



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

Committee Draft 06

14 October 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd06.html>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd06.doc>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd06.pdf> (Authoritative)

Previous Version:

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

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

Technical Committee:

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

Chair:

Bryan Aupperle, IBM

Editors:

Bryan Aupperle, IBM
David Haney
Pete Robbins, IBM

Related work:

This specification replaces or supercedes:

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

This specification is related to:

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

Downloadable API Documentation:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-apidoc-1.1-cd06.zip>

Hosted API Documentation:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/c/apidoc/index.html>

Declared XML Namespace(s):

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

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

C Artifacts:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-apis-cd06.zip>

Abstract:

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

The SCA C implementation model describes how to implement SCA components in C. A component implementation itself can also be a client to other services provided by other components or external services. The document describes how a 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 – 2010. All Rights Reserved.

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

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

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

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

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

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

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

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

Table of Contents

1	Introduction	8
1.1	Terminology	8
1.2	Normative References	8
1.3	Non-Normative References	9
1.4	Conventions	9
1.4.1	Naming Conventions	9
1.4.2	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 Lifecycles	12
2.3	Implementing a Configuration Property	13
2.4	Component Type and Component	13
2.4.1	Interface.c	14
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	18
3	Basic Client Model	20
3.1	Accessing Services from Component Implementations	20
3.2	Accessing Services from non-SCA Component Implementations	21
3.3	Calling Service Operations	21
3.3.1	Proxy Functions	21
3.4	Long Running Request-Response Operations	22
3.4.1	Asynchronous Invocation	22
3.4.2	Polling Invocation	24
3.4.3	Synchronous Invocation	25
4	Asynchronous Programming	26
4.1	Non-blocking Calls	26
4.2	Callbacks	26
4.2.1	Using Callbacks	27
4.2.2	Callback Instance Management	28
4.2.3	Implementing Multiple Bidirectional Interfaces	28
5	Error Handling	29
6	C API	30
6.1	SCA Programming Interface	30
6.1.1	SCAGetReference	33
6.1.2	SCAGetReferences	33
6.1.3	SCAInvoke	33
6.1.4	SCAProperty<T>	34
6.1.5	SCAGetReplyMessage	36

6.1.6 SCAGetFaultMessage.....	36
6.1.7 SCASetFaultMessage	37
6.1.8 SCASelf.....	38
6.1.9 SCAGetCallback	38
6.1.10 SCAResponse.....	38
6.1.11 SCAIInvokeAsync.....	39
6.1.12 SCAIInvokePoll	39
6.1.13 SCACheckResponse.....	40
6.1.14 SCACancellInvoke	40
6.1.15 SCAEntryPoint	41
6.2 Program-Based Implementation Support	41
6.2.1 SCAService	41
6.2.2 SCAOperation	42
6.2.3 SCAMessageIn	42
6.2.4 SCAMessageOut.....	42
7 C Contributions.....	44
7.1 Executable files.....	44
7.1.1 Executable in contribution	44
7.1.2 Executable shared with other contribution(s) (Export)	44
7.1.3 Executable outside of contribution (Import).....	45
7.2 componentType files.....	46
7.3 C Contribution Extensions	46
7.3.1 Export.c	46
7.3.2 Import.c.....	47
8 C Interfaces	48
8.1 Types Supported in Service Interfaces.....	48
8.1.1 Local Service.....	48
8.1.2 Remotable Service	48
8.2 Restrictions on C header files.....	48
9 WSDL to C and C to WSDL Mapping.....	49
9.1 Interpretations for WSDL to C Mapping.....	49
9.1.1 Definitions.....	49
9.1.2 PortType	49
9.1.3 Operations.....	50
9.1.4 Types.....	50
9.1.5 Fault.....	50
9.1.6 Service and Port.....	51
9.1.7 XML Names.....	51
9.2 Interpretations for C to WSDL Mapping.....	51
9.2.1 Package.....	51
9.2.2 Class.....	51
9.2.3 Interface.....	51
9.2.4 Method.....	52
9.2.5 Method Parameters and Return Type.....	52
9.2.6 Service Specific Exception	52

9.2.7 Generics	53
9.3 Data Binding	53
9.3.1 Simple Content Binding	53
9.3.2 Complex Content Binding	58
10 Conformance	62
10.1 Conformance Targets	62
10.2 SCA Implementations	62
10.3 SCA Documents	63
10.4 C Files	63
10.5 WSDL Files	63
A C SCA Annotations	64
A.1 Application of Annotations to C Program Elements	64
A.2 Interface Header Annotations	64
A.2.1 @Interface	64
A.2.2 @Function	65
A.2.3 @Operation	66
A.2.4 @Remotable	67
A.2.5 @Callback	67
A.2.6 @OneWay	67
A.3 Implementation Annotations	68
A.3.1 @ComponentType	68
A.3.2 @Service	68
A.3.3 @Reference	69
A.3.4 @Property	70
A.3.5 @Init	70
A.3.6 @Destroy	71
A.3.7 @EagerInit	71
A.3.8 @AllowsPassByReference	72
A.4 Base Annotation Grammar	72
B C SCA Policy Annotations	74
B.1 General Intent Annotations	74
B.2 Specific Intent Annotations	75
B.2.1 Security Interaction	76
B.2.2 Security Implementation	76
B.2.3 Reliable Messaging	76
B.2.4 Transactions	77
B.2.5 Miscellaneous	77
B.3 Policy Set Annotations	77
B.4 Policy Annotation Grammar Additions	78
B.5 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	101
D.8 sca-wsdlext-c-1.1.xsd.....	101
E XML Schemas	103
E.1 sca-interface-c-1.1.xsd.....	103
E.2 sca-implementation-c-1.1.xsd	103
E.3 sca-contribution-c-1.1.xsd	104
F Normative Statement Summary	106
F.1 Program-Based Normative Statements Summary	109
F.2 Annotation Normative Statement Summary.....	109
F.3 WSDL Extension Normative Statement Summary.....	110
F.4 JAX-WS Normative Statements	111
F.4.1 Ignored Normative Statements	114
G Migration.....	116
G.1 Implementation.c attributes.....	116
G.2 SCALocate and SCALocateMultiple	116
H Acknowledgements	117
I Revision History.....	118

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.

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

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

18 1.2 Normative References

- 19 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF
20 RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>
- 21 [ASSEMBLY] OASIS Committee Draft 06, *Service Component Architecture Assembly Model Specification Version 1.1*, August 2010. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd06.pdf>
- 22 [POLICY] OASIS Committee Draft 04, *SCA Policy Framework Version 1.1*, September 2010. <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd04.pdf>
- 23 [SDO21] OSOA, *Service Data Objects For C Specification*, September 2007. http://www.osoa.org/download/attachments/36/SDO_Specification_C_V2.1.pdf
- 24 [WSDL11] World Wide Web Consortium, *Web Service Description Language (WSDL)*, March 2001. <http://www.w3.org/TR/wsdl>
- 25 [XSD] World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*, October 2004. <http://www.w3.org/TR/xmlschema-2/>
- 26 [JAXWS21] Doug Kohlert and Arun Gupta, *The Java API for XML-Based Web Services (JAX-WS) 2.1*, JSR, JCP, May 2007. <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>

36 **1.3 Non-Normative References**

37 N/A

38 **1.4 Conventions**

39 **1.4.1 Naming Conventions**

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

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

51 **1.4.2 Typographic Conventions**

52 This specification follows typographic conventions for specific constructs:

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

58 2 Basic Component Implementation Model

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

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

64 2.1 Implementing a Service

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

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

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

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

74 Snippet 2-1 and Snippet 2-2 show a C service interface and the C functions of a C implementation.

75

```
76    /* LoanService interface */  
77    char approveLoan(long customerNumber, long loanAmount);
```

78 *Snippet 2-1: A C Service Interface*

79

```
80    #include "LoanService.h"  
81  
82    char approveLoan(long customerNumber, long loanAmount)  
83    {  
84     ...  
85    }
```

86 *Snippet 2-2: C Service Implementation*

87

88 Snippet 2-3 shows the component type for this component implementation.

89

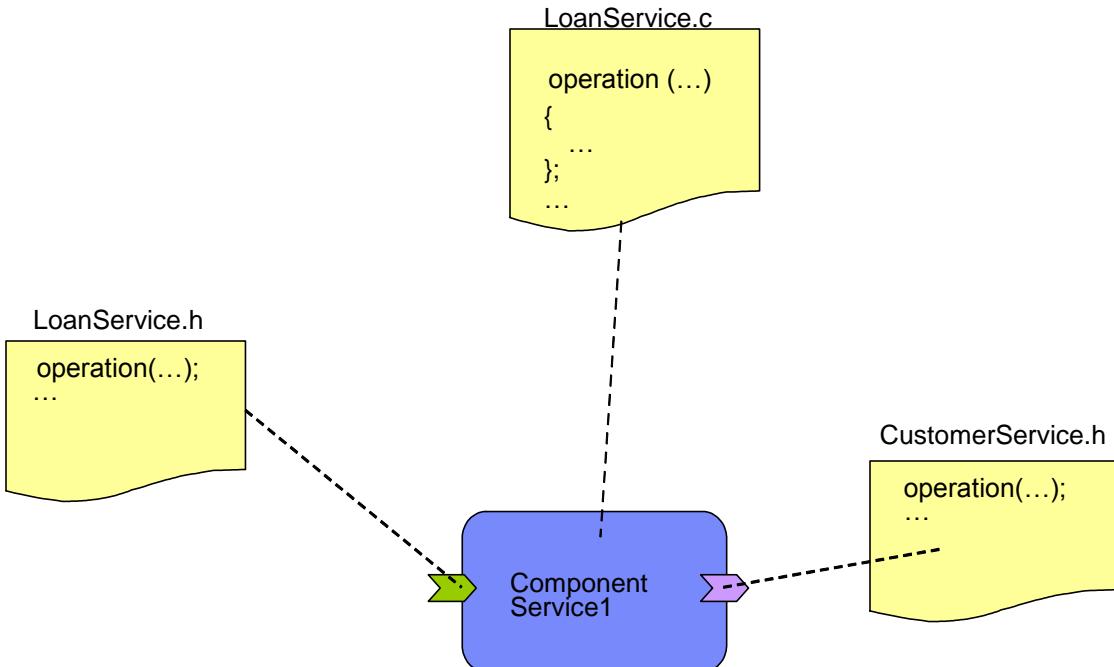
```
90    <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
91     <service name="LoanService">  
92       <interface.c header="LoanService.h"/>  
93     </service>  
94 </componentType>
```

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

96

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

99



100

101 *Figure 2-1: Relationship of C Implementation Artifacts*102

2.1.1 Implementing a Remotable Service

103 A `@remotable="true"` attribute on an `interface.c` element indicates that the interface is **remotable** as
 104 described in the Assembly Specification **[ASSEMBLY]**. Snippet 2-4 shows the component type for a
 105 remotable service:

106

```

107 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
108   <service name="LoanService">
109     <interface.c header="LoanService.h" remotable="true"/>
110   </service>
111 </componentType>

```

112 *Snippet 2-4: ComponentType for a Remotable Service*113

2.1.2 AllowsPassByReference

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

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

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

132 **2.1.2.1 Marking services and references as “allows pass by reference”**

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

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

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

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

147 **2.1.2.2 Using “allows pass by reference” to optimize remotable calls**

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

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

155 **2.1.3 Implementing a Local Service**

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

157 **2.2 Component and Implementation Lifecycles**

158 Component implementations have to manage their own state. A library can be loaded as early as when
159 any component implemented by the library enters the running state [ASSEMBLY] but no later than the
160 first function invocation of a service provided by a component implemented by the library. Component
161 implementations can not make any assumptions about when a library might be unloaded. An SCA
162 runtime MUST NOT perform any synchronization of access to component implementations. [C20015]

163 Component implementations can also specify **lifecycle functions** which are called when a component
164 using the implementation enters the running state or the component leaves running state. An
165 implementation is either initialized eagerly when the component enters the running state (specified by
166 @eagerInit="true"), or lazily when the first client request is received. Lazy instantiation is the default. The
167 C implementation uses the @init="true" attribute of an implementation function element to denote the
168 function to be called upon initialization and the @destroy="true" attribute for the function to be called
169 when exiting the running state. A C implementation MUST only designate functions with no arguments
170 and a void return type as lifecycle functions. [C20004] If an implementation is used by components that
171 are not in a domain-level composite [ASSEMBLY], it is possible for a lifecycle function to be called
172 multiple times.

173 2.3 Implementing a Configuration Property

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

177 Snippet 2-5 shows how to get a property value.

178

```
179     #include "SCA.h"
180
181     void clientFunction()
182     {
183
184         ...
185
186         int32_t loanRating;
187         int values, compCode, reason;
188
189         ...
190
191         SCAPropertyInt(L"maxLoanValue", &loanRating, &values, &compCode, &reason);
192
193         ...
194
195     }
```

196 *Snippet 2-5: Retrieving a Property Value*

197

198 If the property is many valued, an array of the appropriate type is used as the second parameter. The
199 SCA runtime populates the elements of the array with the configured values, using a stride based on `<T>`
200 and a size parameter value for strings and binary data (see `SCAProperty<T>()`) or the size of `struct`
201 resulting from the default mapping in the case of complexTypes (see Complex Content Binding). On
202 input, the `num_values` parameter indicates the number of configured values the client has memory to
203 receive. On output, this parameter will indicate the actual number of configured values available. If this
204 number exceeds the input value, only the input value will be returned and `compCode` and `reason` will be
205 set to indicate that additional values exist.

