

2064 where

- **name : NCName (1..1)** – specifies the name of the operation.
- **input : NCName (1..1)** – specifies the name of a struct that defines the format of the input message.
- **output : NCName (0..1)** – specifies the name of a struct that defined the format of the output message if one is used.

2069 **Applies to (program based implementations):** stuct declarations

2070 Example:

2071 Interface header (loans.h):

```
2072 /* @Operation(name="getRate", input="rateInput", output="rateOutput") */  
2073 struct rateInput {  
2074     char cust[25];  
2075     int term;  
2076 };  
2077 struct rateOutput {  
2078     float rate;  
2079     int rateClass;  
2080 };
```

2081 Interface definition:

```
2082 <interface.c header="loans.h">  
2083     <operation name="getRate" input="rateInput" output="rateOutput"/>  
2084 </interface.c>
```

2085 A.2.3 @Remotable

2086 Annotation on service interface to indicate that a service is remotable.

2087 **Corresponds to:** `@remotable="true"` attribute of an *interface.c* element.

2088 **Format:**

```
2089 /* @Remotable */
```

2090 The default is **false** (not remotable).

2091 **Applies to:** Interface

2092 Example:

2093 Interface header (LoanService.h):

```
2094 /* @Remotable */
```

2095 Service definition:

```
2096 <service name="LoanService">  
2097     <interface.c header="LoanService.h" remotable="true" />  
2098 </service>
```

2099 A.2.4 @Callback

2100 Annotation on a service interface to specify the callback interface.

2101 **Corresponds to:** `@callbackHeader` attribute of an *interface.c* element.

2102 **Format:**

```
2103 /* @Callback(header="headerName") */
```

2104 where

- 2105 • **header : Name (1..1)** – specifies the name of the header defining the callback service interface.

2106 **Applies to:** Interface

2107 Example:

2108 Interface header (MyService.h):

```
2109     /* @Callback(header="MyServiceCallback.h") */
```

2110 Service definition:

```
2111     <service name="MyService">
2112         <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h" />
2113     </service>
```

2114 **A.2.5 @OneWay**

2115 Annotation on a service interface function declaration to indicate the function is one way. The @OneWay
2116 annotation also affects the representation of a service in WSDL. See @OneWay.

2117 **Corresponds to:** @oneWay="true" attribute of function element of an *interface.c* element.

2118 **Format:**

```
2119     /* @OneWay */
```

2120 The default is **false** (not OneWay).

2121 **Applies to:** Function Prototype

2122 Example:

2123 Interface header:

```
2124     /* @OneWay */
2125     reportEvent(int eventID);
```

2126 Service definition:

```
2127     <service name="LoanService">
2128         <interface.c header="LoanService.h">
2129             <function name="reportEvent" oneWay="true" />
2130         </interface.c>
2131     </service>
```

2132 **A.3 Implementation Annotations**

2133 This section lists the annotations that can be used in the file that implements a service.

2134 **A.3.1 @ComponentType**

2135 Annotation used to indicate the start of a new componentType.

2136 **Corresponds to:** @componentType attribute of an *implementation.c* element.

2137 **Format:**

```
2138     /* @ComponentType */
```

2139 **Applies to:** Set of services, references and properties

2140 Example:

2141 Implementation:

```
2142     /* @ComponentType */
```

2143 Component definition:

```
2144     <component name="LoanService">
2145         <implementation.c module="loan" componentType="LoanService" />
2146     </component>
```

2147 A.3.2 @Service

2148 Annotation that indicates the start of a new service implementation.

2149 **Corresponds to:** *implementation.c* element

2150 **Format:**

```
2151 /* @Service(name="serviceName", interfaceHeader="headerFile") */
```

2152 where

- **name : NCName (1..1)** – specifies the name of the service.

- **interfaceHeader : Name (1..1)** – specifies the C header defining the interface.

2155 **Applies to:** Set of functions implementing a service

2156 Function definitions following this annotation form the implementation of this service. This annotation also
2157 serves to bound the scope of the remaining annotations in this section,

2158 Example:

2159 Implementation:

```
2160 /* @Service(name="LoanService", interfaceHeader="loan.h") */
```

2161 ComponentType definition:

```
2162 <componentType name="LoanService">
2163   <service name="LoanService">
2164     <interface.c header="loans.h" />
2165   </service>
2166 </componentType>
```

2167 A.3.3 @Reference

2168 Annotation on a service implementation to indicate it depends on another service providing a specified
2169 interface.

2170 **Corresponds to:** *reference* element of a *componentType* element.

2171 **Format:**

```
2172 /* @Reference(name="referenceName", interfaceHeader="headerFile",
2173               required="true", multiple="true")
2174 */
```

2175 where

- **name : NCName (1..1)** – specifies the name of the reference.

- **interfaceHeader : Name (1..1)** – specifies the C header defining the interface.

- **required : boolean (0..1)** – specifies whether a value has to be set for this reference. Default is **true**.

- **multiple : boolean (0..1)** – specifies whether this reference can be wired to multiple services. Default is **false**.

2181 The multiplicity of the reference is determined from the **required** and **multiple** attributes. If the value of
2182 the **multiple** attribute is true, then component type has a reference with a multiplicity of either 0..n or 1..n
2183 depending on the value of the **required** attribute – 1..n applies if **required=true**. Otherwise a multiplicity
2184 of 0..1 or 1..1 is implied.

2185 **Applies to:** Service

2186 Example:

2187 Implementation:

```
2188 /* @Reference(name="getRate", interfaceHeader="rates.h") */
2189 /* @Reference(name="publishRate", interfaceHeader="myRates.h",
2190               required="false", multiple="yes")
2191 */
```

2193

2194 **ComponentType definition:**

```
2195     <componentType name="LoanService">
2196         <reference name="getRate">
2197             <interface.c header="rates.h">
2198         </reference>
2199         <reference name="publishRate" multiplicity="0..n">
2200             <interface.c header="myRates.h">
2201         </reference>
2202     </componentType>
```

2203 **A.3.4 @Property**

2204 Annotation on a service implementation to define a property of the service. Should immediately precedes
2205 the global variable that the property is based on. The variable declaration is only used for determining the
2206 type of the property. The variable will not be populated with the property value at runtime. Programs use
2207 the SCAProperty<Type>() functions for accessing property data.

2208 **Corresponds to:** *property* element of a *componentType* element.

2209 **Format:**

```
2210     /* @Property(name="propertyName", type="typeName",
2211                    default="defaultValue", required="true")
2212     */
```

2213 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 global variable.
- **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 global variable to an xsd type as defined in Data Binding. If the variable 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**.

2222 **Applies to:** Variable

2223 Example:

2224 Implementation:

```
2225     /* @Property */
2226     long loanType;
```

2227 ComponentType definition:

```
2228     <componentType name="LoanService">
2229         <property name="loanType" type="xsd:int" />
2230     </componentType>
```

2231 **A.3.5 @Scope**

2232 Annotation on a service implementation to indicate the scope of the service.

2233 **Corresponds to:** *@scope* attribute of an *implementation.c* element.

2234 **Format:**

```
2235     /* @Scope("value") */
```

2236 where

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

2239 **Applies to:** Service

2240 Example:

2241 Implementation:

```
2242     /* @Scope("composite") */
```

2243 Component definition:

```
2244     <component name="LoanService">
2245         <implementation.c module="loan" componentType="LoanService"
2246             scope="composite" />
2247     </component>
```

2248 **A.3.6 @Init**

2249 Annotation on a service implementation to indicate a function to be called when the service is
2250 instantiated. If the service is implemented in a program, this annotation indicates the program is to be
2251 called with an initialization flag prior to the first operation.

2252 **Corresponds to:** `@init="true"` attribute of an *implementation.c* element or a *function* child element of an
2253 *implementation.c* element.

2254 **Format:**

```
2255     /* @Init */
```

2256 The default is **false** (the function is not to be called on service initialization).

2257 **Applies to:** Function or Service

2258 Example:

2259 Implementation:

```
2260     /* @Init */
2261     void init();
```

2262

2263 Component definition:

```
2264     <component name="LoanService">
2265         <implementation.c module="loan" componentType="LoanService">
2266             <function name="init" init="true" />
2267         </implementation.c>
2268     </component>
```

2269 **A.3.7 @Destroy**

2270 Annotation on a service implementation to indicate a function to be called when the service is terminated.
2271 If the service is implemented in a program, this annotation indicates the program is to be called with a
2272 termination flag after the final operation.

2273 **Corresponds to:** `@destroy="true"` attribute of an *implementation.c* element or a *function* child element of
2274 an *implementation.c* element.

2275 **Format:**

```
2276     /* @Destroy */
```

2277 The default is **false** (the function is not to be called on service termination).

2278 **Applies to:** Function or Service

2279 Example:

2280 Implementation:

```
2281     /* @Destroy */
2282     void cleanup();
```

2283 Component definition:

```
2284     <component name="LoanService">
2285         <implementation.c module="loan" componentType="LoanService">
2286             <function name="cleanup" destroy="true" />
```

```
2287     </implementation.c>
2288   </component>
```

2289 A.3.8 @EagerInit

2290 Annotation on a service implementation to indicate the service is to be instantiated when its containing
2291 component is started.

2292 **Corresponds to:** `@eagerInit="true"` attribute of an *implementation.c* element.

2293 **Format:**

```
2294     /* @EagerInit */
```

2295 The default is **false** (the service is initialized lazily).

2296 **Applies to:** Service

2297 Example:

2298 Implementation:

```
2299     /* @EagerInit */
```

2300 Component definition:

```
2301     <component name="LoanService">
2302       <implementation.c module="loan" componentType="LoanService"
2303         eagerInit="true" />
2304     </component>
```

2305 A.3.9 @AllowsPassByReference

2306 Annotation on service implementation or operation to indicate that a service or operation allows pass by
2307 reference semantics.

2308 **Corresponds to:** `@allowsPassByReference="true"` attribute of an *implementation.c* element or a *function*
2309 child element of an *implementation.c* element.

2310 **Format:**

```
2311     /* @AllowsPassByReference */
```

2312 The default is **false** (the service does not allow by reference parameters).

2313 **Applies to:** Service or Function

2314 Example:

2315 Implementation:

```
2316     /* @Service(name="LoanService")
2317      * @AllowsPassByReference
2318      */
```

2319 Component definition:

```
2320     <component name="LoanService">
2321       <implementation.c module="loan" componentType="LoanService"
2322         allowsPassByReference="true" />
2323     </component>
```

2324 A.4 Base Annotation Grammar

2325 While annotations are defined using the `/* ... */` format for comments, if the `// ...` format is supported by a
2326 C compiler, the `// ...` format MAY be supported by an annotation processor.

2327

```
2328   <annotation> ::= /* @<baseAnnotation> */
2329
2330   <baseAnnotation> ::= <name> [ (<params>) ]
2331
2332   <params> ::= <paramNameValue>[, <paramNameValue>]* |
```

```
2333             <paramValue>[, <paramValue>] *
2334
2335             <paramNameValue> ::= <name>=<value>
2336
2337             <paramValue> ::= "<value>"
2338
2339             <name> ::= NCName
2340
2341             <value> ::= string
```

- 2342 • Adjacent string constants are concatenated
2343 • NCName is as defined by XML schema **[XSD]**
2344 • Whitespace including newlines between tokens is ignored.
2345 • Annotations with parameters can span multiple lines within a comment, and are considered complete
2346 when the terminating ")" is reached.

2347

B C SCA Policy Annotations

2348 SCA provides facilities for the attachment of policy-related metadata to SCA assemblies, which influence
 2349 how implementations, services and references behave at runtime. The policy facilities are described in
 2350 [POLICY]. In particular, the facilities include Intents and Policy Sets, where intents express abstract,
 2351 high-level policy requirements and policy sets express low-level detailed concrete policies.