206 If `<T>` is Bytes, Chars, CChars, String or CString and the property is many valued, the size
207 parameter is also an array. On input only the first value of the array is relevant – indicating the width of
208 each member of the value array. On return, for each returned configured value, the value of the size
209 array is the number of bytes of characters in the corresponding configured value. If this number exceeds
210 the input value, the configured value is truncated and `compCode` and `reason` will be set to indicate the
211 data truncation.

212 2.4 Component Type and Component

213 For a C component implementation, a component type is specified in a side file. By default, the
214 `componentType` side file is in the root directory of the composite containing the component or some
215 subdirectory of the composite root directory with a name specified on the `@componentType` attribute. The
216 location can be modified as described in Implementation.c.

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

219 Snippet 2-6 and Snippet 2-7 show a C service interface and a C implementation of a service.

220

```
221     /* LoanService interface */
222     char approveLoan(long customerNumber, long loanAmount);
```

223 *Snippet 2-6: A C Service Interface*

```
224  
225  
226     #include "LoanService.h"  
227  
228     char approveLoan(long customerNumber, long loanAmount)  
229     {  
230         ...  
231     }
```

232 *Snippet 2-7: C Service Implementation*

233

234 Snippet 2-8 shows the component type for this component implementation.

235

```
236     <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
237         <service name="LoanService">  
238             <interface.c header="LoanService.h" />  
239         </service>  
240     </componentType>
```

241 *Snippet 2-8: Component Type for Service Implementation in Snippet 2-7*

242

243 Snippet 2-9 shows the component using the implementation.

244

```
245     <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
246         name="LoanComposite" >  
247  
248         ...  
249  
250         <component name="LoanService">  
251             <implementation.c module="loan" componentType="LoanService" />  
252         </component>  
253  
254         ...  
255  
256     </composite>
```

257 *Snippet 2-9: Component Using Implementation in Snippet 2-7*

258 **2.4.1 Interface.c**

259 Snippet 2-10 shows the pseudo-schema for the C interface element used to type services and references
260 of component types.

261

```
262     <!-- interface.c schema snippet -->  
263     <interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
264         header="string" remotable="boolean"? callbackHeader="string"?  
265         requires="listOfQNames"? policySets="listOfQNames"? >  
266  
267         <function ... />*  
268         <callbackFunction ... />*  
269         <requires/>*  
270         <policySetAttachment/>*  
271  
272     </interface.c>
```

273 *Snippet 2-10: Pseudo-schema for C Interface Element*

274

275 The **interface.c** element has the **attributes**:

- 276 • ***header : string (1..1)*** – full name of the header file, including either a full path, or its equivalent, or a
 277 relative path from the composite root. This header file describes the interface.
- 278 • ***callbackHeader : string (0..1)*** – full name of the header file that describes the callback interface,
 279 including either a full path, or its equivalent, or a relative path from the composite root.
- 280 • ***remotable : boolean (0..1)*** – indicates whether the service is remotable or local. The default is local.
 281 See Implementing a Remotable Service
- 282 • ***requires : listOfQNames (0..1)*** – a list of policy intents. See the Policy Framework specification
 283 **[POLICY]** for a description of this attribute. If intents are specified at both the interface and function
 284 level, the effective intents for the function is determined by merging the combined intents from the
 285 function with the combined intents for the interface according to the Policy Framework rules for
 286 merging intents within a structural hierarchy, with the function at the lower level and the interface at
 287 the higher level.
- 288 • ***policySets : listOfQNames (0..1)*** – a list of policy sets. See the Policy Framework specification
 289 **[POLICY]** for a description of this attribute.
- 290 The *interface.c* element has the ***child elements***:
- 291 • ***function : CFunction (0..n)*** – see Function and CallbackFunction
- 292 • ***callbackFunction : CFunction (0..n)*** – see Function and CallbackFunction
- 293 • ***requires : requires (0..n)*** - See the Policy Framework specification **[POLICY]** for a description of this
 294 element.
- 295 • ***policySetAttachment : policySetAttachment (0..n)*** - See the Policy Framework specification
 296 **[POLICY]** for a description of this element.

297 2.4.2 Function and CallbackFunction

298 A function of an interface might have behavioral characteristics that need to be identified. This is done
 299 using a *function* or *callbackFunction* child element of *interface.c*. These child elements are also used
 300 when not all functions in a header file are part of the interface or when the interface is implemented by a
 301 program.

- 302 • If the header file identified by the *@header* attribute of an *<interface.c/>* element contains function or
 303 struct declarations that are not operations of the interface, then the functions or structs that are not
 304 operations of the interface MUST be excluded using *<function/>* child elements of the *<interface.c/>*
 305 element with *@exclude="true"*. **[C20006]**
- 306 • If the header file identified by the *@callbackHeader* attribute of an *<interface.c/>* element contains
 307 function or struct declarations that are not operations of the callback interface, then the functions or
 308 structs that are not operations of the callback interface MUST be excluded using *<callbackFunction/>*
 309 child elements of the *<interface.c/>* element with *@exclude="true"*. **[C20007]**

310 Snippet 2-11 shows the *interface.c* pseudo-schema with the pseudo-schema for the *function* and
 311 *callbackFunction* child elements:

```

312
313 <!-- interface.c schema snippet -->
314 <interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"... >
315
316     <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
317         oneWay="Boolean"? exclude="Boolean"?
318         input="NCName"? output="NCName"? >
319         <requires/>*
320         <policySetAttachment/>*
321     </function> *
322
323     <callbackFunction name="NCName" requires="listOfQNames"?
324         policySets="listOfQNames"? oneWay="Boolean"? exclude="Boolean"?
325         input="NCName"? output="NCName"? >
326         <requires/>*
```

```
327     <policySetAttachment/>*
328     </callbackFunction> *
329
330 </interface.c>
```

331 Snippet 2-11: Pseudo-schema for Interface Function and CallbackFunction Sub-elements

332

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

- 334 • **name : NCName (1..1)** – name of the operation being decorated. If the operation is implemented as a
335 function, this is the function name. The @name attribute of a <function> child element of a
336 <interface.c> MUST be unique amongst the <function> elements of that <interface.c>. [C20009]
337 The @name attribute of a <callbackFunction> child element of a <interface.c> MUST be unique
338 amongst the <callbackFunction> elements of that <interface.c>. [C20010]
- 339 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification
340 [POLICY] for a description of this attribute.
- 341 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification
342 [POLICY] for a description of this attribute.
- 343 • **oneWay : boolean (0..1)** – see Non-blocking Calls
- 344 • **exclude : boolean (0..1)** – if true, the function or message struct is excluded from the interface. The
345 default is false.
- 346 • **input : NCNAME (0..1)** – name of the request message struct if it not the same as the operation
347 name. (See Implementing a Service with a Program)
- 348 • **output : NCNAME (0..1)** – name of the response message struct if it not the same as the operation
349 name “Response” appended.

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

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

355 2.4.3 Implementation.c

356 Snippet 2-12 shows the pseudo-schema for the C implementation element used to define the
357 implementation of a component.

358

```
359 <!-- implementation.c schema snippet -->
360 <implementation.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
361   module="NCName" library="boolean"? path="string"?
362   componentType="string" allowsPassByReference="Boolean"?
363   eagerInit="Boolean"? init="Boolean"? destroy="Boolean"?
364   requires="listOfQNames"? policySets="listOfQNames"? >
365
366   <function ... />*
367   <requires/>*
368   <policySetAttachment/>*
369
370 </implementation.c>
```

371 Snippet 2-12: Pseudo-schema for C Implementation Element

372

373 The **implementation.c** element has the **attributes**:

- 374 • ***module* : NCName (1..1)** – name of the binary executable for the service component. This is the root
 375 name of the module.
- 376 • ***library* : boolean (0..1)** – indicates whether the service is implemented as a library or a program. The
 377 default is library. See Implementing a Service with a Program
- 378 • ***path* : string (0..1)** – path to the module which is either relative to the root of the contribution
 379 containing the composite or is prefixed with a contribution import name and is relative to the root of
 380 the import. See C Contributions.
- 381 • ***componentType* : string (1..1)** – name of the componentType file. A “*.componentType*” extention
 382 will be appended. A path to the componentType file which is relative to the root of the contribution
 383 containing the composite or is prefixed with a contribution import name and is relative to the root of
 384 the import (see C Contributions) can be included.
- 385 • ***allowsPassByReference* : boolean (0..1)** – indicates the implementation allows pass by reference
 386 data exchange semantics on calls to it or from it. These semantics apply to all services provided by
 387 and references used by an implementation. See AllowsPassByReference
- 388 • ***eagerInit* : boolean (0..1)** – indicates a composite scoped implementation is to be initialized when it
 389 is loaded. See Component and Implementation Lifecycles
- 390 • ***init* : boolean (0..1)** – indicates program is to be called with an initialize flag to initialize the
 391 implementation. See Component and Implementation Lifecycles
- 392 • ***destroy* : boolean (0..1)** – indicates is to be called with a destroy flag to cleanup the
 393 implementation. See Component and Implementation Lifecycles
- 394 • ***requires* : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification
 395 [POLICY] for a description of this attribute. If intents are specified at both the implementation and
 396 function level, the effective intents for the function is determined by merging the combined intents
 397 from the function with the combined intents for the implementation according to the Policy Framework
 398 rules for merging intents within a structural hierarchy, with the function at the lower level and the
 399 implementation at the higher level.
- 400 • ***policySets* : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification
 401 [POLICY] for a description of this attribute.
- 402 The *interface.c* element has the ***child elements***:
- 403 • ***function* : CImplementationFunction (0..n)** – see Implementation Function
- 404 • ***requires* : requires (0..n)** - See the Policy Framework specification [POLICY] for a description of this
 405 element.
- 406 • ***policySetAttachment* : policySetAttachment (0..n)** - See the Policy Framework specification
 407 [POLICY] for a description of this element.

408 **2.4.4 Implementation Function**

409 A function of an implementation might have operational characteristics that need to be identified. This is
 410 done using a *function* child element of *implementation.c*

411 Snippet 2-13 shows the *implementation.c* pseudo-schema with the pseudo-schema for a *function* child
 412 element:

413

```
414 <!-- ImplementationFunction schema snippet -->
415 <implementation.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"... >
416
417     <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
418         allowsPassByReference="Boolean"? init="Boolean"?
419         destroy="Boolean"? >
420         <requires/>*
421         <policySetAttachment/>*
422     </function> *
```

424 </implementation.c>

425 Snippet 2-13: Pseudo-schema for Implementation Function Sub-element

426

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

- 428 • **name : NCName (1..1)** – name of the function being decorated. The @name attribute of a
429 child element of a <implementation.c/> MUST be unique amongst the <function/>
430 elements of that <implementation.c/>. [C20013]
- 431 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification
432 [POLICY] for a description of this attribute.
- 433 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification
434 [POLICY] for a description of this attribute.
- 435 • **allowsPassByReference : boolean" (0..1)** – indicates the function allows pass by reference data
436 exchange semantics. See AllowsPassByReference
- 437 • **init : boolean (0..1)** – indicates this function is to be called to initialize the implementation. See
438 Component and Implementation Lifecycles
- 439 • **destroy : boolean (0..1)** – indicates this function is to be called to cleanup the implementation. See
440 Component and Implementation Lifecycles

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

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

446 2.5 Implementing a Service with a Program

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

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

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

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

468 Snippet 2-14 a program implementing a service using these support functions.

469

```
470       #include "SCA.h"
471       #include "myInterface.h"
```

```
472 main () {
473     wchar_t myService [255];
474     wchar_t myOperation [255];
475     int compCode, reason;
476     struct FirstInputMsg myFirstIn;
477     struct FirstOutputMsg myFirstOut;
478
479
480     SCAService(myService, &compCode, &reason);
481
482     SCAOperation(myOperation, &compCode, &reason);
483
484     if (wcscmp(myOperation,L"myFirstOperation")==0) {
485         SCAMessageIn(myService, myOperation,
486                     sizeof(struct FirstInputMsg), (void *)&myFirstIn,
487                     &compCode, &reason);
488         ...
489         SCAMessageOut(myService, myOperation,
490                     sizeof(struct FirstOutputMsg),(void *)&myFirstOut,
491                     &compCode, &reason);
492     }
493     else
494     {
495         ...
496     }
497 }
```

498 *Snippet 2-14: C Service Implementation in a Program*

499 3 Basic Client Model

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

502 3.1 Accessing Services from Component Implementations

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

504 Snippet 3-1 the SCAGetReference() function used for this.

505

```
506    void SCAGetReference(wchar_t *referenceName, SCAREF *referenceToken,  
507                                 int *compCode, int *reason);  
508    void SCAInvoke(SCAREF referenceToken, wchar_t *operationName,  
509                                 int inputMsgLen, void *inputMsg,  
510                                 int outputMsgLen, void *outputMsg, int *compCode, int *reason);
```

511 *Snippet 3-1: Partial SCA API Definition*

512

513 Snippet 3-2 shows a sample of how a service is called in a C component implementation.

514

```
515    #include "SCA.h"  
516  
517    void clientFunction()  
{  
518  
519       SCAREF serviceToken;  
520       int compCode, reason;  
521       long custNum = 1234;  
522       short rating;  
523  
524       ...  
525       SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);  
526       SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),  
527                                 (void *)&custNum, sizeof(rating), (void *)&rating,  
528                                 &compCode, &reason);  
529  
530     }  
531 }
```

532 *Snippet 3-2: Using SCAGetReference*

533

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

536

```
537    SCAREF *tokens;  
538    int num_targets;  
539    ...  
540    myFunction(...){  
541       int compCode, reason;  
542       ...  
543       SCAGetReferences(L"myReference", &tokens, &num_targets, &compCode,  
544                                 &reason);  
545       for (i = 0; i < num_targets; i++)  
546       {  
547           SCAInvoke(tokens[i], L"myOperation", sizeof(inputMsg),  
548                                 (void *)&inputMsg, 0, NULL, &compCode, &reason);
```

```
549     };  
550 };
```

551 *Snippet 3-3: Using SCAGetReferences*

552

553 **3.2 Accessing Services from non-SCA Component Implementations**

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

556 Snippet 3-4 shows a sample of how a service is called in non-SCA C code.

557

```
558 #include "SCA.h"  
559  
560 void externalFunction()  
561 {  
562     SCAREF serviceToken;  
563     int compCode, reason;  
564     long custNum = 1234;  
565     short rating;  
566  
567     SCAEntryPoint(L"customerService", L"http://example.com/mydomain",  
568                 &serviceToken, &compCode, &reason);  
569     SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),  
570               (void *)&custNum, sizeof(rating), (void *)&rating,  
571               &compCode, &reason);  
572 }
```

573 *Snippet 3-4: Using SCAEntryPoint*

574

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

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

- componentName/serviceName/bindingName

579 **3.3 Calling Service Operations**

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

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

585 **3.3.1 Proxy Functions**

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

591 For SCA, proxy functions have the form:

```
592 <functionReturn> SCA_<functionName>(<SCAREF referenceToken,>  
593                                     <functionParameters> )
```

594 where:

- <functionName> is the name of interface function

- 596 • <functionParameters> are the parameters of the interface function
 597 • <functionReturn> is the return type of the interface function

598 Snippet : Proxy Function Format

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

- 600 • `ENOENT` if a remote service is unavailable
 601 • `EFAULT` if a fault is returned by the operation

602 Snippet 3-5 shows a sample of using a proxy function.

603

```
#include "SCA.h"

void clientFunction()
{
    SCAREF serviceToken;
    int compCode, reason;
    long custNum = 1234;
    short rating;

    ...
    SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
    errno = 0;
    rating = SCA_getCreditRating(serviceToken, custNum);
    if (errno) {
        /* handle error or fault */
    }
    else {
        ...
    }
}
```

626 *Snippet 3-5: Using a Proxy Function*

627

628 An SCA implementation MAY support proxy functions. [C30001]

629 3.4 Long Running Request-Response Operations

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

639 3.4.1 Asynchronous Invocation

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

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

645 *Snippet 3-6: Asynchronous Handler Function Format*

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

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

652 If proxy functions are supported, for a service operation with signature

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

654 the asynchronous invocation style includes a proxy function

```
655 void SCA_<function name>Async(SCAREF, <in_parameters>, void (*)(short));
```

656 *Snippet 3-7: Asynchronous Proxy Function Format*

657 which will set `errno` to `EBUSY` if one request is outstanding and another is attempted.

658 Snippet 3-8 shows a sample of how the asynchronous invocation style is used in a C component
659 implementation.

660

```
661 #include "SCA.h"
662 #include "TravelService.h"
663
664 SCAREF serviceToken;
665 int compCode, reason;
666
667 void makeReservationsHandler(short rspType)
668 {
669     struct confirmationData cd;
670     wchar_t *fault, *faultDetails;
671
672     if (rspType == SCA_REPLY_MESSAGE) {
673         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
674         ...
675     }
676     else {
677         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
678                             &faultDetails, &compCode, &reason);
679         if (wcscmp(*fault, L"noFlight") {
680             ...
681         }
682         else {
683             ...
684         }
685     }
686
687     return;
688 }
689
690 void clientFunction()
691 {
692
693     struct itineraryData id;
694
695     ...
696
697     void (*ah)(short) = &makeReservationsHandler;
698
699     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
700
701     SCAIInvokeAsync(serviceToken, L"makeReservations", sizeof(itineraryData),
702                     (void *)&id, ah, &compCode, &reason);
```

```
703     return;
704 }
705 }
```

706 *Snippet 3-8: Using Asynchronous Invocation*

707 3.4.2 Polling Invocation

708 The polling style of invocation uses `SCAIInvokePoll()` which has the same signature as `SCAIInvoke()`
709 but without the `outputMsgLen` or `outputMsg` parameters. This API sends the operation request. After
710 the request is sent the client can check to see if a response has been received by using
711 `SCACheckResponse()` or cancel the request with `SCACancelInvoke()`.

712 If proxy functions are supported, for a service operation with signature

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

714 the polling invocation style includes a proxy function

```
void SCA_<function name>Poll(SCAREF, <in_parameters>);
```

716 *Snippet 3-9: Asynchronous Pooling Proxy Function Format*

717 which will set `errno` to `EBUSY` if one request is outstanding and another is attempted.

718 Snippet 3-10 shows a sample of how the polling invocation style is used in a C component
719 implementation.

```
720
721 #include "SCA.h"
722 #include "TravelService.h"
723
724 void pollingClientFunction()
725 {
726     SCAREF serviceToken;
727     int compCode, reason;
728     short rspType;
729
730     struct itineraryData id;
731     struct confirmationData cd;
732     wchar_t *fault, *faultDetails;
733
734     ...
735
736     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
737
738     SCAInvokePoll(serviceToken, L"makeReservations", sizeof(itineraryData),
739                   (void *) &id), &compCode, &reason);
740
741     SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
742     while (!rspType) {
743         // do something, then wait for some time...
744         SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
745     }
746     if (rspType == SCA_REPLY_MESSAGE) {
747         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
748         ...
749     }
750     else {
751         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
752                             &faultDetails, &compCode, &reason);
753         if (wcscmp(*fault, L"noFlight") {
754             ...
755         }
756     }
757 }
```

```
758     }
759 }
760     return;
761 }
```

763 *Snippet 3-10: Using Asynchronous Polling Invocation*

764 **3.4.3 Synchronous Invocation**

765 In this style the client uses API `SCAIInvoke()` but the implementation of this API suspends the main
766 thread after the request is made, and in an implementation-dependent manner receives the response,
767 resumes the main thread and returns from the member function call. If proxy functions are supported, the
768 client can call `SCA_<function name>()` as normal, and again the implementation handles the
769 asynchronous aspects.

770 Snippet 3-11 shows a sample of how the synchronous invocation style is used in a C component
771 implementation.

```
772
773 #include "SCA.h"
774 #include "TravelService.h"
775
776 void synchronousClientFunction()
777 {
778     SCAREF serviceToken;
779     int compCode, reason;
780
781     struct itineraryData id;
782     struct confirmationData *cd;
783     wchar_t *fault, *faultDetails;
784
785     ...
786
787     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
788
789     SCAInvoke(serviceToken, L"makeReservations", sizeof(itineraryData),
790               (void *)&id, sizeof(confirmationData), (void *)&cd,
791               &compCode, &reason);
792     if (compCode == SCA_FAULT) {
793         ...
794     }
795     else {
796         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
797                             &faultDetails, &compCode, &reason);
798         if (wcscmp(*fault, L"noFlight") {
799             ...
800         }
801         else {
802             ...
803         }
804     }
805
806     return;
807 }
```

808 *Snippet 3-11: Using Synchronous Invocation for an Asynchronous Operation*

809 4 Asynchronous Programming

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

817 4.1 Non-blocking Calls

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

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

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

```
827
828 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
829   <service name="LoanService">
830     <interface.c header="LoanService.h">
831       <function name="reportEvent" oneWay="true" />
832     </interface.c>
833   </service>
834 </componentType>
```

835 *Snippet 4-1: ComponentType with oneWay Function*

836

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

840 4.2 Callbacks

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

843 A callback interface is declared by the `@callbackHeader` and `@callbackFunctions` attributes in the
844 interface definition of the service. Snippet 4-2 shows the component type for a service *MyService* with the
845 interface defined in *MyService.h* and the interface for callbacks defined in *MyServiceCallback.h*,

```
846
847 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" >
848   <service name="MyService">
849     <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h"/>
850   </service>
851 </componentType>
```

852 *Snippet 4-2: ComponentType with a Callback Interface*

853 **4.2.1 Using Callbacks**

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

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

863

```
864     double requestQuotation(char *productCode,int quantity);  
865  
866     char *getState();  
867     char *getZipCode();  
868     char *getCreditRating();
```

869 *Snippet 4-3: C Interface with a Callback Interface*

870

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

877 Snippet 4-4 illustrates a possible implementation of the example service.

878

```
879 #include "QuotationCallback.h"  
880 #include "SCA.h"  
881  
882 double requestQuotation(char *productCode,int quantity) {  
883     double price, discount = 0;  
884     char state[3], creditRating[4];  
885     SCAREF callbackRef;  
886     int compCode, reason;  
887  
888     price = getPrice(productQuote, quantity);  
889  
890     SCAGetCallback(L"", &callbackRef, &compCode, &reason);  
891     SCAInvoke(callbackRef, L"getState", 0, NULL, sizeof(state), state,  
892                 &compCode, &reason);  
893     if (quantity > 1000 && strcmp(state,"FL") == 0)  
894         discount = 0.05;  
895     SCAInvoke(callbackRef, L"getCreditRating", 0, NULL, sizeof(creditRating),  
896                 creditRating, &compCode, &reason);  
897     if (quantity > 10000 && creditRating[0] == 'A')  
898         discount += 0.05;  
899     SCAResponseCallback(callbackRef, &compCode, &reason);  
900     return price * (1-discount);  
901 }
```

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

903

904 Snippet 4-5 is taken from the client of this example service. The client's service implementation
905 implements the functions of the `QuotationCallback` interface as well as those of its own service interface
906 `ClientService`.

```

907
908 #include "QuotationCallback.h"
909 #include "SCA.h"
910
911 char state[3] = "TX", zipCode[6] = "78746", creditRating[3] = "AA";
912
913 ClientFunction() {
914     SCAREF serviceToken;
915     int compCode, reason;
916
917     SCAGetReference(L"quotationService", &serviceToken, &compCode, &reason);
918
919     SCA_requestQuotation(serviceToken, "AB123", 2000);
920 }
921
922 char *getState() {
923     return state;
924 }
925 char *getZipCode() {
926     return zipCode;
927 }
928 char *getCreditRating() {
929     return creditRating;
930 }

```

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

932

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

937 **4.2.2 Callback Instance Management**

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

943 **4.2.3 Implementing Multiple Bidirectional Interfaces**

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

947 5 Error Handling

948 Clients calling service operations will experience business logic errors, and SCA runtime errors.
949 Business logic errors are generated by the implementation of the called service operation. They are
950 handled by client the invoking the operation of the service.
951 SCA runtime errors are generated by the SCA runtime and signal problems in the management of the
952 execution of components, and in the interaction with remote services. The SCA C API includes two return
953 codes on every function, a completion code and a reason code. The reason code is used to provide more
954 detailed information if a function does not complete successfully.