2352 Policy metadata can be added to SCA assemblies through the means of declarative statements placed
 2353 into Composite documents and into Component Type documents. These annotations are completely
 2354 independent of implementation code, allowing policy to be applied during the assembly and deployment
 2355 phases of application development.

2356 However, it can be useful and more natural to attach policy metadata directly to the code of
 2357 implementations. This is particularly important where the policies concerned are relied on by the code
 2358 itself. An example of this from the Security domain is where the implementation code expects to run
 2359 under a specific security Role and where any service operations invoked on the implementation have to
 2360 be authorized to ensure that the client has the correct rights to use the operations concerned. By
 2361 annotating the code with appropriate policy metadata, the developer can rest assured that this metadata
 2362 is not lost or forgotten during the assembly and deployment phases.

2363 The SCA C policy annotations provide the capability for the developer to attach policy information to C
 2364 implementation code. The annotations concerned first provide general facilities for attaching SCA Intents
 2365 and Policy Sets to C code. Secondly, there are further specific annotations that deal with particular policy
 2366 intents for certain policy domains such as Security.

B.1 General Intent Annotations

2368 SCA provides the annotation **@Requires** for the attachment of any intent to a C function, to a C function
 2369 declaration or to sets of functions implementing a service or sets of function declarations defining a
 2370 service interface.

2371 The @Requires annotation can attach one or multiple intents in a single statement.

2372 Each intent is expressed as a string. Intents are XML QNames, which consist of a Namespace URI
 2373 followed by the name of the Intent. The precise form used is as follows:

2374

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

2376

2377 Intents can be qualified, in which case the string consists of the base intent name, followed by a ".",
 2378 followed by the name of the qualifier. There can also be multiple levels of qualification.

2379 This representation is quite verbose, so we expect that reusable constants will be defined for the
 2380 namespace part of this string, as well as for each intent that is used by C code. SCA defines constants
 2381 for intents such as the following:

2382

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

2387

2388 Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,
 2389 separated by an underscore. These intent constants are defined in the file that defines an annotation for
 2390 the intent (annotations for intents, and the formal definition of these constants, are covered in a following
 2391 section).

2392 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.

2393

2394 **Corresponds to:** @requires attribute of a *service*, *reference*, *operation* or *property* element.

2395 **Format:**

```
2396       /* @Requires("qualifiedIntent" | {"qualifiedIntent" [, "qualifiedIntent"]}) */
```

2397 where

```
2398       qualifiedIntent ::= QName | QName.qualifier | QName.qualifier1.qualifier2
```

2399 **Applies to:** Interface, Service, Function, Function Prototype

2400 **Examples:**

2401 Attaching the intents "confidentiality.message" and "integrity.message".

```
2402       /* @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE}) */
```

2403

2404 A reference requiring support for confidentiality:

```
2405       /* @Requires(CONFIDENTIALITY)
2406        * @Reference(interfaceHeader="SetBar.h") */
2407       void setBar(struct barType *bar);
```

2408

2409 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:

```
2412
2413       /* @Requires(SCA_PREFIX "confidentiality")
2414        * @Reference(interfaceHeader="SetBar.h") */
2415       void setBar(struct barType *bar);
```

2416 B.2 Specific Intent Annotations

2417 In addition to the general intent annotation supplied by the @Requires annotation described above, there
2418 are C annotations that correspond to some specific policy intents.

2419 The general form of these specific intent annotations is an annotation with a name derived from the name
2420 of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation
2421 in the form of a string or an array of strings.

2422 For example, the SCA confidentiality intent described in General Intent Annotations using the
2423 @Requires(CONFIDENTIALITY) intent can also be specified with the specific @Confidentiality intent
2424 annotation. The specific intent annotation for the "integrity" security intent is:

```
2425
2426       /* @Integrity */
```

2427

2428 **Corresponds to:** @requires=<Intent>" attribute of a *service*, *reference*, *operation* or *property* element.

2429

2430 **Format:**

```
2431       /* @<Intent>[ (qualifiers) ] */
```

2432 where Intent is an NCName that denotes a particular type of intent.

```
2433       Intent ::= NCName
2434       qualifiers ::= "qualifier" | {"qualifier" [, "qualifier"] }
2435       qualifier ::= NCName | NCName/qualifier
```

2436

2437 **Applies to:** Interface, Service, Function, Function Prototype – but see specific intents for restrictions

2438

2439 **Example:**

2440 /* @Authentication({"message", "transport"}) */
 2441 This annotation attaches the pair of qualified intents: *authentication.message* and *authentication.transport*
 2442 (the *sca*: namespace is assumed in both of these cases – "http://docs.oasis-
 2443 open.org/ns/opencsa/sca/200903").
 2444
 2445 The Policy Framework **[POLICY]** defines a number of intents and qualifiers. The following sections
 2446 define the annotations for those intents.

2447 B.2.1 Security Interaction

Intent	Annotation
authentication	@Authentication
confidentiality	@Confidentiality
integrity	@Integrity

2448
 2449 These three intents can be qualified with
 2450 • transport
 2451 • message

2452 B.2.2 Security Implementation

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

2453
 2454 In addition to allow roles to defined, an SCA runtime MAY use the following annotation
 2455 @DeclareRoles(<comma separated list of roles>")

2456 B.2.3 Reliable Messaging

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

2457
 2458 B.2.4 Transactions

Intent	Annotation	Qualifiers

managedTransaction	@ManagedTransaction	Local global
transactedOneWay	@TransactedOneWay	
immediateOneWay	@ImmediateOneWay	
propagates Transaction	@PropagatesTransaction	
suspendsTransaction	@SuspendsTransaction	

2459

2460 **B.2.5 Miscellaneous**

Intent	Annotation	Qualifiers
SOAP	@SOAP	1_1 1_2
JMS	@JMS	

2461 **B.3 Application of Intent Annotations**

2462 Where multiple intent annotations (general or specific) are applied to the same C element, they are
 2463 additive in effect. An example of multiple policy annotations being used together follows:

2464

```
2465 /* @Authentication
2466 * @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE}) */
```

2467

2468 In this case, the effective intents are *authentication*, *confidentiality.message* and *integrity.message*.
 2469 If an annotation is specified at both the implementation/interface level and the function or variable level,
 2470 then the function or variable level annotation completely overrides the implementation/interface level
 2471 annotation of the same type.

2472 The intent annotation can be applied either to interface or to functions when adding annotated policy on
 2473 SCA services.

2474 **B.4 Policy Annotation Scope**

2475 The following examples show scope of intents on functions, function declarations and sets of each.

2476

```
2477 /* @Remotable
2478 * @Integrity("transport")
2479 * @Authentication
2480 * @Service(name="HelloService", interfaceHeader="helloservice.h") */
2481
2482 /* @Integrity
2483 * @Authentication("message")*/
2484 wchar_t* hello(wchar_t* message) {...}
2485
2486 /* @Integrity
2487 * @Authentication("transport") */
2488 wchar_t* helloThere() {...}
2489
2490 /* @Remotable
2491 * @Confidentiality("message")*/
```

```

2492     * @Service(name="HelloHelperService", interfaceHeader="helloService.h") */
2493
2494     /* @Confidentiality("transport") */
2495     wchar_t* hello(wchar_t* message) {...}
2496
2497     /* @Authentication */
2498     wchar_t* helloWorld() {...}

```

2499 Example 1a. Usage example of annotated policy and inheritance.

2500

- The effective intent annotation on the helloWorld function is @Authentication, and @Confidentiality("message").
- The effective intent annotation on the hello function of the HelloHelperService is @Confidentiality("transport"),
- The effective intent annotation on the helloThere function of the HelloService is @Integrity and @Authentication("transport").
- The effective intent annotation on the hello function of the HelloService is @Integrity and @Authentication("message")

2509 The listing below contains the equivalent declarative security interaction policy of the HelloService and
2510 HelloHelperService implementation corresponding to the Java interfaces and classes shown in Example
2511 1a.

2512

```

2513 <?xml version="1.0" encoding="ASCII"?>
2514
2515 <composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
2516           name="HelloServiceComposite" >
2517 ...
2518
2519 <component name="HelloServiceComponent">
2520   <service name="HelloService" requires="integrity/transport
2521             authentication">
2522   ...
2523   </service>
2524   <implementation.c module="HelloService.dll"
2525             componentType="LoanService" header="HelloService.h">
2526     <function name="hello" requires="integrity
2527                   authentication/message"/>
2528     <function name="helloThere" requires="integrity
2529                   authentication/transport"/>
2530   </implementation.c>
2531 </component>
2532 <component name="HelloHelperServiceComponent">
2533   <service name="HelloHelperService" requires="integrity/transport
2534             authentication confidentiality/message">
2535 ...
2536   </service>
2537   <implementation.c module="HelloService.dll"
2538             componentType="LoanService" header="HelloService.h">
2539     <function name="hello" requires="confidentiality/transport"/>
2540     <function name="helloWorld" requires="authentication"/>
2541   </implementation.c>
2542 </component>
2543 ...
2544 ...
2545 ...
2546 </composite>

```

2547 Example 1b. Declaratives intents equivalent to annotated intents in Example 1a.

2548 B.5 Relationship of Declarative And Annotated Intents

2549 Annotated intents on a C functions or function declarations cannot be overridden by declarative intents
2550 either in a composite document which uses the functions as an implementation or by statements in a
2551 componentType document associated with the functions. This rule follows the general rule for intents that
2552 they represent fundamental requirements of an implementation.

2553 An unqualified version of an intent expressed through an annotation in the C function or function
2554 declaration can be qualified by a declarative intent in a using composite document.

2555 B.6 Policy Set Annotations

2556 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies (for example,
2557 a concrete policy is the specific encryption algorithm to use when encrypting messages when using a
2558 specific communication protocol to link a reference to a service).

2559
2560 Policy Sets can be applied directly to C implementations using the **@PolicySets** annotation. The
2561 PolicySets annotation either takes the QName of a single policy set as a string or the name of two or
2562 more policy sets as an array of strings.

2563 **Corresponds to:** @policySets attribute of a *service*, *reference*, *operation* or *property* element.

2564

2565 Format:

```
2566 /* @PolicySets( "<policy set QName>" |  
2567 * { "<policy set QName>" [, "<policy set QName>" ] }) */
```

2568 As for intents, PolicySet names are QNames – in the form of “{Namespace-URI}localPart”.

2569

2570 **Applies to:** Interface, Service, Function, Function Prototype

2571

2572 Example:

```
2573 /* @Reference(name="helloService", interfaceHeader="helloService.h",  
2574 * required=true)  
2575 * @PolicySets({ MY_NS "WS_Encryption_Policy",  
2576 * MY_NS "WS_Authentication_Policy" }) */  
2577 HelloService* helloService;  
2578 ...  
2579 }
```

2580

2581 In this case, the Policy Sets WS_Encryption_Policy and WS_Authentication_Policy are applied, both
2582 using the namespace defined for the constant MY_NS.

2583 PolicySets satisfy intents expressed for the implementation when both are present, according to the rules
2584 defined in [POLICY].