955

```
956 /* Completion Codes */  
957 #define SCACC_OK 0  
958 #define SCACC_WARNING 1  
959 #define SCACC_FAULT 2  
960 #define SCACC_ERROR 3  
961  
962 /* Reason Codes */  
963 #define SCA_SERVICE_UNAVAILABLE 1  
964 #define SCA_MULTIPLE_SERVICES 2  
965 #define SCA_DATA_TRUNCATED 3  
966 #define SCA_PARAMETER_ERROR 4  
967 #define SCA_BUSY 5  
968 #define SCA_RUNTIME_ERROR 6  
969 #define SCA_ADDITIONAL_VALUES 7  
970  
971 /* Response Types */  
972 #define SCA_NO_RESPONSE 0  
973 #define SCA_REPLY_MESSAGE 1  
974 #define SCA_FAULT_MESSAGE 2
```

975 *Snippet 5-1: SCA Constant Definitions*

976

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

979 **6 C API**

980 **6.1 SCA Programming Interface**

981 The SCA API definition is:

982

```
983     typedef void *SCAREF;
984
985     void SCAGetReference(wchar_t *referenceName,
986                           SCAREF *referenceToken,
987                           int *compCode,
988                           int *reason);
989
990     void SCAGetReferences(wchar_t *referenceName,
991                           SCAREF **referenceTokens,
992                           int *num_targets,
993                           int *compCode,
994                           int *Reason);
995
996     void SCAInvoke(SCAREF token,
997                     wchar_t *operationName,
998                     int inputMsgLen,
999                     void *inputMsg,
1000                    int *outputMsgLen,
1001                    void *outputMsg,
1002                    int *compCode,
1003                    int *reason);
1004
1005    void SCAPropertyBoolean(wchar_t *propertyName,
1006                            char *value,
1007                            int *num_values,
1008                            int *compCode,
1009                            int *reason);
1010
1011    void SCAPropertyByte(wchar_t *propertyName,
1012                          int8_t *value,
1013                          int *num_values,
1014                          int *compCode,
1015                          int *reason);
1016
1017    void SCAPropertyBytes(wchar_t *propertyName,
1018                          int8_t **value,
1019                          int *size,
1020                          int *num_values,
1021                          int *compCode,
1022                          int *reason);
1023
1024    void SCAPropertyChar(wchar_t *propertyName,
1025                          wchar_t *value,
1026                          int *num_values,
1027                          int *compCode,
1028                          int *reason);
1029
1030    void SCAPropertyChars(wchar_t *propertyName,
1031                          wchar_t **value,
1032                          int *size,
1033                          int *num_values,
1034                          int *compCode,
1035                          int *reason);
```

```
1036
1037 void SCAPropertyCChar(wchar_t *propertyName,
1038             char *value,
1039             int *num_values,
1040             int *compCode,
1041             int *reason);
1042
1043 void SCAPropertyCChars(wchar_t *propertyName,
1044             char **value,
1045             int *size,
1046             int *num_values,
1047             int *compCode,
1048             int *reason);
1049
1050 void SCAPropertyShort(wchar_t *propertyName,
1051             int16_t *value,
1052             int *num_values,
1053             int *compCode,
1054             int *reason);
1055
1056 void SCAPropertyInt(wchar_t *propertyName,
1057             int32_t *value,
1058             int *num_values,
1059             int *compCode,
1060             int *reason);
1061
1062 void SCAPropertyLong(wchar_t *propertyName,
1063             int64_t *value,
1064             int *num_values,
1065             int *compCode,
1066             int *reason);
1067
1068 void SCAPropertyFloat(wchar_t *propertyName,
1069             float *value,
1070             int *num_values,
1071             int *compCode,
1072             int *reason);
1073
1074 void SCAPropertyDouble(wchar_t *propertyName,
1075             double *value,
1076             int *num_values,
1077             int *compCode,
1078             int *reason);
1079
1080 void SCAPropertyString(wchar_t *propertyName,
1081             wchar_t **value,
1082             int *size,
1083             int *num_values,
1084             int *compCode,
1085             int *reason);
1086
1087 void SCAPropertyCString(wchar_t *propertyName,
1088             char **value,
1089             int *size,
1090             int *num_values,
1091             int *compCode,
1092             int *reason);
1093
1094 void SCAPropertyStruct(wchar_t *propertyName,
1095             void *value,
1096             int *num_values,
1097             int *compCode,
1098             int *reason);
1099
```

```

1100 void SCAGetReplyMessage(SCAREF token,
1101                         int *bufferLen,
1102                         void *buffer,
1103                         int *compCode,
1104                         int *reason);
1105
1106 void SCAGetFaultMessage(SCAREF token,
1107                         int *bufferLen,
1108                         wchar_t **faultName,
1109                         void *buffer,
1110                         int *compCode,
1111                         int *reason);
1112
1113 void SCASetFaultMessage(wchar_t *serviceName,
1114                           wchar_t *operationName,
1115                           wchar_t *faultName,
1116                           int bufferLen,
1117                           void *buffer,
1118                           int *compCode,
1119                           int *reason);
1120
1121 void SCASelf(wchar_t *serviceName,
1122               SCAREF *serviceToken,
1123               int *compCode,
1124               int *reason);
1125
1126 void SCAGetCallback(wchar_t *serviceName,
1127                       SCAREF *serviceToken,
1128                       int *compCode,
1129                       int *reason);
1130
1131 void SCAResponseCallback(SCAREF serviceToken,
1132                           int *compCode,
1133                           int *reason);
1134
1135 void SCAInvokeAsync(SCAREF token,
1136                      wchar_t *operationName,
1137                      int inputMsgLen,
1138                      void *inputMsg,
1139                      void (*handler)(short),
1140                      int *compCode,
1141                      int *reason);
1142
1143 void SCAInvokePoll(SCAREF token,
1144                      wchar_t *operationName,
1145                      int inputMsgLen,
1146                      void *inputMsg,
1147                      int *compCode,
1148                      int *reason);
1149
1150 void SCACheckResponse(SCAREF token,
1151                        short *responseType,
1152                        int *compCode,
1153                        int *reason);
1154
1155 void SCACancelInvoke(SCAREF token,
1156                       int *compCode,
1157                       int *reason);
1158
1159 void SCAEntryPoint(wchar_t *serviceURI,
1160                     wchar_t *domainURI,
1161                     SCAREF *serviceToken,
1162                     int *compCode,
1163                     int *reason);

```

1164 Snippet 6-1: SCA API Definition

6.1.1 SCAGetReference

1166 A C component implementation uses `SCAGetReference()` to initialize a Reference before invoking any
1167 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>SCAIInvoke()</code> calls. This will be NULL if <code>referenceName</code> is not defined for the component.
	compCode	<code>SCACC_OK</code> , if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
	reason	<code>SCA_SERVICE_UNAVAILABLE</code> if no suitable service exists in the domain <code>SCA_MULTIPLE_SERVICES</code> 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.	

1168 Table 6-1: SCAGetReference Details

6.1.2 SCAGetReferences

1170 A C component implementation uses `SCAGetReferences()` to initialize a Reference that might be
1171 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>SCAIInvoke()</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	<code>SCACC_OK</code> , if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
	Reason	<code>SCA_SERVICE_UNAVAILABLE</code> 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.	

1172 Table 6-2: SCAGetReferencse Details

6.1.3 SCAInvoke

1174 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 SCAGetReference() or SCAGetReferences(), SCASelf() or SCAGetCallback() 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	SCACC_OK, if the call is successful SCACC_WARNING, if the response data was truncated. The buffer size needs to be increased and SCAGetReplyMessage() called with the larger buffer. SCACC_FAULT, if the operation returned a business fault. SCAGetFaultMessage() needs to be called to get the fault details. SCACC_ERROR, otherwise – see reason for details
	Reason	SCA_DATA_TRUNCATED if the response data was truncated SCA_PARAMETER_ERROR if the operationName is not defined for the interface SCA_SERVICE_UNAVAILABLE if the provider for the interface is no longer operational
Post Condition	Unless a SCA_SERVICE_UNAVAILABLE reason is returned, the token remains valid for subsequent calls.	

1175 Table 6-3: SCALnvoke Details

1176 **6.1.4 SCAProperty<T>**

1177 A C component implementation uses SCAProperty<T>() to get the configured value for a Property.

1178 This API is available for Boolean, Byte, Bytes, Char, Chars, CChar, CChars, Short, Int, Long, Float, Double, String, CString and Struct. The Char, Chars, and String variants return wchar_t based data while the CChar, CChars, and CString variants return char based data. The Bytes, Chars, and CChars variants return a buffer of data. The String and CString variants return a null terminated string.

1182 An SCA runtime MAY additionally provide a DataObject variant of this API for handling properties with
1183 complex XML types. The type of the value parameter in this variant is DATAOBJECT. [C60002]

1184 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
In/Out Parameter	num_values	Input: Maximum number of configured values that can

		be returned Output: Actual number of configured values
Output Parameters	value	Configured value(s) of the property
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the number of configured values exceeds the input value of num_values. The call needs to be repeated with value pointing to a location sufficient in size to hold all of the configured values. SCACC_ERROR, otherwise – see reason for details
	reason	SCA_ADDITIONAL_VALUES if the number of configured values exceeds the input value of num_values 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.	

1185 *Table 6-4: SCAProperty<T> Details for fixed length types*

1186

1187 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 Parameters	size	<p>Input: Maximum number of bytes or characters that can be returned for each configured value</p> <p>Output: Actual number of bytes or characters returned or size needed to hold a configured value.</p> <p>If the property is many valued, size is an array. On input only the first value of the array is relevant – indicating the width of each member of the value array. On return, for each returned configured value, the corresponding value of size is the number of bytes or characters in the configured value. If this number exceeds the input value, the configured value is truncated and compCode and reason are set to indicate the data truncation.</p>
	num_values	<p>Input: Maximum number of configured values that can be returned.</p> <p>Output: Actual number of configured values</p>
Output Parameters	value	Configured value(s) of the property
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the data was truncated or the number of configured values exceeds the input value of num_values

		SCACC_ERROR, otherwise – see reason for details
	reason	<p>SCA_ADDITIONAL_VALUES if the number of configured values exceeds the input value of num_values. The call needs to be repeated with value pointing to a location sufficient in size to hold all of the configured values.</p> <p>SCA_DATA_TRUNCATED, if the data was truncated. The buffer size for each configured value needs to be increased and the call repeated with the larger buffer. If both the number of configured values exceeds the input value of num_values and some configured values were truncated, SCA_ADDITIONAL_VALUES is returned.</p> <p>SCA_PARAMETER_ERROR if the propertyName is not defined for the component or its type is incompatible with <T></p>
Post Condition	The configured value of the Property is loaded into the appropriate variable.	

1188 *Table 6-5: SCAProperty<T> Details for variable length types*

1189 **6.1.5 SCAGetReplyMessage**

1190 A C component implementation uses `SCAGetReplyMessage()` to retrieve the reply message of an
1191 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 SCACC_WARNING compCode or has a valid serviceToken and an <code>SCACallback()</code> returned a SCACC_WARNING 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	SCA_DATA_TRUNCATED if the fault data was truncated.
Post Condition	The referenceToken remains valid for subsequent calls.	

1192 *Table 6-6: SCAGetReplyMessage Details*

1193 **6.1.6 SCAGetFaultMessage**

1194 A C component implementation uses `SCAGetFaultMessage()` to retrieve the details of a business fault
1195 received in response to an operation invocation.

Precondition	C component instance is running, has a valid token and an <code>SCAIInvoke()</code> returned a <code>SCACC_FAULT compCode</code>	
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	faultName	Name of the business fault
	buffer	Fault message
	compCode	<code>SCACC_OK</code> , if the call is successful <code>SCACC_WARNING</code> , if the fault data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer. <code>SCACC_ERROR</code> , otherwise – see reason for details
	reason	<code>SCA_DATA_TRUNCATED</code> if the fault data was truncated. <code>SCA_PARAMETER_ERROR</code> if the last operation invoked on the Reference did not return a business fault
Post Condition	The <code>referenceToken</code> remains valid for subsequent calls.	

1196 *Table 6-7: SCAGetFaultMessage Details*

1197 **6.1.7 SCASetFaultMessage**

1198 A C component implementation uses `SCASetFaultMessage()` to return a business fault in response to
1199 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	<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 the <code>serviceName</code> is not defined for the component, <code>operationName</code> is not defined for the Service or the <code>faultName</code> is not defined for the operation
Post Condition	No change	

1200 *Table 6-8: SCASetFaultMessage Details*

6.1.8 SCASelf

1202 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>SCAInvoke()</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.	

1203 *Table 6-9: SCASelf Details*

6.1.9 SCAGetCallback

1205 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>SCAInvoke()</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.	

1207 *Table 6-10: SCAGetCallback Details*

6.1.10 SCAResleaseCallback

1209 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 serviceToken is not valid
Post Condition	The token becomes invalid for subsequent calls.	

1211 *Table 6-11: SCAResponseCallback Details*

1212 **6.1.11 SCAInvokeAsync**

1213 A C component implementation uses `SCAInvokeAsync()` to invoke a long running operation of an
1214 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.	

1215 *Table 6-12: SCAInvokeAsynch Details*

1216 **6.1.12 SCAInvokePoll**

1217 A C component implementation uses `SCAInvokePoll()` to invoke a long running operation of a
1218 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
Output Parameters	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	Reason	SCA_BUSY if an operation is already outstanding for this Reference or Callback SCA_PARAMETER_ERROR if the operationName is not defined for the interface SCA_SERVICE_UNAVAILABLE if provider of the interface is no longer operational
Post Condition	Unless a SCA_SERVICE_UNAVAILABLE reason is returned, the token remains valid for subsequent calls.	

1219 *Table 6-13: SCAIInvokePoll Details*1220 **6.1.13 SCACheckResponse**1221 A C component implementation uses SCACheckResponse() to determine if a response to a long
1222 running operation request has been received.

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

1223 *Table 6-14: SCACheckResponse Details*1224 **6.1.14 SCACancelInvoke**

1225 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 SCAIInvokeAsync() or SCAIInvokePoll() but has not received a response.	
Input Parameter	token	Token returned by prior SCALocate(), SCALocateMultiple(), SCASelf() or SCAGetCallback() call.
Output Parameters	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_PARAMETER_ERROR 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.
----------------	---

1226 *Table 6-15 : SCACancelInvokee Details*

1227 **6.1.15 SCAEntryPoint**

1228 Non-SCA C code uses `SCAEntryPoint()` to access a Service before invoking any operations of the
1229 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.	

1230 *Table 6-16: SCAEntryPoint Details*

1231 **6.2 Program-Based Implementation Support**

1232 Support for components implemented via C programs is provided by the functions `SCAService()`,
1233 `SCAOperation()`, `SCAMessageIn()` and `SCAMessageOut()`.

1234

```

1235 void SCAService(wchar_t *serviceName, int *compCode, int *reason);
1236
1237 void SCAOperation(wchar_t *operationName, int *compCode, int *reason);
1238
1239 void SCAMessageIn(wchar_t *serviceName,
1240                   wchar_t *operationName,
1241                   int *bufferLen,
1242                   void *buffer,
1243                   int *compCode,
1244                   int *reason);
1245
1246 void SCAMessageOut(wchar_t *serviceName,
1247                     wchar_t *operationName,
1248                     int bufferLen,
1249                     void *buffer,
1250                     int *CompCode,
1251                     int *Reason);
```

1252 *Snippet 6-2: SCA API for Program Implementations Definition*

1253 **6.2.1 SCAService**

1254 A program-based C component implementation uses `SCAService()` to determine which service was
1255 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	

1256 *Table 6-17: SCAService Details*

6.2.2 SCAOperation

1258 A program-based C component implementation uses `SCAOperation()` to determine which operation of
1259 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.	

1260 *Table 6-18: SCAOperation Details*

6.2.3 SCAMessageIn

1261 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.	

1263 *Table 6-19: SCAaMessgeln Details*

6.2.4 SCAMessageOut

1264 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	<code>SCACC_OK</code>
	reason	
Post Condition	The component normally ends processing.	

1266 *Table 6-20: SCAMessgeOut Details*

1267 7 C Contributions

1268 Contributions are defined in the Assembly specification [ASSEMBLY] C contributions are typically, but
1269 not necessarily contained in .zip files. In addition to SCDL and potentially WSDL artifacts, C contributions
1270 include binary executable files, componentType files and potentially C interface headers. No additional
1271 discussion is needed for header files, but there are additional considerations for executable and
1272 componentType files discussed.

1273 7.1 Executable files

1274 Executable files containing the C implementations for a contribution can be contained in the contribution,
1275 contained in another contribution or external to any contribution. In some cases, it could be desirable to
1276 have contributions share an executable. In other cases, an implementation deployment policy might
1277 dictate that executables are placed in specific directories in a file system.

1278 7.1.1 Executable in contribution

1279 When the executable file containing a C implementation is in the same contribution, the `@path` attribute of
1280 the `implementation.c` element is used to specify the location of the executable. The specific location of an
1281 executable within a contribution is not defined by this specification.

1282 Snippet 7-1 shows a contribution containing a DLL.

```
1283
1284     META-INF/
1285         sca-contribution.xml
1286     bin/
1287         autoinsurance.dll
1288     AutoInsurance/
1289         AutoInsurance.composite
1290         AutoInsuranceService/
1291             AutoInsurance.h
1292             AutoInsurance.componentType
1293         include/
1294             Customers.h
1295             Underwriting.h
1296             RateUtils.h
```

1297 *Snippet 7-1: Contribution Containing a DLL*

1298

1299 The SCDL for the AutoInsuranceService component of Snippet 7-1 is:

```
1300
1301 <component name="AutoInsuranceService">
1302     <implementation.c module="autoinsurance" path="bin/">
1303         componentType="AutoInsurance" />
1304 </component>
```

1305 *Snippet 7-2: Component Definition Using Implementation in a Common DLL*

1306 7.1.2 Executable shared with other contribution(s) (Export)

1307 If a contribution contains an executable that also implements C components found in other contributions,
1308 the contribution has to export the executable. An executable in a contribution is made visible to other
1309 contributions by adding an `export.c` element to the contribution definition as shown in Snippet 7-3.

```
1310
1311 <contribution>
1312     <deployable composite="myNS:RateUtilities"
```

```
1313     <export.c name="contribNS:rates" >
1314   </contribution>
```

1315 *Snippet 7-3: Exporting a Contribution*

1316

1317 It is also possible to export only a subtree of a contribution. For a contribution:

1318

```
1319   META-INF/
1320     sca-contribution.xml
1321   bin/
1322     rates.dll
1323   RateUtilities/
1324     RateUtilities.composite
1325     RateUtilitiesService/
1326       RateUtils.h
1327       RateUtils.componentType
```

1328 *Snippet 7-4: Contribution with a Subdirectory to be Shared*

1329

1330 An export of the form:

1331

```
1332   <contribution>
1333     <deployable composite="myNS:RateUtilities"
1334       <export.c name="contribNS:ratesbin" path="bin/" >
1335   </contribution>
```

1336 *Snippet 7-5: Exporting a Subdirectory of a Contribution*

1337

1338 only makes the contents of the bin directory visible to other contributions. By placing all of the executable
1339 files of a contribution in a single directory and exporting only that directory, the amount of information
1340 available to a contribution that uses the exported executable files is limited. This is considered a best
1341 practice.

1342 **7.1.3 Executable outside of contribution (Import)**

1343 When the executable that implements a C component is located outside of a contribution, the contribution
1344 has to import the executable. If the executable is located in another contribution, the **import.c** element of
1345 the contribution definition uses a **@location** attribute that identifies the name of the export as defined in
1346 the contribution that defined the export as shown in Snippet 7-6.

1347

```
1348   <contribution>
1349     <deployable composite="myNS:Underwriting"
1350       <import.c name="rates" location="contribNS:rates">
1351   </contribution>
```

1352 *Snippet 7-6: Contribution with an Import*

1353

1354 The SCDL for the UnderwritingService component of Snippet 7-6 is:

1355

```
1356   <component name="UnderwritingService">
1357     <implementation.c module="rates" path="rates:bin/"
1358       componentType="Underwriting" />
1359   </component>
```

1360 *Snippet 7-7: Component Definition Using Implementation in an External DLL*

1361

1362 If the executable is located in the file system, the `@location` attribute identifies the location in the files
1363 system used as the root of the import as shown in Snippet 7-8.

1364

```
1365     <contribution>
1366         <deployable composite="myNS:CustomerUtilities">
1367             <import.c name="usr-bin" location="/usr/bin/" >
1368     </contribution>
```

1369 *Snippet 7-8: Component Definition Using Implementation in a File System*

1370 **7.2 componentType files**

1371 As stated in Component Type and Component, each component implemented in C has a corresponding
1372 componentType file. This componentType file is, by default, located in the root directory of the composite
1373 containing the component or a subdirectory of the composite root with a name specified on the
1374 `@componentType` attribute as shown in the Snippet 7-9.

1375

```
1376     META-INF/
1377         sca-contribution.xml
1378     bin/
1379         autoinsurance.dll
1380     AutoInsurance/
1381         AutoInsurance.composite
1382         AutoInsuranceService/
1383             AutoInsurance.h
1384             AutoInsurance.componentType
```

1385 *Snippet 7-9: Contribution with ComponentType*

1386

1387 The SCDL for the AutoInsuranceService component of Snippet 7-9 is:

1388

```
1389     <component name="AutoInsuranceService">
1390         <implementation.c module="autoinsurance" path="bin/" 
1391             componentType="AutoInsurance" />
1392     </component>
```

1393 *Snippet 7-10: Component Definition with Local ComponentType*

1394

1395 Since there is a one-to-one correspondence between implementations and componentTypes, when an
1396 implementation is shared between contributions, it is desirable to also share the componentType file.
1397 ComponentType files can be exported and imported in the same manner as executable files. The location
1398 of a .componentType file can be specified using the `@componentType` attribute of the `implementation.c`
1399 element.

1400

```
1401     <component name="UnderwritingService">
1402         <implementation.c library="rates" path="rates:bin/" 
1403             componentType="rates:types/Underwriting" />
1404     </component>
```

1405 *Snippet 7-11: Component Definition with Imported ComponentType*

1406 **7.3 C Contribution Extensions**

1407 **7.3.1 Export.c**

1408 Snippet 7-12 shows the pseudo-schema for the C export element used to make an executable or
1409 componentType file visible outside of a contribution.

```

1410
1411     <!-- export.c schema snippet -->
1412     <export.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1413         name="QName" path="string"? >

```

1414 Snippet 7-12: Pseudo-schema for C Export Element

1415

1416 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.

1421 7.3.2 Import.c

1422 Snippet 7-13 shows the pseudo-schema for the C import element used to reference an executable or
1423 componentType file that is outside of a contribution.

1424

```

1425     <!-- import.c schema snippet -->
1426     <import.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1427         name="QName" location="string" >

```

1428 Snippet 7-13: Pseudo-schema for C Import Element

1429

1430 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.

8 C Interfaces

1435 A service interface can be defined by a set of C function and/or struct declarations.
1436 When mapping a C interface to WSDL or when comparing two C interfaces for compatibility, as defined
1437 by the Assembly specification [ASSEMBLY], it is necessary for an SCA implementation to determine the
1438 signature (return type, name, and the names and types of the parameters) of every function or the type of
1439 every member of every struct in the service interface definition. An SCA implementation **MUST** translate
1440 declarations to tokens as part of conversion to WSDL or compatibility testing. [C80001] Snippet 8-1
1441 shows a case where a macro has to be processed to understand the return type of a function.
1442

```
1443
1444     #if LIB_BUILD
1445     # define DECLSPEC_FUNC(ReturnType) __declspec(dllexport) ReturnType
1446     #else
1447     # define DECLSPEC_FUNC(ReturnType) __declspec(dllexport) ReturnType
1448     #endif
1449
1450     DECLSPEC_FUNC(int) fooFunc(void) {}
```

1451 *Snippet 8-1: Example Macro Impacting Function Signature*

1452
1453 Macros and typedefs in function or struct declarations might lead to portability problems. Complete
1454 function or struct declarations within a macro are discouraged. The processing of typedefs needs to be
1455 aware of the types that impact mapping to WSDL (see Table 9-1 and Table 9-2)

8.1 Types Supported in Service Interfaces

1456 Not all service interfaces support the complete set of the types available in C.

8.1.1 Local Service

1457 Any fundamental or compound type defined by C can be used in the interface of a local service.

8.1.2 Remotable Service

1458 For a remotable service being called by another service the data exchange semantics is by-value. The
1459 return type and types of the parameters of a function of a remotable service interface **MUST** be one of:

- Any of the C types specified in Simple Content Binding and Complex Content Binding. These types
1460 may be passed by-value or by-pointer. Unless the function and client indicate that they allow by-
1461 reference semantics (see AllowsPassByReference), a copy will be explicitly created by the runtime
1462 for any parameters passed by-pointer.
- An SDO DATAOBJECT. This type may be passed by-value or by-pointer. Unless the function and
1463 client indicate that they allow by-reference semantics (see AllowsPassByReference), a deep-copy of
1464 the DATAOBJECT will be created by the runtime for any parameters passed by-value or by-pointer.
1465 When by-reference semantics are allowed, the DATAOBJECT handle will be passed. [C80002]

8.2 Restrictions on C header files

1466 A C header file used to define an interface **MUST** declare at least one function or message format struct
1467 [C80003]

1468 Function definitions in a header file are not considered part of a service definition.

9 WSDL to C and C to WSDL Mapping

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

A detailed mapping of C to WSDL types and WSDL to C types is covered in Data Binding.

General rules for the application of JAX-WS to C:

- References to Java are considered references to C.
- References to Java classes are considered references to a collection of C functions or programs that implement an interface.
- References to Java methods are considered references to C functions or message format struct declarations.
- References to Java interfaces are considered references to a collection of C function or message format struct declarations used to define an interface.

9.1 Interpretations for WSDL to C Mapping

External binding files are not supported.

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

9.1.1 Definitions

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

MIME binding is not supported.

9.1.2 PortType

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

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

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

An SCA implementation MUST map a WSDL portType to a remotable C interface definition. [C100023]

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. [C100001] For example, all portTypes in a WSDL document with a common sca-c:prefix binding declaration could be mapped to a single header file.

Header file naming is implementation dependent.

1515 9.1.3 Operations

1516 Asynchronous mapping is not supported.

1517 9.1.3.1 Operation Names

1518 WSDL operation names are only guaranteed to be unique with a portType. C requires function and struct
1519 names loaded into an address space to be distinct. The mapping of operation names to function or struct
1520 names have to take this into account.

1521 For components implemented in libraries, in the absence of customizations, an SCA implementation
1522 MUST map an operation name, with the first character converted to lower case, to a function name. If
1523 necessary, to avoid name collisions, an SCA implementation MAY prepend the portType name, with the
1524 first character converted to lower case, and the operation name, with the first character converted to
1525 upper case, to form the function name. [C100002]

1526 An application can customize this mapping using the sca-c:prefix and/or sca-c:function binding
1527 declarations.

1528 For program-based service implementations:

- 1529 • If the number of **In** parameters plus the number of **In/Out** parameters is greater than one there will be
1530 a request struct.
- 1531 • If the number of **Out** parameters plus the number of **In/Out** parameters is greater than one there will
1532 be a response struct.