2585 B.7 Policy Annotation Grammar Additions

```
2586 <annotation> ::= /* @<baseAnnotation> | @<requiresAnnotation> |  
2587 * @<intentAnnotation> | @<policySetAnnotation> */  
2588  
2589 <requiresAnnotation> ::= Requires(<intents>)  
2590  
2591 <intents> ::= "<qualifiedIntent>" |  
2592 * { "<qualifiedIntent>" [, "<qualifiedIntent>" ] * }  
2593  
2594 <qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |  
2595 * <intentName>.<qualifier>.qualifier>  
2596  
2597 <intentName> ::= {anyURI}NCName
```

```
2598 <intentAnnotation> ::= <intent>[(<qualifiers>)]  
2599  
2600 <intent> ::= NCName[ (param) ]  
2601  
2602 <qualifiers> ::= "<qualifier>" | {"<qualifier>" [, "<qualifier>"]*}  
2603  
2604 <qualifier> ::= NCName | NCName/<qualifier>  
2605  
2606 <policySetAnnotation> ::= policySets(<policysets>)  
2607  
2608 <policySets> ::= "<policySetName>" | {"<policySetName>" [, "<policySetName>"]*}  
2609  
2610 <policySetName> ::= {anyURI}NCName  
2611
```

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

2613 B.8 Annotation Constants

```
2614 <annotationConstant> ::= /* @Define <identifier> <token string> */  
2615  
2616 <identifier> ::= token  
2617  
2618 <token string> ::= "string" | "string"[ ## <token string>]  
2619
```

- Constants are immediately expanded

2620 C C WSDL Annotations

2621 To allow developers to control the mapping of C to WSDL, a set of annotations is defined. If WSDL
2622 mapping annotations are supported by an implementation, the annotations defined here MUST be
2623 supported and MUST be mapped to WSDL as described. [CC0005]

2624 C.1 Interface Header Annotations

2625 C.1.1 @WebService

2626 Annotation on a C header file indicating that it represents a web service. A second or subsequent
2627 instance of this annotation in a file, or a first instance after any function declarations indicates the start of
2628 a new service and has to contain a name value. An SCA implementation MUST treat any instance of a
2629 @Interface annotation and without an explicit @WebService annotation as if a @WebService annotation
2630 with a name value equal to the name value of the @Interface annotation and no other parameters was
2631 specified. [CC0001]

2632 **Corresponds to:** javax.jws.WebService annotation in the JAX-WS specification (7.11.1)

2633 **Format:**

```
2634 /* @WebService(name="portType", targetNamespace="namespaceURI",  
2635 *           serviceName="WSDLServiceName", portName="WSDLPortName") */
```

2636 where

- **name : NCName (0..1)** – specifies the name of the web service portType. The default is the root
2638 name of the header file containing the annotation.
- **targetNamespace : anyURI (0..1)** – specifies the target namespace for the web service. The default
2640 namespace is determined by the implementation.
- **serviceName : NCName (0..1)** – specifies the name for the associated WSDL service. The default
2642 service name is the name of the header file containing the annotation suffixed with “Service”. The
2643 name of the associated binding is also determined by the serviceName. In the case of a SOAP
2644 binding, the binding name is the name of the service suffixed with “SoapBinding”.
- **portName : NCName (0..1)** – specifies the name for the associated WSDL port for the service. If a
2646 @WebService does not have a **portName** element, an SCA implementation MUST use the value
2647 associated with the **name** element, suffixed with “Port”. [CC0008]

2648 **Applies to:** Header file

2649 Example:

2650 Input C header file (stockQuote.h):

```
2651 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
2652 *           serviceName="StockQuoteService") */  
2653 ...
```

2655

2656 Generated WSDL file:

```
2657 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
2658   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
2659   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
2660   xmlns:tns="http://www.example.org/"  
2661   targetNamespace="http://www.example.org/">  
2662  
2663   <portType name="StockQuote">  
2664     <sca-c:bindings>  
2665       <sca-c:prefix name="stockQuote"/>  
2666     </sca-c:bindings>
```

```

2667 </portType>
2668
2669 <binding name="StockQuoteServiceSoapBinding">
2670   <soap:binding style="document"
2671     transport="http://schemas.xmlsoap.org/soap/http"/>
2672 </binding>
2673
2674 <service name="StockQuoteService">
2675   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2676     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2677   </port>
2678 </service>
2679 </definitions>
```

2680 C.1.2 @WebFunction

2681 Annotation on a C function indicating that it represents a web service operation. An SCA implementation
 2682 MUST treat a function annotated with an @Operation annotation and without an explicit @WebFunction
 2683 annotation as if a @WebFunction annotation with an operationName value equal to the name value
 2684 of the @Operation annotation and no other parameters was specified. [CC0002]

2685 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2686 **Format:**

```

2687 /* @WebFunction(operationName="operation",    action="SOAPAction",
2688 *                  exclude="false") */
```

2689 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 function the annotation is applied to omitting any preceding namespace prefix and portType name.
- **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 function is included in the web service interface. The default value is “false”.

2697 **Applies to:** Function.

2698

2699 Example:

2700 Input C header file:

```

2701 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2702 *                  serviceName="StockQuoteService") */
2703
2704 /* @WebFunction(operationName="GetLastTradePrice",
2705 *                  action="urn:GetLastTradePrice") */
2706 float getLastTradePrice(const char *tickerSymbol);
2707
2708 /* @WebFunction(exclude="true") */
2709 void setLastTradePrice(const char *tickerSymbol, float value);
```

2710

2711 Generated WSDL file:

```

2712 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2713   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2714   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
2715   xmlns:tns="http://www.example.org/"
2716   targetNamespace="http://www.example.org/">
2717
2718   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2719     xmlns:tns="http://www.example.org/"
2720     attributeFormDefault="unqualified"
```

```

2721     elementFormDefault="unqualified"
2722     targetNamespace="http://www.example.org/">
2723 <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2724 <xs:element name="GetLastTradePriceResponse"
2725     type="tns:GetLastTradePriceResponse"/>
2726 <xs:complexType name="GetLastTradePrice">
2727     <xs:sequence>
2728         <xs:element name="tickerSymbol" type="xs:string"/>
2729     </xs:sequence>
2730 </xs:complexType>
2731 <xs:complexType name="GetLastTradePriceResponse">
2732     <xs:sequence>
2733         <xs:element name="return" type="xs:float"/>
2734     </xs:sequence>
2735 </xs:complexType>
2736 </xs:schema>
2737
2738 < message name="GetLastTradePrice">
2739     <part name="parameters" element="tns:GetLastTradePrice">
2740     </part>
2741 </message>
2742
2743 < message name="GetLastTradePriceResponse">
2744     <part name="parameters" element="tns:GetLastTradePriceResponse">
2745     </part>
2746 </ message>
2747
2748 <portType name="StockQuote">
2749     <sca-c:bindings>
2750         <sca-c:prefix name="stockQuote"/>
2751     </sca-c:bindings>
2752     <operation name="GetLastTradePrice">
2753         <sca-c:bindings>
2754             <sca-c:function name="getLastTradePrice"/>
2755         </sca-c:bindings>
2756         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2757             </input>
2758         <output name="GetLastTradePriceResponse"
2759             message="tns:GetLastTradePriceResponse">
2760             </output>
2761         </operation>
2762     </portType>
2763
2764 <binding name="StockQuoteServiceSoapBinding">
2765     <soap:binding style="document"
2766         transport="http://schemas.xmlsoap.org/soap/http"/>
2767     <wsdl:operation name="GetLastTradePrice">
2768         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2769         <wsdl:input name="GetLastTradePrice">
2770             <soap:body use="literal"/>
2771         </wsdl:input>
2772         <wsdl:output name="GetLastTradePriceResponse">
2773             <soap:body use="literal"/>
2774         </wsdl:output>
2775     </wsdl:operation>
2776 </binding>
2777
2778 <service name="StockQuoteService">
2779     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2780         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2781     </port>
2782 </service>
2783 </definitions>

```

2784 C.1.3 @WebOperation

2785 Annotation on a C request message struct indicating that it represents a web service operation. An SCA
2786 implementation MUST treat an @Operation annotation without an explicit @WebOperation annotation as
2787 if a @WebOperation annotation with with an operationName value equal to the name value of the
2788 @Operation annotation, a response value equal to the output value of the @Operation annotation and no
2789 other parameters was specified is applied to the struct identified as the input value of the @Operation
2790 annotation. [CC0003]

2791 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2792 **Format:**

```
2793 /* @WebOperation(operationName="operation", response="responseStruct",
2794 *                  action="SOAPAction", exclude="false") */
```

2795 where:

- **operationName : NCName (0..1)** – specifies the name of the WSDL operation to associate with this request message struct. The default is the name of the C struct the annotation is applied to omitting any preceding namespace prefix and portType name.
- **response : NMToken (0..1)** – specifies the name of the struct that defines the format of the response message.
- **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 binary : (0..1)** – specifies whether this struct is included in the web service interface. The default value is "false".

2805 **Applies to:** Struct.

2806 Example:

2807 Input C header file:

```
2808 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2809 *                  serviceName="StockQuoteService") */
2810
2811 /* @WebOperation(operationName="GetLastTradePrice",
2812 *                  response="getLastTradePriceResponse"
2813 *                  action="urn:GetLastTradePrice") */
2814 struct getLastTradePriceMsg {
2815     char tickerSymbol[10];
2816 } getLastTradePrice;
2817
2818 struct getLastTradePriceResponseMsg {
2819     float return;
2820 } getLastTradePriceResponse;
```

2821 Generated WSDL file:

```
2822 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2823   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2824   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
2825   xmlns:tns="http://www.example.org/"
2826   targetNamespace="http://www.example.org/">
2827
2828   <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"
2829     xmlns:tns="http://www.example.org/"
2830     attributeFormDefault="unqualified"
2831     elementFormDefault="unqualified"
2832     targetNamespace="http://www.example.org/">
2833     <xselement name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2834     <xselement name="GetLastTradePriceResponse"
2835       type="tns:GetLastTradePriceResponse"/>
2836   <xssimpleType name="TickerSymbolType">
```

```

2838         <xs:restriction base="xs:string">
2839             <xsd:maxlength value="9"/>
2840         </xs:restriction>
2841     </xs:simpleType>
2842     <xs:complexType name="GetLastTradePrice">
2843         <xs:sequence>
2844             <xs:element name="tickerSymbol" type="TickerSymbolType"/>
2845         </xs:sequence>
2846     </xs:complexType>
2847     <xs:complexType name="GetLastTradePriceResponse">
2848         <xs:sequence>
2849             <xs:element name="return" type="xs:float"/>
2850         </xs:sequence>
2851     </xs:complexType>
2852 </xs:schema>
2853
2854 < message name="GetLastTradePrice">
2855     <sca-c:bindings>
2856         <sca-c:struct name="getLastTradePrice"/>
2857     </sca-c:bindings>
2858     <part name="parameters" element="tns:GetLastTradePrice">
2859     </part>
2860 </message>
2861
2862 < message name="GetLastTradePriceResponse">
2863     <sca-c:bindings>
2864         <sca-c:struct name="getLastTradePriceResponse"/>
2865     </sca-c:bindings>
2866     <part name="parameters" element="tns:GetLastTradePriceResponse">
2867     </part>
2868 </ message>
2869
2870 <portType name="StockQuote">
2871     <sca-c:bindings>
2872         <sca-c:prefix name="stockQuote"/>
2873     </sca-c:bindings>
2874     <operation name="GetLastTradePrice">
2875         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2876         </input>
2877         <output name="GetLastTradePriceResponse"
2878             message="tns:GetLastTradePriceResponse">
2879         </output>
2880     </operation>
2881 </portType>
2882
2883 <binding name="StockQuoteServiceSoapBinding">
2884     <soap:binding style="document"
2885         transport="http://schemas.xmlsoap.org/soap/http"/>
2886     <wsdl:operation name="GetLastTradePrice">
2887         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2888         <wsdl:input name="GetLastTradePrice">
2889             <soap:body use="literal"/>
2890         </wsdl:input>
2891         <wsdl:output name="GetLastTradePriceResponse">
2892             <soap:body use="literal"/>
2893         </wsdl:output>
2894     </wsdl:operation>
2895 </binding>
2896
2897 <service name="StockQuoteService">
2898     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2899         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2900     </port>
2901 </service>

```

```
2902     </definitions>
```

2903 C.1.4 @OneWay

2904 Annotation on a C function indicating that it represents a one-way request. The @OneWay annotation
2905 also affects the service interface. See @OneWay.