1533 For components implemented in a program, in the absence of customizations, an SCA implementation
1534 MUST map an operation name, with the first character converted to lowercase to a request struct name. If
1535 necessary, to avoid name collisions, an SCA implementation MAY concatenate the portType name, with
1536 the first character converted to lower case, and the operation name, with the first character converted to
1537 upper case, to form the request stuct name. Additionally an SCA implementation MUST append
1538 "Response" to the request struct name to form the response struct name. [C100005]

1539 An application can customize this mapping using the sca-c:prefix and/or sca-c:struct binding declarations.

1540 9.1.3.2 Message and Part

1541 In the absence of any customizations for a WSDL operation that does not meet the requirements for the
1542 wrapped style, the name of a mapped function parameter or struct member MUST be the value of the
1543 name attribute of the wsdl:part element with the first character converted to lower case. [C100003]

1544 In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped
1545 style, the name of a mapped function parameter or struct member MUST be the value of the local name
1546 of the wrapper child with the first character converted to lower case. [C100004]

1547 An application can customize this mapping using the sca-c:parameter binding declaration.

1548 For library-based service implementations, an SCA implementation MUST map **In** parameters as pass
1549 by-value or const and **In/Out** and **Out** parameters as pass via pointers. [C100019]

1550 For program-based service implementations, an SCA implementation MUST map all values in the input
1551 message as pass by-value and the updated values for **In/Out** parameters and all **Out** parameters in the
1552 response message as pass by-value. [C100020]

1553 9.1.4 Types

1554 As per section Data Binding (based on SDO type mapping).

1555 MTOM/XOP content processing is left to the application.

1556 9.1.5 Fault

1557 C has no exceptions so an API is provided for getting and setting fault messages (see
1558 SCAGetFaultMessage and SCASetFaultMessage). Fault messages are mapped in same manner as
1559 input and output messages.

1560 In the absence of customizations, an SCA implementation MUST map the name of the message element
1561 referred to by a fault element to the name of the struct describing the fault message content. If necessary,
1562 to avoid name collisions, an implementation MAY append “*Fault*” to the name of the message element
1563 when mapping to the struct name. [C100006]

1564 An application can customize this mapping using the sca-c:struct binding declaration.

1565 **9.1.6 Service and Port**

1566 This mapping does not define generation of client side code.

1567 **9.1.7 XML Names**

1568 See comments in Operations

1569 **9.2 Interpretations for C to WSDL Mapping**

1570 Where annotations are discussed as a means for an application to control the mapping to WSDL, an
1571 implementation-specific means of controlling the mapping can be used instead.

1572 **9.2.1 Package**

1573 Not relevant.

1574 An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be
1575 configurable. [C100007]

1576 **9.2.2 Class**

1577 Not relevant since mapping is only based on declarations.

1578 **9.2.3 Interface**

1579 The declarations in a header file are used to define an interface. A header file can be used to define an
1580 interface if it satisfies either (for components implemented in libraries):

- 1581 • Contains one or more function declarations
- 1582 • Any of these functions declarations might carry a @WebFunction annotation
- 1583 • The parameters and return types of these function declarations are compatible with the C to XML
1584 Schema mapping in Data Binding

1585 or (for components implemented in programs):

- 1586 • Contains one request message struct declarations
- 1587 • Any of the request message struct declarations might carry a @WebOperation annotation
- 1588 • Any of the request message struct declarations can have a corresponding response message struct,
1589 identified by either having a name with “Response” appended to the request message struct name or
1590 identified in a @WebOperation annotation
- 1591 • Members of these struct declarations are compatible with the C to XML Schema mapping in Data
1592 Binding

1593 An SCA implementation MUST map a C interface definition to WSDL as if it has a @WebService
1594 annotation with all default values. [C100024]

1595 In the absence of customizations, an SCA implementation MUST map the header file name to the
1596 portType name. An implementation MAY append “*PortType*” to the header file name in the mapping to the
1597 portType name. [C100008]

1598 An application can customize this mapping using the @WebService annotation.

- 1599 **9.2.4 Method**
- 1600 For components implemented in libraries, functions map to operations.
- 1601 In the absence of customizations, an SCA implementation MUST map a function name to an operation name, stripping the portType name, if present and any namespace prefix from the front of function name before mapping it to the operation name. [C100009]
- 1604 An application can customize function to operation mapping or exclude a function from an interface using the @WebFunction annotation.
- 1606 For components implemented in programs, operations are mapped from request structs.
- 1607 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. [C100010]
- 1611 An application can customize stuct to operation mapping or exclude a struct from an interface using the @WebOperation annotation.
- 1613 **9.2.5 Method Parameters and Return Type**
- 1614 For components implemented in libraries, function parameters and return type map to either message or global element components.
- 1616 In the absence of customizations, an SCA implementation MUST map a parameter name, if present, to a 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. [C100011]
- 1619 An application can customize parameter to message or global element component mapping using the @WebParam annotation.
- 1621 In the absence of customizations, an SCA implementation MUST map the return type to a part or global element child named “return”. [C100012]
- 1623 An application can customize return type to message or global element component mapping using the @WebReturn annotation.
- 1625 An SCA implementation MUST map:
- 1626 • a function’s return value as an **out** parameter.
 - 1627 • by-value and const parameters as **in** parameters.
 - 1628 • in the absence of customizations, pointer parameters as **in/out** parameters. [C100017]
- 1629 An application can customize parameter classification using the @WebParam annotation.
- 1630 Program based implementation SHOULD use the Document-Literal style and encoding. [C100013]
- 1631 In the absence of customizations, an SCA implementation MUST map the struct member name to the part or global element child name. [C100014]
- 1633 An application can customize struct member to message or global element component mapping using the @WebParam annotation.
- 1635 • Members of the request struct that are not members of the response struct are **in** parameters
 - 1636 • Members of the response struct that are not members of the request struct are **out** parameters
 - 1637 • Members of both the request and response structs are **in/out** parameters. Matching is done by member name. An SCA implementation MUST ensure that **in/out** parameters have the same type in the request and response structs. [C100015]
- 1640 **9.2.6 Service Specific Exception**
- 1641 C has no exceptions. A struct can be annotated as a fault message type. A function or operation declaration can be annotated to indicate that it potentially generates a specific fault.
- 1643 An application can define a fault message format using the @WebFault annotation.

1644 An application can indicate that a WSDL fault might be generated by a function or operation using the
1645 @WebThrows annotation.

1646 **9.2.7 Generics**

1647 Not relevant.

1648 **9.3 Data Binding**

1649 The data in wsdl:parts or wrapper children is mapped to and from C function parameters and return
1650 values (for library-based component implementations), or struct members (for program-based component
1651 implementations and fault messages).

1652 **9.3.1 Simple Content Binding**

1653 The mapping between XSD simple content types and C types follows the convention defined in the SDO
1654 specification [SDO21]. Table 9-1 summarizes that mapping as it applies to SCA services.

1655

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

XSD Schema Type →	C Type	→ XSD Schema Type
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

1656 *Table 9-1: XSD simple type to C type mapping*

1657

1658 Table 9-2 defines the mapping of C++ types to XSD schema types that are not covered in Table 9-1.

1659

C Type →	XSD Schema Type
_Bool	boolean
wchar_t	string

C Type →	XSD Schema Type
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
long double	decimal
time_t	time
struct tm	dateTime

1660 *Table 9-2: C type to XSD type mapping*

1661

1662 The C standard does not define value ranges for integer types so it is possible that on a platform
 1663 parameters or return values could have values that are out of range for the default XSD schema type. In
 1664 these circumstances, the mapping would need to be customized, using @WebParam or @WebResult if
 1665 supported, or some other implementation-specific mechanism.

1666 An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.
 1667 [C100021]

1668 An SCA implementation MAY map boolean to _Bool by default. [C100022]

1669 9.3.1.1 WSDL to C Mapping Details

1670 In general, when xsd:string and types derived from xsd:string map to a struct member, the
 1671 mapping is to a combination of a wchar_t * and a separately allocated data array. If either the length
 1672 or maxLength facet is used, then a wchar_t[] is used. If the pattern facet is used, this might allow
 1673 the use of char and/or also constrain the length.

1674 Example:

```
<xsd:element name="myString" type="xsd:string"/>
maps to:
wchar_t *myString;
/* this points to a dynamically allocated buffer with the data */
```

1679 *Snippet 9-1: Unbounded String Mapping*

1680

```
<xsd:simpleType name="boundedString25">
  <xsd:restriction base="xsd:string">
    <xsd:length value="25"/>
  </xsd:restriction>
</xsd:simpleType>
...
```

```
1687     <xsd:element name="myString" type="boundedString25"/>
1688 maps to:
1689     wchar_t myString[26];
```

1690 *Snippet 9-2: Bounded String Mapping*

1691

- 1692 • When unbounded binary data maps to a struct member, the mapping is to a `char *` that points to
1693 the location where the actual data is located. Like strings, if the binary data is bounded in length, a
1694 `char[]` is used.

1695 Examples:

```
1696     <xsd:element name="myData" type="xsd:hexBinary"/>
1697 maps to:
1698     char *myData;
1699     /* this points to a dynamically allocated buffer with the data */
```

1700 *Snippet 9-3: Unbounded Binary Data Mapping*

1701

```
1702     <xsd:simpleType name="boundedData25">
1703         <xsd:restriction base="xsd:hexBinary">
1704             <xsd:length value="25"/>
1705         </xsd:restriction>
1706     </xsd:simpleType>
1707 ...
1708     <xsd:element name="myData" type="boundedData25"/>
1709 maps to:
```

```
1710     char myData[26];
```

1711 *Snippet 9-4: Bounded Binary Data Mapping*

1712

- 1713 • 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
1714 of instances, map to a struct, the mapping is to a count of the number of occurrences and an array. If
1715 the count is 0, then the content of the array is undefined.

1717 Examples:

```
1718     <xsd:element name="counts" type="xsd:int" maxOccurs="5"/>
1719 maps to:
1720     size_t counts_num;
1721     int counts[5];
```

1722 *Snippet 9-5: minOccurs != maxOccurs Mapping*

1723

```
1724     <xsd:simpleType name="lineNumList">
1725         <xsd:list itemType="xsd:int"/>
1726     </xsd:simpleType>
1727     <xsd:simpleType name="lineNumList6">
1728         <xsd:restriction base="lineNumList ">
1729             <xsd:minLength value="1"/>
1730             <xsd:maxLength value="6"/>
1731         </xsd:restriction>
1732     </xsd:simpleType>
1733 ...
1734     <xsd:element name="lineNums" type="lineNumList6"/>
1735 maps to:
```

```
1736     size_t lineNums_num;
```

```
1737     long lineNums[6];  
1738 Snippet 9-6: minLength != maxLength Mapping  
1739  
1740 • Since C does not allow for unbounded arrays, when elements with maxOccurs = unbounded and  
1741 lists without a defined length or maxLength, map to a struct, the mapping is to a count of the  
1742 number of occurrences and a pointer to the location where the actual data is located as an array  
1743 Examples:  
1744 <xsd:element name="counts" type="xsd:int" maxOccurs="unbounded"/>  
1745 maps to:
```

```
1746     size_t counts_num;  
1747     int *counts;  
1748     /* this points to a dynamically allocated array of longs */
```

1749 *Snippet 9-7: Unbounded Array Mapping*

1750

- 1751 • Union Types are not supported.

1752 9.3.1.2 C to WSDL Mapping Details

- 1753 • wchar_t[] and char[] map to xsd:string with a maxLength facet.
- 1754 • C arrays map as normal elements but with multiplicity allowed via the minOccurs and maxOccurs facets.

1756 Example:

```
1757     int idList[];  
1758 maps to:  
1759     <xsd:element name="idList" type="xsd:int"  
1760                 minOccurs="0" maxOccurs="unbounded"/>
```

1761 *Snippet 9-8: Array Mapping*

1762

- 1763 • Multi-dimensional arrays map into nested elements.

1764 Example:

```
1765     int multiIdArray[4][2];  
1766 maps to:  
1767     <xsd:element name="multiIdArray"  
1768                 minOccurs="0" maxOccurs="4"/>  
1769     <xsd:complexType>  
1770         <xsd:sequence>  
1771             <xsd:element name="multiIdArray" type="xsd:int"  
1772                         minOccurs="2" maxOccurs="2" />  
1773         </xsd:sequence>  
1774     </xsd:complexType>  
1775 </xsd:element>
```

1776 *Snippet 9-9: Multi-Dimensional Array Mapping*

1777

- 1778 • Except as detailed in the table above, pointers do not affect the type mapping, only the classification
1779 as in, out, or in/out.

9.3.2 Complex Content Binding

When mapping between XSD complex content types and C, either instances of SDO DataObjects or structs are used. 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 struct. The mapping from WSDL MAY be to DataObjects and/or structs. The mapping to and from structs MUST follow the rules defined in WSDL to C Mapping Details. [C100016]

9.3.2.1 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.