2906 **Corresponds to:** javax.jws.OneWay annotation in the JAX-WS specification (7.11.3)

2907 **Format:**

```
2908     /* @OneWay */
```

2909 **Applies to:** Function.

2910 Example:

2911 Input C header file:

```
2912     /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
2913      *             serviceName="StockQuoteService") */  
2914  
2915     /* @WebFunction(operationName="SetTradePrice",  
2916      *             action="urn:SetTradePrice")  
2917      * @OneWay */  
2918     void setTradePrice(const char *tickerSymbol, float price);
```

2919

2920 Generated WSDL file:

```
2921 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
2922   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
2923   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
2924   xmlns:tns="http://www.example.org/"  
2925   targetNamespace="http://www.example.org/">  
2926  
2927   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
2928     xmlns:tns="http://www.example.org/"  
2929     attributeFormDefault="unqualified"  
2930     elementFormDefault="unqualified"  
2931     targetNamespace="http://www.example.org/">  
2932   <xsd:element name="SetTradePrice" type="tns:SetTradePrice"/>  
2933   <xsd:complexType name="SetTradePrice">  
2934     <xsd:sequence>  
2935       <xsd:element name="tickerSymbol" type="xsd:string"/>  
2936       <xsd:element name="price" type="xsd:float"/>  
2937     </xsd:sequence>  
2938   </xsd:complexType>  
2939 </xsd:schema>  
2940  
2941   < message name="SetTradePrice">  
2942     <part name="parameters" element="tns:SetTradePrice">  
2943     </part>  
2944   </message>  
2945  
2946   <portType name="StockQuote">  
2947     <sca-c:bindings>  
2948       <sca-c:prefix name="stockQuote"/>  
2949     </sca-c:bindings>  
2950     <operation name="SetTradePrice">  
2951       <sca-c:bindings>  
2952         <sca-c:function name="setTradePrice"/>  
2953       </sca-c:bindings>  
2954       <input name="SetTradePrice" message="tns:SetTradePrice">  
2955         </input>  
2956       </operation>  
2957   </portType>
```

```

2959     <binding name="StockQuoteServiceSoapBinding">
2960         <soap:binding style="document"
2961             transport="http://schemas.xmlsoap.org/soap/http"/>
2962         <wsdl:operation name="SetTradePrice">
2963             <soap:operation soapAction="urn:SetTradePrice" style="document"/>
2964             <wsdl:input name="SetTradePrice">
2965                 <soap:body use="literal"/>
2966             </wsdl:input>
2967         </wsdl:operation>
2968     </binding>
2969
2970     <service name="StockQuoteService">
2971         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2972             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2973         </port>
2974     </service>
2975 </definitions>

```

2976 C.1.5 @WebParam

2977 Annotation on a C function indicating the mapping of a parameter to the associated input and output
 2978 WSDL messages. Or on a C struct indicating the mapping of a member to the associated WSDL
 2979 message.

2980 **Corresponds to:** javax.jws.WebParam annotation in the JAX-WS specification (7.11.4)

2981 **Format:**

```

2982     /* @WebParam(paramName="parameter", name="WSDLElement",
2983      *           targetNamespace="namespaceURI", mode="IN" | "OUT" | "INOUT",
2984      *           header="false", partName="WSDLPart", type="xsdType") */

```

2985 where:

- **paramName : NCName (1..1)** – specifies the name of the parameter that this annotation applies to.
 Only named parameters MAY be referenced by a @WebParam annotation. [CC0009]
- **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default value is the name of the parameter. If an @WebParam annotation is not present, and the parameter is unnamed, then a name of “argN”, where N is an incrementing value from 1 indicating the position of the parameter in the argument list, will be used.
- **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default namespace is the namespace of the associated @WebService. The targetNamespace attribute is ignored unless the binding style is document, and the binding parameterStyle is bare. See @SOAPBinding.
- **mode : token (0..1)** – specifies whether the parameter is associated with the input message, output message, or both. The default value is determined by the passing mechanism for the parameter. See Method Parameters and Return Type.
- **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>. [CC0006] The default type is determined by the mapping defined in Simple Content Binding.

3007 **Applies to:** Function parameter or struct member.

3008 Example:

3009 Input C header file:

```

3010   /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/", */

```

```

3011     *             serviceName="StockQuoteService") */
3012
3013     /* @WebFunction(operationName="GetLastTradePrice",
3014      *             action="urn:GetLastTradePrice")
3015      * @WebParam(paramName="tickerSymbol", name="symbol", mode="IN") */
3016     float getLastTradePrice(char *tickerSymbol);

```

3017

3018 Generated WSDL file:

```

3019 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3020   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3021   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3022   xmlns:tns="http://www.example.org/"
3023   targetNamespace="http://www.example.org/">
3024
3025   <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3026     xmlns:tns="http://www.example.org/"
3027     attributeFormDefault="unqualified"
3028     elementFormDefault="unqualified"
3029     targetNamespace="http://www.example.org/">
3030     <xselement name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3031     <xselement name="GetLastTradePriceResponse"
3032       type="tns:GetLastTradePriceResponse"/>
3033     <xsccomplexType name="GetLastTradePrice">
3034       <xsssequence>
3035         <xselement name="symbol" type="xs:string"/>
3036       </xsssequence>
3037     </xsccomplexType>
3038     <xsccomplexType name="GetLastTradePriceResponse">
3039       <xsssequence>
3040         <xselement name="return" type="xs:float"/>
3041       </xsssequence>
3042     </xsccomplexType>
3043   </xsschema>
3044
3045   < message name="GetLastTradePrice">
3046     <part name="parameters" element="tns:GetLastTradePrice">
3047     </part>
3048   </message>
3049
3050   < message name="GetLastTradePriceResponse">
3051     <part name="parameters" element="tns:GetLastTradePriceResponse">
3052     </part>
3053   </message>
3054
3055   <portType name="StockQuote">
3056     <sca-c:bindings>
3057       <sca-c:prefix name="stockQuote"/>
3058     </sca-c:bindings>
3059     <operation name="GetLastTradePrice">
3060       <sca-c:bindings>
3061         <sca-c:function name="getLastTradePrice"/>
3062         <sca-c:parameter name="tickerSymbol"
3063           part="tns:GetLastTradePrice/parameter"
3064           childElementName="symbol"/>
3065       </sca-c:bindings>
3066       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3067       </input>
3068       <output name="GetLastTradePriceResponse"
3069         message="tns:GetLastTradePriceResponse">
3070       </output>
3071     </operation>
3072   </portType>
3073

```

```

3074 <binding name="StockQuoteServiceSoapBinding">
3075   <soap:binding style="document"
3076     transport="http://schemas.xmlsoap.org/soap/http"/>
3077   <wsdl:operation name="GetLastTradePrice">
3078     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3079     <wsdl:input name="GetLastTradePrice">
3080       <soap:body use="literal"/>
3081     </wsdl:input>
3082     <wsdl:output name="GetLastTradePriceResponse">
3083       <soap:body use="literal"/>
3084     </wsdl:output>
3085   </wsdl:operation>
3086 </binding>
3087
3088 <service name="StockQuoteService">
3089   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3090     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3091   </port>
3092 </service>
3093 </definitions>

```

3094 C.1.6 @WebResult

3095 Annotation on a C function indicating the mapping of the function's return type to the associated output
 3096 WSDL message.

3097 **Corresponds to:** javax.jws.WebResult annotation in the JAX-WS specification (7.11.5)

3098 **Format:**

```

3099 /* @WebResult(name="WSDLElement", targetNamespace="namespaceURI",
3100 *           header="false", partName="WSDLPart", type="xsdType") */

```

3101 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>. [CC0007] The default type is determined by the mapping defined in 11.3.1.

3116 **Applies to:** Function.

3117 Example:

3118 Input C header file:

```

3119 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3120 *               serviceName="StockQuoteService") */
3121
3122 /* @WebFunction(operationName="GetLastTradePrice",
3123 *               action="urn:GetLastTradePrice")
3124 * @WebResult(name="price") */
3125 float getLastTradePrice(const char *tickerSymbol);

```

3126

```

3127 Generated WSDL file:
3128 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3129   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3130   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3131   xmlns:tns="http://www.example.org/"
3132   targetNamespace="http://www.example.org/">
3133
3134   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3135     xmlns:tns="http://www.example.org/"
3136     attributeFormDefault="unqualified"
3137     elementFormDefault="unqualified"
3138     targetNamespace="http://www.example.org/">
3139     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3140     <xs:element name="GetLastTradePriceResponse"
3141       type="tns:GetLastTradePriceResponse"/>
3142     <xs:complexType name="GetLastTradePrice">
3143       <xs:sequence>
3144         <xs:element name="tickerSymbol" type="xs:string"/>
3145       </xs:sequence>
3146     </xs:complexType>
3147     <xs:complexType name="GetLastTradePriceResponse">
3148       <xs:sequence>
3149         <xs:element name="price" type="xs:float"/>
3150       </xs:sequence>
3151     </xs:complexType>
3152   </xs:schema>
3153
3154   < message name="GetLastTradePrice">
3155     <part name="parameters" element="tns:GetLastTradePrice">
3156     </part>
3157   </message>
3158
3159   < message name="GetLastTradePriceResponse">
3160     <part name="parameters" element="tns:GetLastTradePriceResponse">
3161     </part>
3162   </message>
3163
3164   <portType name="StockQuote">
3165     <sca-c:bindings>
3166       <sca-c:prefix name="stockQuote"/>
3167     </sca-c:bindings>
3168     <operation name="GetLastTradePrice">
3169       <sca-c:bindings>
3170         <sca-c:function name="getLastTradePrice"/>
3171       </sca-c:bindings>
3172       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3173       </input>
3174       <output name="GetLastTradePriceResponse"
3175         message="tns:GetLastTradePriceResponse">
3176       </output>
3177     </operation>
3178   </portType>
3179
3180   <binding name="StockQuoteServiceSoapBinding">
3181     <soap:binding style="document"
3182       transport="http://schemas.xmlsoap.org/soap/http"/>
3183     <wsdl:operation name="GetLastTradePrice">
3184       <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3185       <wsdl:input name="GetLastTradePrice">
3186         <soap:body use="literal"/>
3187       </wsdl:input>
3188       <wsdl:output name="GetLastTradePriceResponse">
3189         <soap:body use="literal"/>
3190       </wsdl:output>

```

```

3191     </wsdl:operation>
3192   </binding>
3193
3194   <service name="StockQuoteService">
3195     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3196       <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3197     </port>
3198   </service>
3199 </definitions>
```

3200 C.1.7 @SOAPBinding

3201 Annotation on a C WebService or function specifying the mapping of the web service onto the SOAP
 3202 message protocol.

3203 **Corresponds to:** javax.jws.SOAPBinding annotation in the JAX-WS specification (7.11.6)

3204 **Format:**

```

3205  /* @SOAPBinding(style="DOCUMENT" | "RPC", use="LITERAL" | "ENCODED",
3206   *           parameterStyle="BARE" | "WRAPPED") */
```

3207 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”.

3212 **Applies to:** WebService, Function.

3213 Example:

3214 Input C header file:

```

3215  /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3216   *           serviceName="StockQuoteService") */
3217  * @SOAPBinding(style="RPC") */
3218 ...
3219 ...
```

3220

3221 Generated WSDL file:

```

3222 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3223   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3224   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3225   xmlns:tns="http://www.example.org/"
3226   targetNamespace="http://www.example.org/">
3227
3228   <portType name="StockQuote">
3229     <sca-c:bindings>
3230       <sca-c:prefix name="stockQuote"/>
3231     </sca-c:bindings>
3232   </portType>
3233
3234   <binding name="StockQuoteServiceSoapBinding">
3235     <soap:binding style="rpc"
3236       transport="http://schemas.xmlsoap.org/soap/http"/>
3237   </binding>
3238
3239   <service name="StockQuoteService">
3240     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3241       <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3242     </port>
3243   </service>
3244 </definitions>
```

3245 C.1.8 @WebFault

3246 Annotation on a C struct indicating that it format of a fault message.

3247 **Corresponds to:** javax.xml.ws.WebFault annotation in the JAX-WS specification (7.2)

3248 **Format:**

```
3249     /* @WebFault(name="WSDLElement", targetNamespace="namespaceURI") */
```

3250 where:

- **name : NCName (1..1)** – specifies the 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.

3254 **Applies to:** struct.

3255 Example:

3256 Input C header file:

```
3257     /* @WebFault(name="UnknownSymbolFault",
3258      *           targetNamespace="http://www.example.org/")
3259     struct UnkSymMsg {
3260       char faultInfo[10];
3261     } unkSymInfo;
3262
3263     /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3264      *           serviceName="StockQuoteService") */
3265
3266     /* @WebFunction(operationName="GetLastTradePrice",
3267      *           action="urn:GetLastTradePrice")
3268      * @WebThrows(faults="unkSymInfo") */
3269     float getLastTradePrice(const char *tickerSymbol);
```

3270

3271 Generated WSDL file:

```
3272 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3273   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3274   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3275   xmlns:tns="http://www.example.org/"
3276   targetNamespace="http://www.example.org/">
3277
3278   <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"
3279     xmlns:tns="http://www.example.org/"
3280     attributeFormDefault="unqualified"
3281     elementFormDefault="unqualified"
3282     targetNamespace="http://www.example.org/">
3283     <xselement name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3284     <xselement name="GetLastTradePriceResponse"
3285       type="tns:GetLastTradePriceResponse"/>
3286     <xsccomplexType name="GetLastTradePrice">
3287       <xsssequence>
3288         <xselement name="tickerSymbol" type="xs:string"/>
3289       </xsssequence>
3290     </xsccomplexType>
3291     <xsccomplexType name="GetLastTradePriceResponse">
3292       <xsssequence>
3293         <xselement name="return" type="xs:float"/>
3294       </xsssequence>
3295     </xsccomplexType>
3296     <xssimpleType name="UnknownSymbolFaultType">
3297       <xssrestriction base="xs:string">
3298         <xsd:maxlength value="9"/>
3299       </xssrestriction>
3300     </xssimpleType>
3301     <xselement name="UnknownSymbolFault" type="UnknownSymbolFaultType"/>
```

```

3302 </xs:schema>
3303
3304 <message name="GetLastTradePrice">
3305   <part name="parameters" element="tns:GetLastTradePrice">
3306     </part>
3307 </message>
3308
3309 <message name="GetLastTradePriceResponse">
3310   <part name="parameters" element="tns:GetLastTradePriceResponse">
3311     </part>
3312 </message>
3313
3314 <message name="UnknownSymbol">
3315   <sca-c:bindings>
3316     <sca-c:struct name="unkSymInfo"/>
3317   </sca-c:bindings>
3318   <part name="parameters" element="tns:UnknownSymbolFault">
3319     </part>
3320 </message>
3321
3322 <portType name="StockQuote">
3323   <sca-c:bindings>
3324     <sca-c:prefix name="stockQuote"/>
3325   </sca-c:bindings>
3326   <operation name="GetLastTradePrice">
3327     <sca-c:bindings>
3328       <sca-c:function name="getLastTradePrice"/>
3329     </sca-c:bindings>
3330     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3331     </input>
3332     <output name="GetLastTradePriceResponse"
3333       message="tns:GetLastTradePriceResponse">
3334     </output>
3335     <fault name="UnknownSymbol" message="tns:UnknownSymbol">
3336       </fault>
3337     </operation>
3338   </portType>
3339
3340 <binding name="StockQuoteServiceSoapBinding">
3341   <soap:binding style="document"
3342     transport="http://schemas.xmlsoap.org/soap/http"/>
3343   <wsdl:operation name="GetLastTradePrice">
3344     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3345     <wsdl:input name="GetLastTradePrice">
3346       <soap:body use="literal"/>
3347     </wsdl:input>
3348     <wsdl:output name="GetLastTradePriceResponse">
3349       <soap:body use="literal"/>
3350     </wsdl:output>
3351     <wsdl:fault>
3352       <soap:fault name="UnknownSymbol" use="literal"/>
3353     </wsdl:fault>
3354   </wsdl:operation>
3355 </binding>
3356
3357 <service name="StockQuoteService">
3358   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3359     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3360   </port>
3361 </service>
3362 </definitions>

```

3363 **C.1.9 @WebThrows**

3364 Annotation on a C function or operation indicating which faults might be thrown by this function or
3365 operation.

3366 **Corresponds to:** No equivalent in JAX-WS.

3367 **Format:**

3368

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

3369 where:

- 3370 • ***faults : NMOKEN (1..n)*** – specifies the names of all faults that might be thrown by this function or
3371 operation. The name of the fault is the name of its associated C struct name. A C struct that is listed
3372 in a @WebThrows annotation MUST itself have a @WebFault annotation. [\[CC0004\]](#)

3373 **Applies to:** Function or Operation

3374 Example:

3375 See @WebFault.

3376 D C WSDL Mapping Extensions

3377 The following WSDL extensions are used to augment the conversion process from WSDL to C. All of
3378 these extensions are defined in the namespace `http://docs.oasis-open.org/ns/opencsa/sca-c-`
3379 `cpp/c/200901`. For brevity, all definitions of these extensions will be fully qualified, and all references to
3380 the “`sca-c`” prefix are associated with the namespace above. If WSDL extensions are supported by an
3381 implementation, all the extensions defined here MUST be supported and MUST be mapped to C as
3382 described. [CD0001]

3383 D.1 <`sca-c:bindings`>

3384 `<sca-c:bindings>` is a container type which can be used as a WSDL extension. All other SCA wsdl
3385 extensions will be specified as children of a `<sca-c:bindings>` element. An `<sca-c:bindings>` element can
3386 be used as an extension to any WSDL type that accepts extensions.

3387 D.2 <`sca-c:prefix`>

3388 `<sca-c:prefix>` provides a mechanism for defining an alternate prefix for the functions or structs
3389 implementing the operations of a `portType`.

3390 Format:

```
3391     <sca-c:prefix name="portTypePrefix">
```

3392 where:

- 3393 • **`prefix/@name : string (1..1)`** – specifies the string to prepend to an operation name when generating
3394 a C function or structure name.

3395 Applicable WSDL element(s):

- 3396 • `wsdl:portType`

3397 A `<sca-c:bindings/>` element MUST NOT have more than one `< sca-c:prefix>` child element. [CD0003]

3398 Example:

3399 Input WSDL file:

```
3400 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3401   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3402   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
3403   xmlns:tns="http://www.example.org/"  
3404   targetNamespace="http://www.example.org/">  
3405  
3406   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
3407     xmlns:tns="http://www.example.org/"  
3408     attributeFormDefault="unqualified"  
3409     elementFormDefault="unqualified"  
3410     targetNamespace="http://www.example.org/">  
3411     <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
3412     <xsd:element name="GetLastTradePriceResponse"  
3413       type="tns:GetLastTradePriceResponse"/>  
3414     <xsd:complexType name="GetLastTradePrice">  
3415       <xsd:sequence>  
3416         <xsd:element name="tickerSymbol" type="xsd:string"/>  
3417       </xsd:sequence>  
3418     </xsd:complexType>  
3419     <xsd:complexType name="GetLastTradePriceResponse">  
3420       <xsd:sequence>  
3421         <xsd:element name="return" type="xsd:float"/>  
3422       </xsd:sequence>  
3423     </xsd:complexType>  
3424   </xsd:schema>
```

```

3425
3426      < message name="GetLastTradePrice">
3427          <part name="parameters" element="tns:GetLastTradePrice">
3428          </part>
3429      </message>
3430
3431      < message name="GetLastTradePriceResponse">
3432          <part name="parameters" element="tns:GetLastTradePriceResponse">
3433          </part>
3434      </ message>
3435
3436      <portType name="StockQuote">
3437          <sca-c:bindings>
3438              <sca-c:prefix name="stockQuote"/>
3439          </sca-c:bindings>
3440          <operation name="GetLastTradePrice">
3441              <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3442                  </input>
3443              <output name="GetLastTradePriceResponse"
3444                  message="tns:GetLastTradePriceResponse">
3445                  </output>
3446          </operation>
3447      </portType>
3448
3449      <binding name="StockQuoteServiceSoapBinding">
3450          <soap:binding style="document"
3451              transport="http://schemas.xmlsoap.org/soap/http"/>
3452          <wsdl:operation name="GetLastTradePrice">
3453              <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3454              <wsdl:input name="GetLastTradePrice">
3455                  <soap:body use="literal"/>
3456              </wsdl:input>
3457              <wsdl:output name="GetLastTradePriceResponse">
3458                  <soap:body use="literal"/>
3459              </wsdl:output>
3460          </wsdl:operation>
3461      </binding>
3462
3463      <service name="StockQuoteService">
3464          <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3465              <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3466          </port>
3467      </service>
3468  </definitions>

```

3469

3470 Generated C header file:

```

3471  /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3472   *             serviceName="StockQuoteService") */
3473
3474  /* @WebFunction(operationName="GetLastTradePrice",
3475   *             action="urn:GetLastTradePrice") */
3476  float stockQuoteGetLastTradePrice(const char *tickerSymbol);

```

3477 D.3 <sca-c:enableWrapperStyle>

3478 <sca-c:enableWrapperStyle> indicates whether or not the wrapper style for messages is applied, when
 3479 otherwise applicable. If false, the wrapper style will never be applied.

3480 **Format:**

```

3481      <sca-c:enableWrapperStyle>value</sca-c:enableWrapperStyle>

```

3482 where:

- 3483 • ***enableWrapperStyle/text() : boolean (1..1)*** – specifies whether wrapper style is enabled or disabled
 3484 for this element and any of its children. The default value is “*true*”.