Example:

```
<xsd:complexType name="myType">
  <xsd:sequence>
    <xsd:element name="name">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:length value="25"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="idList" type="xsd:int"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="value" type="xsd:double"/>
  </xsd:sequence>
</xsd:complexType>
```

maps to:

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

Snippet 9-10: Sequence Group Mapping

Example:

```
<xsd:element name="myVariable">
  <xsd:complexType name="myType">
    <xsd:all>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="idList" type="xsd:int"
        minOccurs="0" maxOccurs="unbounded"/>
```

```

1831     <xsd:element name="value" type="xsd:double"/>
1832   </xsd:all>
1833 </xsd:complexType>
1834 </xsd:element>
1835 maps to:
1836 struct myType {
1837   wchar_t *name;
1838   /* this points to a dynamically allocated string */
1839   size_t idList_num;
1840   long *idList;
1841   /* this points to a dynamically allocated array of longs */
1842   double *value;
1843   /* this points to a dynamically allocated long */
1844 } *pmyVariable, myVariable;

```

1845 *Snippet 9-11: All Group Mapping*

1846

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

1851 Example:

```

1852 <xsd:element name="priority" type="xsd:short" nillable="true"/>
1853 maps to:

```

```

1854 int16_t *pprioiry, priority;

```

1855 *Snippet 9-12: Nillable Mapping*

1856

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

9.3.2.2 C to WSDL Mapping Details

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

1860 Example:

```

1861 struct DataStruct {
1862   char *name;
1863   double value;
1864 };

```

1865 maps to:

```

1866 <xsd:complexType name="DataStruct">
1867   <xsd:sequence>
1868     <xsd:element name="name" type="xsd:string"/>
1869     <xsd:element name="value" type="xsd:double"/>
1870   </xsd:sequence>
1871 </xsd:complexType>

```

1872 *Snippet 9-13: Struct Mapping*

1873

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

1876 Example:

```

1877 struct DataStruct {
1878   char name[256];
1879   double value;

```

```

1880    } ;
1881 maps to:
1882 <xsd:complexType name="DataStruct">
1883   <xsd:sequence>
1884     <xsd:element name="name">
1885       <xsd:simpleType>
1886         <xsd:restriction base="xsd:string">
1887           <xsd:maxLength value="255"/>
1888         </xsd:restriction>
1889       </xsd:simpleType>
1890     </xsd:element>
1891     <xsd:element name="value" type="xsd:double"/>
1892   </xsd:sequence>
1893 </xsd:complexType>
```

1894 *Snippet 9-14: Character Array in Struct Mapping*

- 1895
- 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.
- 1898 Example:

```

1899 enum ParameterType {
1900   UNSET = 1,
1901   TYPEA,
1902   TYPEB,
1903   TYPEC
1904 };
1905 maps to:
```

```

1906 <xsd:simpleType name="ParameterType">
1907   <xsd:restriction base="xsd:int">
1908     <xs:minInclusive value="1"/>
1909     <xs:maxInclusive value="4"/>
1910   </xsd:restriction>
1911 </xsd:simpleType>
```

1912 *Snippet 9-15: Enum Mapping*

1913

1914 The restriction used will have to be appropriate to the values of the enum elements.

1915 Example:

```

1916 enum ParameterType {
1917   UNSET = 'u',
1918   TYPEA = 'A',
1919   TYPEB = 'B',
1920   TYPEC = 'C'
1921 };
1922 maps to:
```

```

1923 <xsd:simpleType name="ParameterType">
1924   <xsd:restriction base="xsd:int">
1925     <xsd:enumeration value="86"/> <!-- Character 'u' -->
1926     <xsd:enumeration value="65"/> <!-- Character 'A' -->
1927     <xsd:enumeration value="66"/> <!-- Character 'B' -->
1928     <xsd:enumeration value="67"/> <!-- Character 'C' -->
1929   </xsd:restriction>
1930 </xsd:simpleType>
```

1931 *Snippet 9-16: Non-contiguous Value Enum Mapping*

1932

- 1933 • If a struct or enum contains other structs or enums, the mapping rules are applied recursively.

1934 Example:

1935 **struct DataStruct data;**
1936 with types defined as follows:

```
1937     struct DataStruct {  
1938         char name[30];  
1939         double values[20];  
1940         ParameterType type;  
1941     }  
1942  
1943     enum ParameterType {  
1944         UNSET = 1,  
1945         TYPEA,  
1946         TYPEB,  
1947         TYPEC  
1948     }  
1949     maps to:
```

```
1950     <xsd:complexType name="DataStruct">  
1951         <xsd:sequence>  
1952             <xsd:element name="name">  
1953                 <xsd:simpleType>  
1954                     <xsd:restriction base="xsd:string">  
1955                         <xsd:maxLength value="29"/>  
1956                     </xsd:restriction>  
1957                 </xsd:simpleType>  
1958             </xsd:element>  
1959             <xsd:element name="values" type="xsd:double" minOccurs=20 maxOccurs=20/>  
1960                 <xsd:element name="type" type=" ParameterType"/>  
1961             </xsd:sequence>  
1962     </xsd:complexType>  
1963  
1964     <xsd:simpleType name="ParameterType">  
1965         <xsd:restriction base="xsd:int">  
1966             <xs:minInclusive value="1"/>  
1967             <xs:maxInclusive value="4"/>  
1968         </xsd:restriction>  
1969     </xsd:simpleType>
```

1970 *Snippet 9-17: Nested Struct Mapping*

- 1971
- 1972 • Mapping of C unions is not supported by this specification.
- 1973 • Typedefs are resolved when evaluating parameter and return types. Typedefs are resolved before the mapping to Schema is done.
- 1974

1975 10 Conformance

1976 The XML schema pointed to by the RDDL document at the SCA namespace URI, defined by the
1977 Assembly specification **[ASSEMBLY]** and extended by this specification, are considered to be
1978 authoritative and take precedence over the XML schema in this document.

1979 The XML schema pointed to by the RDDL document at the SCA C namespace URI, defined by this
1980 specification, is considered to be authoritative and takes precedence over the XML schema in this
1981 document.

1982 Normative code artifacts related to this specification are considered to be authoritative and take
1983 precedence over specification text.

1984 An SCA implementation MUST reject a composite file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd> or <http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd>. **[C110001]**

1987 An SCA implementation MUST reject a componentType file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd>. **[C110002]**

1989 An SCA implementation MUST reject a contribution file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200912/sca-contribution-c-1.1.xsd>. **[C110003]**

1991 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]**

1993 10.1 Conformance Targets

1994 The conformance targets of this specification are:

- 1995 • **SCA implementations**, which provide a **runtime** for SCA components and potentially **tools** for
1996 authoring SCA artifacts, component descriptions and/or runtime operations.
- 1997 • **SCA documents**, which describe SCA artifacts, and specific **elements** within these documents.
- 1998 • **C files**, which define SCA service interfaces and implementations.
- 1999 • **WSDL files**, which define SCA service interfaces.

2000 10.2 SCA Implementations

2001 An implementation conforms to this specification if it meets these conditions:

- 2002 1. It MUST conform to the SCA Assembly Model Specification **[ASSEMBLY]** and the SCA Policy
2003 Framework **[POLICY]**.
- 2004 2. It MUST comply with all statements in Table F-1 and Table F-5 related to an SCA implementation,
2005 notably all mandatory statements have to be implemented.
- 2006 3. It MUST implement the SCA C API defined in section SCA Programming Interface.
- 2007 4. It MAY support program-based component implementations. If program-based component
2008 implementations are supported, the implementation MUST implement the Program-Based
2009 Implementation Support API defined in Program-Based Implementation Support and MUST comply
2010 with all statements in Table F-2 related to an SCA implementation, notably all mandatory statements
2011 in that section have to be implemented.
- 2012 5. It MUST implement the mapping between C and WSDL 1.1 **[WSDL11]** defined in WSDL to C and C
2013 to WSDL Mapping.
- 2014 6. It MUST support `<interface.c/>` and `<implementation.c/>` elements as defined in Component Type
2015 and Component in composite and componentType and documents.
- 2016 7. It MUST support `<export.c/>` and `<import.c/>` elements as defined in C Contributions in contribution
2017 documents.

- 2018 8. It MAY support source file annotations as defined in C SCA Annotations, C SCA Policy Annotations
2019 and C WSDL Annotations. If source file annotations are supported, the implementation MUST comply
2020 with all statements in Table F-3 related to an SCA implementation, notably all mandatory statements
2021 in that section have to be implemented.
- 2022 9. It MAY support WSDL extensions as defined in C WSDL Mapping Extensions. If WSDL extensions
2023 are supported, the implementation MUST comply with all statements in Table F-4 related to an SCA
2024 implementation, notably all mandatory statements in that section have to be implemented.

2025 **10.3 SCA Documents**

2026 An SCA document conforms to this specification if it meets these conditions:

- 2027 1. It MUST conform to the SCA Assembly Model Specification [**ASSEMBLY**] and, if appropriate, the
2028 SCA Policy Framework [**POLICY**].
- 2029 2. If it is a composite document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd> and <http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd> schema and MUST comply with the
2030 additional constraints on the document contents as defined in Table F-1.
2031 If it is a componentType document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd> schema and MUST comply with the additional
2032 constraints on the document contents as defined in Table F-1.
2033 If it is a contribution document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200912/sca-contribution-c-1.1.xsd> schema and MUST comply with the
2034 additional constraints on the document contents as defined in Table F-1.
2035

2039 **10.4 C Files**

2040 A C file conforms to this specification if it meets the conditions:

- 2041 1. It MUST comply with all statements in Table F-1, Table F-3 and Table F-5 related to C contents and
2042 annotations, notably all mandatory statements have to be satisfied.

2043 **10.5 WSDL Files**

2044 A WSDL conforms to this specification if it meets these conditions:

- 2045 1. It is a valid WSDL 1.1 [**WSDL11**] document.
- 2046 2. It MUST comply with all statements in Table F-1, Table F-4 and Table F-5 related to WSDL contents
2047 and extensions, notably all mandatory statements have to be satisfied.

2048 A C SCA Annotations

2049 To allow developers to define SCA related information directly in source files, without having to separately
2050 author SCDL files, a set of annotations is defined. If SCA annotations are supported by an
2051 implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as
2052 described. The SCA runtime MUST only process the SCDL files and not the annotations. [CA0001]

2053 A.1 Application of Annotations to C Program Elements

2054 In general an annotation immediately precedes the program element it applies to. If multiple annotations
2055 apply to a program element, all of the annotations SHOULD be in the same comment block. [CA0002]

- 2056 • Function or Function Prototype

2057 The annotation immediately precedes the function definition or declaration.

2058 Example:

```
2059 /* @OneWay */  
2060 reportEvent(int eventID);
```

2061 *Snippet A-1: Example Function Annotation*

- 2062 • Variable

2063 The annotation immediately precedes the variable definition.

2064 Example:

```
2065 /* @Property */  
2066 long loanType;
```

2067 *Snippet A-2: Example Variable Annotation*

- 2068 • Set of Functions Implementing a Service

2069 A set of functions implementing a service begins with an @Service annotations. Any annotations
2070 applying to this service as a whole immediately precede the @Service annotation. These annotations
2071 SHOULD be in the same comment block as the @Service annotation.

2072 Example:

```
2073 /* @ComponentType  
2074 * @Service(name="LoanService", interfaceHeader="loan.h") */
```

2075 *Snippet A-3: Example Set of Functions Annotation*

- 2076 • Set of Function Prototypes Defining an Interface

2077 To avoid any ambiguity about the application of an annotation to a specific function or the set of
2078 functions defining an interface, if an annotation is to apply to the interface as a whole, then the
2079 @Interface annotation is used, even in the case where there is just one interface defined in a header
2080 file. Any annotations applying to the interface immediately precede the @Interface annotation.

```
2081 /* @Remoteable  
2082 * @Interface(name="LoanService") */
```

2083 *Snippet A-4: Example Set of Function Declarations Annotation*

2084 A.2 Interface Header Annotations

2085 This section lists the annotations that can be used in the header file that defines a service interface.

2086 A.2.1 @Interface

2087 Annotation that indicates the start of a new interface definition.

2088 **Corresponds to:** *interface.c* element
 2089 **Format:**
 2090 `/* @Interface(name="serviceName") */`
 2091 *Snippet A-5: @Interface Annotation Format*
 2092 where
 2093 • **name : NCName (0..1)** – specifies the name of a service using this interface. The default is the root name of the header file containing the annotation.
 2094
 2095 **Applies to:** Set of functions defining an interface.
 2096 Function declarations following this annotation form the definition of this interface. This annotation also
 2097 serves to bound the scope of the remaining annotations in this section,
 2098 Example:
 2099 Interface header:
 2100 `/* @Interface(name="LoanService") */`
 2101
 2102 Service definition:
 2103 `<service name="LoanService">`
 2104 `<interface.c header="loans.h" />`
 2105 `</service>`
 2106 *Snippet A-6: Example of @Interface Annotation*

2107 **A.2.2 @Function**

2108 Annotation that indicates that a function defines an operation of a service. There are two formats for this
 2109 annotation depending on if the service is implemented as a set of subroutines or in a program. An SCA
 2110 implementation MUST treat a function with a @WebFunction annotation specified as if @Function was
 2111 specified with the operationName value of the @WebFunction annotation used as the name value of the
 2112 @Function annotation and the exclude value of the @WebFunction annotation used as the exclude value
 2113 of the @Function annotation. [CA0004]

2114 **Corresponds to:** *function* or *callbackFunction* child element of an *interface.c* element. If the file the
 2115 function is contained in is being processed because it was identified via either
 2116 *interface.c/@callbackHeader* or a @Callback annotation, then the @Function annotation corresponds to
 2117 a *callbackFunction* element, otherwise it corresponds to a *function* element.