3485 **Applicable WSDL element(s):**

- 3486 • wsdl:definitions
 3487 • wsdl:portType – overrides a binding applied to wsdl:definitions
 3488 • wsdl:portType/wsdl:operation – overrides a binding applied to wsdl:definitions or the enclosing
 3489 wsdl:portType

3490 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:enableWrapperStyle/> child
 3491 element. [CD0004]

3492 Example:

3493 Input WSDL file:

```

3494 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3495   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3496   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3497   xmlns:tns="http://www.example.org/"
3498   targetNamespace="http://www.example.org/">
3499
3500   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3501     xmlns:tns="http://www.example.org/"
3502     attributeFormDefault="unqualified"
3503     elementFormDefault="unqualified"
3504     targetNamespace="http://www.example.org/">
3505     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3506     <xs:element name="GetLastTradePriceResponse"
3507       type="tns:GetLastTradePriceResponse"/>
3508     <xs:complexType name="GetLastTradePrice">
3509       <xs:sequence>
3510         <xs:element name="tickerSymbol" type="xs:string"/>
3511       </xs:sequence>
3512     </xs:complexType>
3513     <xs:complexType name="GetLastTradePriceResponse">
3514       <xs:sequence>
3515         <xs:element name="return" type="xs:float"/>
3516       </xs:sequence>
3517     </xs:complexType>
3518   </xs:schema>
3519
3520   < message name="GetLastTradePrice">
3521     <part name="parameters" element="tns:GetLastTradePrice">
3522     </part>
3523   </message>
3524
3525   < message name="GetLastTradePriceResponse">
3526     <part name="parameters" element="tns:GetLastTradePriceResponse">
3527     </part>
3528   </message>
3529
3530   <portType name="StockQuote">
3531     <sca-c:bindings>
3532       <sca-c:prefix name="stockQuote"/>
3533       <sca-c:enableWrapperStyle>false</sca-c:enableWrapperStyle>
3534     </sca-c:bindings>
3535     <operation name="GetLastTradePrice">
3536       <sca-c:bindings>
3537         <sca-c:function name="getLastTradePrice"/>
3538       </sca-c:bindings>
3539       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3540       </input>
3541       <output name="GetLastTradePriceResponse"
3542         message="tns:GetLastTradePriceResponse">
```

```

3543         </output>
3544     </operation>
3545   </portType>
3546 </definitions>
3547
3548 Generated C header file:
3549 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3550 *           serviceName="StockQuoteService") */
3551
3552 /* @WebFunction(operationName="GetLastTradePrice",
3553 *               action="urn:GetLastTradePrice") */
3554 DATAOBJECT getLastTradePrice(DATAOBJECT parameters);

```

3555 D.4 <sca-c:function>

3556 <sca-c:function> specifies the name of the C function that the associated WSDL operation is associated
 3557 with. If <sca-c:function> is used, the portType prefix, either default or a specified with <sca-c:prefix> is
 3558 not prepended to the function name.

3559 **Format:**

```
3560   <sca-c:function name="myFunction"/>
```

3561 where:

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

3564 **Applicable WSDL element(s):**

- wsdl:portType/wsdl:operation

3566 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:function/> child element. [CD0005]

3567 Example:

3568 Input WSDL file:

```

3569 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3570   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3571   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3572   xmlns:tns="http://www.example.org/"
3573   targetNamespace="http://www.example.org/">
3574
3575   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3576     xmlns:tns="http://www.example.org/"
3577     attributeFormDefault="unqualified"
3578     elementFormDefault="unqualified"
3579     targetNamespace="http://www.example.org/">
3580     <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3581     <xsd:element name="GetLastTradePriceResponse"
3582       type="tns:GetLastTradePriceResponse"/>
3583     <xsd:complexType name="GetLastTradePrice">
3584       <xsd:sequence>
3585         <xsd:element name="tickerSymbol" type="xsd:string"/>
3586       </xsd:sequence>
3587     </xsd:complexType>
3588     <xsd:complexType name="GetLastTradePriceResponse">
3589       <xsd:sequence>
3590         <xsd:element name="return" type="xsd:float"/>
3591       </xsd:sequence>
3592     </xsd:complexType>
3593   </xsd:schema>
3594
3595   < message name="GetLastTradePrice">
3596     <part name="parameters" element="tns:GetLastTradePrice">
3597       </part>

```

```

3598     </message>
3599
3600     < message name="GetLastTradePriceResponse">
3601         <part name="parameters" element="tns:GetLastTradePriceResponse">
3602             </part>
3603     </ message>
3604
3605     <portType name="StockQuote">
3606         <sca-c:bindings>
3607             <sca-c:prefix name="stockQuote"/>
3608         </sca-c:bindings>
3609         <operation name="GetLastTradePrice">
3610             <sca-c:bindings>
3611                 <sca-c:function name="getTradePrice"/>
3612             </sca-c:bindings>
3613             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3614                 </input>
3615             <output name="GetLastTradePriceResponse"
3616                 message="tns:GetLastTradePriceResponse">
3617                 </output>
3618             </operation>
3619         </portType>
3620     </definitions>

```

3621

3622 Generated C header file:

```

3623 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3624 *               serviceName="StockQuoteService") */
3625
3626 /* @WebFunction(operationName="GetLastTradePrice",
3627 *               action="urn:GetLastTradePrice") */
3628 float getTradePrice(const wchar_t *tickerSymbol);

```

3629 D.5 <sca-c:struct>

3630 <sca-c:struct> specifies the name of the C struct that the associated WSDL message is associated with. If
3631 <sca-c:struct> is used for an operation request or response message, the portType prefix, either default
3632 or a specified with <sca-c:prefix> is not prepended to the struct name.

3633 **Format:**

```

3634     <sca-c:struct name="myStruct"/>

```

3635 where:

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

3638 **Applicable WSDL element(s):**

- wsdl:message

3640 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:struct/> child element. [CD0006]

3641 Example:

3642 Input WSDL file:

```

3643 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3644   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3645   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3646   xmlns:tns="http://www.example.org/"
3647   targetNamespace="http://www.example.org/">
3648
3649   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3650     xmlns:tns="http://www.example.org/"
3651     attributeFormDefault="unqualified"
3652     elementFormDefault="unqualified"

```

```

3653         targetNamespace="http://www.example.org/">
3654     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3655     <xs:element name="GetLastTradePriceResponse"
3656         type="tns:GetLastTradePriceResponse"/>
3657     <xs:complexType name="GetLastTradePrice">
3658         <xs:sequence>
3659             <xs:element name="tickerSymbol" type="xs:string"/>
3660         </xs:sequence>
3661     </xs:complexType>
3662     <xs:complexType name="GetLastTradePriceResponse">
3663         <xs:sequence>
3664             <xs:element name="return" type="xs:float"/>
3665         </xs:sequence>
3666     </xs:complexType>
3667 </xs:schema>
3668
3669 < message name="GetLastTradePrice">
3670     <sca-c:bindings>
3671         <sca-c:struct name="getTradePrice"/>
3672     </sca-c:bindings>
3673     <part name="parameters" element="tns:GetLastTradePrice">
3674     </part>
3675 </message>
3676
3677 < message name="GetLastTradePriceResponse">
3678     <sca-c:bindings>
3679         <sca-c:struct name="getTradePriceResponse"/>
3680     </sca-c:bindings>
3681     <part name="parameters" element="tns:GetLastTradePriceResponse">
3682     </part>
3683 </ message>
3684
3685 <portType name="StockQuote">
3686     <sca-c:bindings>
3687         <sca-c:prefix name="stockQuote"/>
3688     </sca-c:bindings>
3689     <operation name="GetLastTradePrice">
3690         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3691         </input>
3692         <output name="GetLastTradePriceResponse"
3693             message="tns:GetLastTradePriceResponse">
3694             </output>
3695         </operation>
3696     </portType>
3697 </definitions>
3698

```

Generated C header file:

```

3700 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3701 *                 serviceName="StockQuoteService") */
3702
3703 /* @WebOperation(operationName="GetLastTradePrice",
3704 *                 response="getLastTradePriceResponse"
3705 *                 action="urn:GetLastTradePrice") */
3706 struct getLastTradePrice {
3707     wchar_t *tickerSymbol; /* Since the length of the element is not
3708                             * restricted, a pointer is returned with the
3709                             * actual value held by the SCA runtime. */
3710 };
3711
3712 struct getLastTradePriceResponse {
3713     float return;
3714 };

```

3715 D.6 <sca-c:parameter>

3716 <sca-c:parameter> specifies the name of the C function parameter or struct member associated with a
3717 specific WSDL message part or wrapper child element.

3718 Format:

```
3719   <sca-c:parameter name="CParameter" part="WSDLPart"
3720     childElementName="WSDLElement" type="CType"/>
```

3721 where:

- **parameter/@name : NCName (1..1)** – specifies the name of the C function parameter or struct member associated with this WSDL operation part or wrapper child element. “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.
- **type : NCName (0..1)** – specifies the type of the parameter or struct member or return type. The @type attribute of a <parameter/> element MUST be a valid C type. [CD0002] The default type is determined by the mapping defined in Data Binding.

3731 Applicable WSDL element(s):

- wsdl:portType/wsdl:operation

3733 Example:

3734 Input WSDL file:

```
3735 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3736   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3737   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3738   xmlns:tns="http://www.example.org/"
3739   targetNamespace="http://www.example.org/">
3740
3741   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3742     xmlns:tns="http://www.example.org/"
3743     attributeFormDefault="unqualified"
3744     elementFormDefault="unqualified"
3745     targetNamespace="http://www.example.org/">
3746     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3747     <xs:element name="GetLastTradePriceResponse"
3748       type="tns:GetLastTradePriceResponse"/>
3749     <xs:complexType name="GetLastTradePrice">
3750       <xs:sequence>
3751         <xs:element name="symbol" type="xs:string"/>
3752       </xs:sequence>
3753     </xs:complexType>
3754     <xs:complexType name="GetLastTradePriceResponse">
3755       <xs:sequence>
3756         <xs:element name="return" type="xs:float"/>
3757       </xs:sequence>
3758     </xs:complexType>
3759   </xs:schema>
3760
3761   < message name="GetLastTradePrice">
3762     <part name="parameters" element="tns:GetLastTradePrice">
3763     </part>
3764   </message>
3765
3766   < message name="GetLastTradePriceResponse">
3767     <part name="parameters" element="tns:GetLastTradePriceResponse">
3768     </part>
3769   </ message>
```

```

3771 <portType name="StockQuote">
3772   <sca-c:bindings>
3773     <sca-c:prefix name="stockQuote"/>
3774   </sca-c:bindings>
3775   <operation name="GetLastTradePrice">
3776     <sca-c:bindings>
3777       <sca-c:function name="getLastTradePrice"/>
3778       <sca-c:parameter name="tickerSymbol"
3779         part="tns:GetLastTradePrice/parameter"
3780         childElementName="symbol"/>
3781     </sca-c:bindings>
3782     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3783     </input>
3784     <output name="GetLastTradePriceResponse"
3785       message="tns:GetLastTradePriceResponse">
3786     </output>
3787   </operation>
3788 </portType>
3789
3790 <binding name="StockQuoteServiceSoapBinding">
3791   <soap:binding style="document"
3792     transport="http://schemas.xmlsoap.org/soap/http"/>
3793   <wsdl:operation name="GetLastTradePrice">
3794     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3795     <wsdl:input name="GetLastTradePrice">
3796       <soap:body use="literal"/>
3797     </wsdl:input>
3798     <wsdl:output name="GetLastTradePriceResponse">
3799       <soap:body use="literal"/>
3800     </wsdl:output>
3801   </wsdl:operation>
3802 </binding>
3803
3804 <service name="StockQuoteService">
3805   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3806     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3807   </port>
3808 </service>
3809 </definitions>
3810
3811 Generated C header file:
3812 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3813 *               serviceName="StockQuoteService") */
3814
3815 /* @WebFunction(operationName="GetLastTradePrice",
3816 *               action="urn:GetLastTradePrice")
3817 * @WebParam(paramName="tickerSymbol", name="symbol") */
3818 float getLastTradePrice(const wchar_t *tickerSymbol);

```

3819 D.7 JAX-WS WSDL Extensions

3820 An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL
 3821 extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.
 3822 [CD0007] The following is a list of JAX-WS WSDL extensions that MAY be recognized, and their
 3823 corresponding SCA WSDL extension.

JAX-WS Extension	SCA Extension
jaxws:bindings	sca-c:bindings

jaxws:class	sca-c:prefix
jaxws:method	sca-c:function
jaxws:parameter	sca-c:parameter
jaxws:enableWrapperStyle	sca-c:enableWrapperStyle

3825 D.8 WSDL Extensions Schema

3826 The XML schema pointed to by the RDDL document at the SCA C namespace URI, defined by this
 3827 specification, is considered to be authoritative and takes precedence over the XML schema in this
 3828 appendix.

3829

```

3830 <?xml version="1.0" encoding="UTF-8"?>
3831 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3832   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3833   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3834   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3835   elementFormDefault="qualified">
3836
3837   <element name="bindings" type="sca-c:BindingsType" />
3838   <complexType name="BindingsType">
3839     <choice minOccurs="0" maxOccurs="unbounded">
3840       <element ref="sca-c:prefix" />
3841       <element ref="sca-c:enableWrapperStyle" />
3842       <element ref="sca-c:function" />
3843       <element ref="sca-c:struct" />
3844       <element ref="sca-c:parameter" />
3845     </choice>
3846   </complexType>
3847
3848   <element name="prefix" type="sca-c:PrefixType" />
3849   <complexType name="PrefixType">
3850     <attribute name="name" type="xsd:string" use="required" />
3851   </complexType>
3852
3853   <element name="function" type="sca-c:FunctionType" />
3854   <complexType name="FunctionType">
3855     <attribute name="name" type="xsd:NCName" use="required" />
3856   </complexType>
3857
3858   <element name="struct" type="sca-c:StructType" />
3859   <complexType name="StructType">
3860     <attribute name="name" type="xsd:NCName" use="required" />
3861   </complexType>
3862
3863   <element name="parameter" type="sca-c:ParameterType" />
3864   <complexType name="ParameterType">
3865     <attribute name="part" type="xsd:string" use="required" />
3866     <attribute name="childElementName" type="xsd:QName" use="required" />
3867     <attribute name="name" type="xsd:NCName" use="required" />
3868     <attribute name="type" type="xsd:string" use="optional" />
3869   </complexType>
3870
3871   <element name="enableWrapperStyle" type="xsd:boolean" />
3872
3873 </schema>
```

3874 E XML Schemas

3875 Three XML schemas are defined to support the use of C for implementation and definition of interfaces.
3876 The XML schema pointed to by the RDDL document at the SCA namespace URI, defined by the
3877 Assembly specification **[ASSEMBLY]** and extended by this specification, are considered to be
3878 authoritative and take precedence over the XML schema in this appendix.

3879 E.1 sca-interface-c-1.1.xsd

```
3880 <?xml version="1.0" encoding="UTF-8"?>
3881 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3882     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3883     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3884     elementFormDefault="qualified">
3885
3886     <include schemaLocation="sca-core.xsd"/>
3887
3888     <element name="interface.c" type="sca:CInterface"
3889         substitutionGroup="sca:interface"/>
3890
3891     <complexType name="CInterface">
3892         <complexContent>
3893             <extension base="sca:Interface">
3894                 <sequence>
3895                     <element name="function" type="sca:CFunction"
3896                         minOccurs="0" maxOccurs="unbounded" />
3897                     <element name="callbackFunction" type="sca:CFunction"
3898                         minOccurs="0" maxOccurs="unbounded" />
3899                     <any namespace="#other" processContents="lax"
3900                         minOccurs="0" maxOccurs="unbounded"/>
3901                 </sequence>
3902                 <attribute name="header" type="string" use="required"/>
3903                 <attribute name="callbackHeader" type="string" use="optional"/>
3904                 <anyAttribute namespace="#other" processContents="lax"/>
3905             </extension>
3906         </complexContent>
3907     </complexType>
3908
3909     <complexType name="CFunction">
3910         <attribute name="name" type="NCName" use="required"/>
3911         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3912         <attribute name="oneWay" type="boolean" use="optional"/>
3913         <attribute name="input" type="NCName" use="optional"/>
3914         <attribute name="output" type="NCName" use="optional"/>
3915         <anyAttribute namespace="#other" processContents="lax"/>
3916     </complexType>
3917
3918 </schema>
```

3919 E.2 sca-implementation-c-1.1.xsd

```
3920 <?xml version="1.0" encoding="UTF-8"?>
3921 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3922     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3923     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3924     elementFormDefault="qualified">
3925
3926     <include schemaLocation="sca-core.xsd"/>
3927
```

```

3928     <element name="implementation.c" type="sca:CImplementation"
3929         substitutionGroup="sca:implementation" />
3930
3931     <complexType name="CImplementation">
3932         <complexContent>
3933             <extension base="sca:Implementation">
3934                 <sequence>
3935                     <element name="operation" type="sca:CImplementationFunction"
3936                         minOccurs="0" maxOccurs="unbounded" />
3937                     <any namespace="#other" processContents="lax"
3938                         minOccurs="0" maxOccurs="unbounded"/>
3939                 </sequence>
3940                 <attribute name="module" type="NCName" use="required"/>
3941                 <attribute name="path" type="string" use="optional"/>
3942                 <attribute name="library" type="boolean" use="optional"/>
3943                 <attribute name="componentType" type="string" use="required"/>
3944                 <attribute name="scope" type="sca:CImplementationScope"
3945                     use="optional"/>
3946                 <attribute name="eagerInit" type="boolean" use="optional"/>
3947                 <attribute name="init" type="boolean" use="optional"/>
3948                 <attribute name="destroy" type="boolean" use="optional"/>
3949                 <attribute name="allowsPassByReference" type="boolean"
3950                     use="optional"/>
3951                     <anyAttribute namespace="#other" processContents="lax"/>
3952                 </extension>
3953             </complexContent>
3954         </complexType>
3955
3956     <simpleType name="CImplementationScope">
3957         <restriction base="string">
3958             <enumeration value="stateless"/>
3959             <enumeration value="composite"/>
3960         </restriction>
3961     </simpleType>
3962
3963     <complexType name="CImplementationFunction">
3964         <attribute name="name" type="NCName" use="required"/>
3965         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3966         <attribute name="allowsPassByReference" type="boolean"
3967             use="optional"/>
3968         <attribute name="init" type="boolean" use="optional"/>
3969         <attribute name="destroy" type="boolean" use="optional"/>
3970         <anyAttribute namespace="#other" processContents="lax"/>
3971     </complexType>
3972
3973 </schema>

```

3974 E.3 sca-contribution-c-1.1.xsd

```

3975 <?xml version="1.0" encoding="UTF-8"?>
3976 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3977     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3978     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3979     elementFormDefault="qualified">
3980
3981     <include schemaLocation="sca-contributions.xsd"/>
3982
3983     <element name="export.c" type="sca:CExport"
3984         substitutionGroup="sca:Export"/>
3985
3986     <complexType name="CExport">
3987         <complexContent>
3988             <attribute name="name" type="QName" use="required"/>
3989             <attribute name="path" type="string" use="optional"/>

```

```
3990     </complexContent>
3991   </complexType>
3992
3993   <element name="import.c" type="sca:CImport"
3994     substitutionGroup="sca:Import"/>
3995
3996   <complexType name="CImport">
3997     <complexContent>
3998       <attribute name="name" type="QName" use="required"/>
3999       <attribute name="location" type="string" use="required"/>
4000     </complexContent>
4001   </complexType>
4002
4003 </schema>
```

4004

F Conformance Items

4005 This section contains a list of conformance items for the SCA C Client and Implementation Model
 4006 specification.

Conformance ID	Description
[C20001]	A C implementation MUST implement all of the operation(s) of the service interface(s) of its componentType.
[C20003]	An SCA runtime MUST support these scopes; stateless and composite . Additional scopes MAY be provided by SCA runtimes.
[C20004]	A C implementation MUST only designate functions with no arguments and a void return type as lifecycle functions.
[C20006]	If the header file identified by the @header attribute of an <i><interface.c/></i> element contains function declarations that are not operations of the interface, then the functions that define operations of the interface MUST be identified using <i><function/></i> child elements of the <i><interface.c/></i> element.
[C20007]	If the header file identified by the @callbackHeader attribute of an <i><interface.c/></i> element contains function declarations that are not operations of the callback interface, then the functions that define operations of the callback interface MUST be identified using <i><callbackFunction/></i> child elements of the <i><interface.c/></i> element.
[C20008]	If the header file identified by the @header or @callbackHeader attribute of an <i><interface.c/></i> element defines the operations of the interface (callback interface) using message formats, then all functions of the interface (callback interface) MUST be identified using <i><function/></i> (<i><callbackFunction/></i>) child elements of the <i><interface.c/></i> element.
[C20009]	The @name attribute of a <i><function/></i> child element of a <i><interface.c/></i> MUST be unique amongst the <i><function/></i> elements of that <i><interface.c/></i> .
[C20010]	The @name attribute of a <i><callbackFunction/></i> child element of a <i><interface.c/></i> MUST be unique amongst the <i><callbackFunction/></i> elements of that <i><interface.c/></i> .
[C20011]	If the header file identified by the @header or @callbackHeader attribute of an <i><interface.c/></i> element defines the operations of the interface (callback interface) using message formats, then the <i>struct</i> defining the input message format MUST be identified using an @input attribute.
[C20012]	If the header file identified by the @header or @callbackHeader attribute of an <i><interface.c/></i> element defines the operations of the interface (callback interface) using message formats, then the <i>struct</i> defining the output message format MUST be identified using an @input attribute.
[C20013]	The @name attribute of a <i><function/></i> child element of a <i><implementation.c/></i> MUST be unique amongst the <i><function/></i> elements of that <i><implementation.c/></i> .
[C20014]	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.
[C20015]	An SCA runtime MAY run multiple threads in a single composite scoped

	implementation instance object and it MUST NOT perform any synchronization.
[C20016]	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 function implementation and the client are marked "allows pass by reference".
[C20017]	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 function implementation is not marked "allows pass by reference" or the client is not marked "allows pass by reference".
[C30001]	An SCA implementation MAY support proxy functions.
[C40001]	An operation marked as oneWay is considered non-blocking and the SCA runtime MAY use a binding that buffers the requests to the function and sends them at some time after they are made.
[C50001]	Vendor defined reason codes SHOULD start at 101.
[C60002]	An SCA runtime MAY additionally provide a DataObject variant of this API for handling properties with complex XML types. The type of the value parameter in this variant is DATAOBJECT.
[C60003]	A SCA runtime MAY provide the functions SCAService(), SCAOperation(), SCAMessageIn() and SCAMessageOut() to support C implementations in programs.
[C70001]	The @name attribute of a <export.c/> element MUST be unique amongst the <export.c/> elements in a domain.
[C70002]	The @name attribute of a <import.c/> child element of a <contribution/> MUST be unique amongst the <import.c/> elements in of that contribution.
[C80001]	The return type and types of the parameters of a function of a local service interface MUST be one of: <ul style="list-style-type: none"> • 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 *). • DATAOBJECT. An SDO handle.
[C80002]	The return type and types of the parameters of a function of a remotable service interface MUST be one of: <ul style="list-style-type: none"> • Any of the C primitive types (for example, int, short, char). This will be copied. • DATAOBJECT. An SDO handle. The SDO will be copied and passed to the destination.
[C90001]	A C header file used to define an interface MUST: <ul style="list-style-type: none"> • Declare at least one function or message format struct
[C90002]	A C header file used to define an interface MUST NOT use the following constructs: <ul style="list-style-type: none"> • Macros
[C100001]	In the absence of customizations, an SCA implementation SHOULD map each portType to separate header file. An SCA implementation MAY use any sca-c:prefix

	binding declarations to control this mapping.
[C100002]	For components implemented in libraries, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the function.
[C100003]	In the absence of any customizations for a WSDL operation that does not meet the requirements for the wrapped style, the name of a mapped function parameter or struct member MUST be the value of the name attribute of the wsdl:part element with the first character converted to lower case.
[C100004]	In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped style, the name of a mapped function parameter or struct member MUST be the value of the local name of the wrapper child with the first character converted to lower case.
[C100005]	For components implemented in a program, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the request struct name. Additionally an SCA implementation MUST append “Response” to the request struct name to form the response struct name.
[C100006]	In the absence of customizations, an SCA implementation MUST map the name of the message element referred to by a fault element to name of the struct describing the fault message content. If necessary, to avoid name collisions, an implementation MAY append “Fault” to the name of the message element when mapping to the struct name.
[C100007]	An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be configurable.
[C100008]	In the absence of customizations, an SCA implementation MUST map the header file name to the portType name. An implementation MAY append “PortType” to the header file name in the mapping to the portType name.
[C100009]	In the absence of customizations, an SCA implementation MUST map the function name to the operation name, stripping the portType name, if present, and any namespace prefix from the function name from the front of function name before mapping it to the operation name.
[C100010]	In the absence of customizations, a struct with a name that does not end in “Response” or “Fault” is considered to be a request message struct and an SCA implementation MUST map the struct name to the operation name, stripping the portType name, if present, and any namespace prefix from the front of the struct name before mapping it to the operation name.
[C100011]	In the absence of customizations, an SCA implementation MUST map the parameter name, if present, to the part or global element component name. If the parameter does not have a name the SCA implementation MUST use argN as the part or global element child name.
[C100012]	In the absence of customizations, an SCA implementation MUST map the return type to a part or global element child named “return”.
[C100013]	Program based implementation SHOULD use the Document-Literal style and encoding.
[C100014]	In the absence of customizations, an SCA implementation MUST map the struct

	member name to the part or global element child name.
[C100015]	An SCA implementation MUST ensure that in/out parameters have the same type in the request and response structs.
[C100016]	An SCA implementation MUST support mapping message parts or global elements with complex types and parameters, return types, and struct members with a type defined by a <code>struct</code> . The mapping from WSDL MAY be to DataObjects and/or <code>structs</code> . The mapping to and from <code>structs</code> MUST follow the rules defined in WSDL to C Mapping Details.
[C100017]	An SCA implementation MUST map: <ul style="list-style-type: none"> • a function's return value as an out parameter. • by-value and const parameters as in parameters. • in the absence of customizations, pointer parameters as in/out parameters.
[C100019]	For library-based service implementations, an SCA implementation MUST map In parameters as pass by-value and In/Out and Out parameters as pass via pointers.
[C100020]	For program-based service implementations, an SCA implementation MUST map all values in the input message as pass by-value and the updated values for In/Out parameters and all Out parameters in the response message as pass by-value.
[C100021]	An SCA implementation MUST map simple types as defined in Table 1 and Table 2 by default.
[C100022]	An SCA implementation MAY map boolean to <code>_Bool</code> by default.
[C110001]	An SCA implementation MUST reject a composite file that does not conform to http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd or http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd .
[C110002]	An SCA implementation MUST reject a componentType or constraining type file that does not conform to http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd .
[C110003]	An SCA implementation MUST reject a contribution file that does not conform to http://docs.oasis-open.org/opencsa/sca/200903/sca-contribution-c-1.1.xsd .
[C110004]	An SCA implementation MUST reject a WSDL file that does not conform to http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200901/sca-wsdl-ext-c-1.1.xsd .
[CA0001]	If SCA 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.
[CA0002]	If multiple annotations apply to a program element, all of the annotations SHOULD be in the same comment block.
[CA0003]	An SCA implementation MUST treat a file with a <code>@WebService</code> annotation specified as if <code>@Interface</code> was specified with the name value of the <code>@WebService</code> annotation used as the name value of the <code>@Interface</code> annotation.
[CA0004]	An SCA implementation MUST treat a function with a <code>@WebFunction</code> annotation specified, unless the <code>exclude</code> value of the <code>@WebFunction</code> annotation is true, as if <code>@Operation</code> was specified with the <code>operationName</code> value of the <code>@WebFunction</code> annotation used as the name value of the <code>@Operation</code> annotation.
[CA0005]	An SCA implementation MUST treat a <code>struct</code> with a <code>@WebOperation</code> annotation specified, unless the <code>exclude</code> value of the <code>@WebOperation</code> annotation is true, as if

	@Operation was specified with the struct as the input value, the operationName value of the @WebOperation annotation used as the name value of the @Operation annotation and the response value of the @WebOperation annotation used as the output values of the @Operation annotation.
[CC0001]	An SCA implementation MUST treat any instance of a @Interface annotation and without an explicit @WebService annotation as if a @WebService annotation with a name value equal to the name value of the @Interface annotation and no other parameters was specified.
[CC0002]	An SCA implementation MUST treat a function annotated with an @Operation annotation and without an explicit @WebFunction annotation as if a @WebFunction annotation with an operationName value equal to the name value of the @Operation annotation and no other parameters was specified.
[CC0003]	An SCA implementation MUST treat an @Operation annotation without an explicit @WebOperation annotation as if a @WebOperation annotation with an operationName value equal to the name value of the @Operation annotation, a response value equal to the output value of the @Operation annotation and no other parameters was specified is applied to the struct identified as the input value of the @Operation annotation.
[CC0004]	A C struct that is listed in a @WebThrows annotation MUST itself have a @WebFault annotation.
[CC0005]	If WSDL mapping annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to WSDL as described.
[CC0006]	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 .
[CC0007]	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 .
[CC0008]	If a @WebService does not have a portName element, an SCA implementation MUST use the value associated with the name element, suffixed with “Port”.
[CC0009]	Only named parameters MAY be referenced by a @WebParam annotation.
[CD0001]	If WSDL extensions are supported by an implementation, all the extensions defined here MUST be supported and MUST be mapped to C as described.
[CD0002]	The @type attribute of a <parameter/> element MUST be a valid C type.
[CD0003]	A <sca-c:bindings/> element MUST NOT have more than one < sca-c:prefix/> child element.
[CD0004]	A <sca-c:bindings/> element MUST NOT have more than one < sca-c:enableWrapperStyle/> child element.
[CD0005]	A <sca-c:bindings/> element MUST NOT have more than one < sca-c:function/> child element.
[CD0006]	A <sca-c:bindings/> element MUST NOT have more than one < sca-c:struct/> child element.
[CD0007]	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.

4007 F.1 JAX-WS Conformance

4008 The JAX-WS 2.1 specification [JAXWS21] defines conformance statements for various requirements
 4009 defined by that specification. The following table outlines those conformance statements, and describes
 4010 whether the conformance statement applies to the WSDL binding described in this specification.

Section	Conformance Statement	Notes	
2	WSDL 1.1 support	[A]	[CF0001]
2	Customization required	[CD0001] The reference to the JAX-WS binding language are treated as a reference to the C WSDL extensions defined in C WSDL Mapping Extensions	
2	Annotations on generated classes		[CF0002]
2.1	WSDL and XML Schema import directives		[CF0003]
2.1.1	Optional WSDL extensions		[CF0004]
2.2	SEI naming	[C100001]	
2.2	javax.jws.WebService required	[B] References to javax.jws.WebService in the conformance statement are treated as the C annotation @WebService.	[CF0005]
2.3	Method naming	[C100002] and [C100005]	
2.3	javax.jws.WebMethod required	[A], [B] References to javax.jws.WebMethod in the conformance statement are treated as the C annotation @WebFunction or @WebOperation.	[CF0006]
2.3	Transmission primitive support		[CF0007]
2.3	Using javax.jws.OneWay	[A], [B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0008]
2.3.1	Using javax.jws.SOAPBinding	[A], [B] References to javax.jws.SOAPBinding in the conformance statement are treated as the C annotation @SOAPBinding.	[CF0009]
2.3.1	Using javax.jws.WebParam	[A], [B] References to javax.jws.WebParam in the conformance statement are treated as the C annotation @WebParam.	[CF0010]
2.3.1	Using javax.jws.WebResult	[A], [B] References to javax.jws.WebResult in the conformance statement are treated as the C annotation @WebResult.	[CF0011]

		annotation @WebResult.	
2.3.1.1	Non-wrapped parameter naming	[C100003]	
2.3.1.2	Default mapping mode		[CF0012]
2.3.1.2	Disabling wrapper style	[B] References to jaxws:enableWrapperStyle in the conformance statement are treated as the C annotation sca-c:enableWrapperStyle.	[CF0013]
2.3.1.2	Wrapped parameter naming	[C100004]	
2.3.1.2	Parameter name clash	[A]	[CF0014]
2.5	javax.xml.ws.WebFault required	[B] References to javax.jws.WebFault in the conformance statement are treated as the C annotation @WebFault.	[CF0015]
2.5	Exception naming	[C100006]	
2.5	Fault equivalence	[A] References to fault exception classes are treated as references to fault message structs.	[CF0016]
2.6	Required WSDL extensions	MIME Binding not necessary	[CF0018]
2.6.1	Unbound message parts	[A]	[CF0019]
2.6.2.1	Duplicate headers in binding		[CF0020]
2.6.2.1	Duplicate headers in message		[CF0021]
3	WSDL 1.1 support	[A]	[CF0022]
3	Standard annotations	[A] [CC0005]	
3.1	Java identifier mapping	[A]	[CF0023]
3.2	WSDL and XML Schema import directives		[CF0024]
3.4	portType naming	[C100008]	
3.5	Operation naming	[C100009] and [C100010]	
3.5.1	One-way mapping	[B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0025]
3.5.1	One-way mapping errors		[CF0026]
3.6.1	Parameter classification	[C100017]	

3.6.1	Parameter naming	[C100011] and [C100014]	
3.6.1	Result naming	[C100012]	
3.6.1	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.	[CF0027]
3.7	Exception naming	[CC0004]	
3.8	Binding selection	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CF0029]
3.10	SOAP binding support	[A]	[CF0030]
3.10.1	SOAP binding style required		[CF0031]
3.11	Port selection		[CF0032]
3.11	Port binding	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CF0033]

4011 [A] All references to Java in the conformance statement are treated as C.

4012 [B] Annotation generation is only necessary if annotations are supported by an SCA implementation.

4013 F.1.1 Ignored Conformance Statements

Section	Conformance Statement	Notes
2.1	Definitions mapping	
2.2	javax.xml.bind.XmlSeeAlso required	
2.3.1	use of JAXB annotations	
2.3.1.2	Using javax.xml.ws.RequestWrapper	
2.3.1.2	Using javax.xml.ws.ResponseWrapper	
2.3.3	Use of Holder	
2.3.4	Asynchronous mapping required	
2.3.4	Asynchronous mapping option	
2.3.4.2	Asynchronous method naming	
2.3.4.2	Asynchronous parameter naming	
2.3.4.2	Failed method invocation	
2.3.4.4	Response bean naming	
2.3.4.5	Asynchronous fault reporting	

2.3.4.5	Asynchronous fault cause	
2.4	JAXB class mapping	
2.4	JAXB customization use	
2.4	JAXB customization clash	
2.4.1	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	
3.1.1	Method name disambiguation	
3.2	Package name mapping	
3.3	Class mapping	
3.4.1	Inheritance flattening	
3.4.1	Inherited interface 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	

4014 **G Migration**

4015 To aid migration of an implementation or clients using an implementation based the version of the Service
4016 Component Architecture for C defined in [SCA C Client and Implementation V1.00](#), this appendix identifies
4017 the relevant changes to APIs, annotations, or behavior defined in V1.00.

4018 **G.1 Implementation.c attributes**

4019 *@location* has been replaced with *@path*.

4020 **G.2 SCALocate and SCALocateMultiple**

4021 `SCALocate()` and `SCALocateMultiple()` have been renamed to `SCAGetReference()`
4022 `SCAGetReferences()` respectively.

4023

H Acknowledgements

4024 The following individuals have participated in the creation of this specification and are gratefully
4025 acknowledged:

4026 **Participants:**

4027

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

4028 | Revision History

4029 [optional; should not be included in OASIS Standards]

4030

Revision	Date	Editor	Changes Made
----------	------	--------	--------------

•

4031