2118 **Format:**
 2119 `/* @Function(name="operationName", exclude="true") */`
 2120 *Snippet A-7: @Operation Annotation Format for Functions*
 2121 where
 2122 • **name : NCName (0..1)** – specifies the name of the operation. The default operation name is the
 2123 function name.
 2124 • **exclude : boolean (0..1)** – specifies whether this function is to be excluded from the SCA interface.
 2125 Default is **false**.
 2126 **Applies to:** Function declaration
 2127 Example:
 2128 Interface header (loans.h):
 2129 `short internalFcn(char *param1, short param2);`
 2130
 2131 `/* @Function(name="getRate") */`
 2132 `void rateFcn(char *cust, float *rate);`

2133

2134 Interface definition:

```
2135     <interface.c header="loans.h">
2136         <function name="getRate" />
2137     </interface.c>
```

2138 *Snippet A-8: Example of @Operation Annotation for Functions*

2139 A.2.3 @Operation

2140 Annotation that indicates a struct declaration defines a request message format of an operation of a service. An SCA implementation MUST treat a struct with a @WebOperation annotation specified as if @Operation was specified with the operationName value of the @WebOperation annotation used as the name value of the @Operation annotation, the response value of the @WebOperation annotation used as the response value of the @Operation annotation and the exclude value of the @WebFunction annotation used as the exclude value of the @Operation annotation. [CA0005]

2141 **Corresponds to:** *function* or *callbackFunction* child element of an *interface.c* element. If the file the struct is contained in is being processed because it was identified via either *interface.c/@callbackHeader* or a *@Callback* annotation, then the @Operation annotation corresponds to a *callbackFunction* element, otherwise it corresponds to a *function* element.

2142 **Format:**

```
2143     /* @Operation(name="operationName", response="outStruct", exclude="true") */
```

2144 *Snippet A-9: @Operation Annotation Format for Structs*

2145 where

- **name : NCName (1..1)** – specifies the name of the operation. The default operation name is the name of the request message struct.
- **response : NCName (0..1)** – specifies the name of a struct that defined the format of the response message if one is used.
- **exclude : boolean (0..1)** – specifies whether this struct is to be excluded from the SCA interface. Default is **false**.

2146 **Applies to:** stuct declarations

2147 Example:

2148 Interface header (loans.h):

```
2149     /* @Operation(name="getRate", response="rateOutput") */
2150     struct rateInput {
2151         char cust[25];
2152         int term;
2153     };
2154     struct rateOutput {
2155         float rate;
2156         int rateClass;
2157     };
2158 
```

2159

2160 Interface definition:

```
2161     <interface.c header="loans.h">
2162         <function name="getRate" input="rateInput" output="rateOutput"/>
2163     </interface.c>
```

2164 *Snippet A-10: Example of @Operation Annotation for Structs*

2178 A.2.4 @Remotable

2179 Annotation on service interface to indicate that a service is remotable and implies an @Interface
2180 annotation applies as well. An SCA implementation MUST treat a file with a @WebService annotation
2181 specified as if @Remotable and @Interface were specified with the name value of the @WebService
2182 annotation used as the name value of the @Interface annotation. [CA0003]

2183 **Corresponds to:** @remotable="true" attribute of an *interface.c* element.

2184 **Format:**

```
2185 /* @Remotable */
```

2186 *Snippet A-11: @Remotable Annotation Format*

2187 The default is **false** (not remotable).

2188 **Applies to:** Interface

2189 Example:

2190 Interface header (LoanService.h):

```
2191 /* @Remotable */
```

2192

2193 Service definition:

```
2194 <service name="LoanService">
2195   <interface.c header="LoanService.h" remotable="true" />
2196 </service>
```

2197 *Snippet A-12: Example of @Remotable Annotation*

2198 A.2.5 @Callback

2199 Annotation on a service interface to specify the callback interface.

2200 **Corresponds to:** @callbackHeader attribute of an *interface.c* element.

2201 **Format:**

```
2202 /* @Callback(header="headerName") */
```

2203 *Snippet A-13: @Callback Annotation Format*

2204 where

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

2206 **Applies to:** Interface

2207 Example:

2208 Interface header (MyService.h):

```
2209 /* @Callback(header="MyServiceCallback.h") */
```

2210

2211 Service definition:

```
2212 <service name="MyService">
2213   <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h" />
2214 </service>
```

2215 *Snippet A-14: Example of @Callback Annotation*

2216 A.2.6 @OneWay

2217 Annotation on a service interface function declaration to indicate the function is one way. The @OneWay
2218 annotation also affects the representation of a service in WSDL. See @OneWay.

2219 **Corresponds to:** @oneWay="true" attribute of function element of an *interface.c* element.

2220 **Format:**
2221 /* @OneWay */

2222 *Snippet A-15: @OneWay Annotation Format*
2223 The default is **false** (not OneWay).
2224 **Applies to:** Function declaration
2225 Example:

2226 Interface header:

```
2227     /* @OneWay */  
2228     reportEvent(int eventID);
```

2229

2230 Service definition:
2231 <service name="LoanService">
2232 <interface.c header="LoanService.h">
2233 <function name="reportEvent" oneWay="true" />
2234 </interface.c>
2235 </service>

2236 *Snippet A-16: Example of @OneWay Annotation*

A.3 Implementation Annotations

2238 This section lists the annotations that can be used in the file that implements a service.

A.3.1 @ComponentType

2240 Annotation used to indicate the start of a new componentType.

2241 **Corresponds to:** @componentType attribute of an *implementation.c* element.

2242 **Format:**

```
2243     /* @ComponentType */
```

2244 *Snippet A-17: @ComponetType Annotation Format*

2245 **Applies to:** Set of services, references and properties

2246 Example:

2247 Implementation:

```
2248     /* @ComponentType */
```

2249

2250 Component definition:

```
2251       <component name="LoanService">  
2252         <implementation.c module="loan" componentType="LoanService" />  
2253      </component>
```

2254 *Snippet A-18: Example of @ComponentType Annotation*

A.3.2 @Service

2256 Annotation that indicates the start of a new service implementation.

2257 **Corresponds to:** *implementation.c* element

2258 **Format:**

```
2259     /* @Service(name="serviceName", interfaceHeader="headerFile") */
```

2260 *Snippet A-19: @Service Annotation Format*

2261 where

- ***name : NCName (0..1)*** – specifies the name of the service. The default is the service name for the interface.
- ***interfaceHeader : Name (1..1)*** – specifies the C header defining the interface.

2264 **Applies to:** Set of functions implementing a service

2265 Function definitions following this annotation form the implementation of this service. This annotation also serves to bound the scope of the remaining annotations in this section,

2266 Example:

2267 Implementation:

```
2268        /* @Service(name="LoanService", interfaceHeader="loan.h") */
```

2269 Implementation:

2270 /* @Service(name="LoanService", interfaceHeader="loan.h") */

2271 Implementation:

2272 ComponentType definition:

```
2273        <componentType name="LoanService">
2274            <service name="LoanService">
2275                <interface.c header="loans.h" />
2276            </service>
2277        </componentType>
```

2278 *Snippet A-20: Example of @Service Annotation*

2279 **A.3.3 @Reference**

2280 Annotation on a service implementation to indicate it depends on another service providing a specified interface.

2281 **Corresponds to:** *reference* element of a *componentType* element.

2282 **Format:**

```
2283        /* @Reference(name="referenceName", interfaceHeader="headerFile",
2284                    required="true", multiple="true") */
```

2285 /* @Reference(name="referenceName", interfaceHeader="headerFile",
2286 required="true", multiple="true") */

2287 *Snippet A-21: @Reference Annotation Format*

2288 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**.

2289 The multiplicity of the reference is determined from the **required** and **multiple** attributes. If the value of the **multiple** attribute is true, then component type has a reference with a multiplicity of either 0..n or 1..n depending on the value of the **required** attribute – 1..n applies if **required=true**. Otherwise a multiplicity of 0..1 or 1..1 is implied.

2290 **Applies to:** Service

2291 Example:

2292 Implementation:

```
2293        /* @Reference(name="getRate", interfaceHeader="rates.h") */
2294        /*
2295        * @Reference(name="publishRate", interfaceHeader="myRates.h",
2296        *              required="false", multiple="yes") */
```

2297 /* @Reference(name="getRate", interfaceHeader="rates.h") */
2298 /*
2299 * @Reference(name="publishRate", interfaceHeader="myRates.h",
2300 * required="false", multiple="yes") */

2301 /* @Reference(name="getRate", interfaceHeader="rates.h") */
2302 /*
2303 * @Reference(name="publishRate", interfaceHeader="myRates.h",
2304 * required="false", multiple="yes") */

2305 Implementation:

2306 <componentType name="LoanService">

```

2307     <reference name="getRate">
2308         <interface.c header="rates.h">
2309     </reference>
2310     <reference name="publishRate" multiplicity="0..n">
2311         <interface.c header="myRates.h">
2312     </reference>
2313 </componentType>
```

2314 *Snippet A-22: Example of @Reference Annotation*

2315 A.3.4 @Property

2316 Annotation on a service implementation to define a property of the service. Should immediately precede
2317 the variable that the property is based on. The variable declaration is only used for determining the type
2318 of the property. The variable will not be populated with the property value at runtime. Programs use the
2319 SCAProperty<Type>() functions for accessing property data.

2320 **Corresponds to:** *property* element of a *componentType* element.

2321 **Format:**

```

2322 /* @Property(name="propertyName", type="typeName",
2323             default="defaultValue", required="true") */
```

2324 *Snippet A-23: @Property Annotation Format*

2325 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 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**.

2334 **Applies to:** Variable

2335 An SCA implementation MUST ensure that all variables in a component implementation with the same
2336 name and annotated with @Property have the same type. [CA0007]

2337 Example:

2338 Implementation:

```

2339 /* @Property */
2340 long loanType;
```

2342 ComponentType definition:

```

2343 <componentType name="LoanService">
2344   <property name="loanType" type="xsd:int" />
2345 </componentType>
```

2346 *Snippet A-24: Example of @Property Annotation*

2347 A.3.5 @Init

2348 Annotation on a service implementation to indicate a function to be called when the service is
2349 instantiated. If the service is implemented in a program, this annotation indicates the program is to be
2350 called with an initialization flag prior to the first operation.

2351 **Corresponds to:** *@init="true"* attribute of an *implementation.c* element or a *function* child element of an
2352 *implementation.c* element.

2353 **Format:**

2354 /* @Init */

2355 *Snippet A-25: @Init Annotation Format*

2356 The default is **false** (the function is not to be called on service initialization).

2357 **Applies to:** Function or Service

2358 Example:

2359 Implementation:

2360 /* @Init */
2361 void init();

2362

2363 Component definition:

```
2364 <component name="LoanService">  
2365   <implementation.c module="loan" componentType="LoanService">  
2366     <function name="init" init="true" />  
2367   </implementation.c>  
2368 </component>
```

2369 *Snippet A-26: Example of @Init Annotation*

A.3.6 @Destroy

2371 Annotation on a service implementation to indicate a function to be called when the service is terminated.
2372 If the service is implemented in a program, this annotation indicates the program is to be called with a
2373 termination flag after to the final operation.

2374 **Corresponds to:** `@destroy="true"` attribute of an *implementation.c* element or a *function* child element of
2375 an *implementation.c* element.

2376 **Format:**

2377 /* @Destroy */

2378 *Snippet A-27: @Destroy Annotation Format*

2379 The default is **false** (the function is not to be called on service termination).

2380 **Applies to:** Function or Service

2381 Example:

2382 Implementation:

2383 /* @Destroy */
2384 void cleanup();

2385

2386 Component definition:

```
2387 <component name="LoanService">  
2388   <implementation.c module="loan" componentType="LoanService">  
2389     <function name="cleanup" destroy="true" />  
2390   </implementation.c>  
2391 </component>
```

2392 *Snippet A-28: Example of @Destroy Annotation*

A.3.7 @EagerInit

2394 Annotation on a service implementation to indicate the service is to be instantiated when its containing
2395 component is started.

2396 **Corresponds to:** `@eagerInit="true"` attribute of an *implementation.c* element.

2397 **Format:**

2398 `/* @EagerInit */`

2399 *Snippet A-29: @EagerInit Annotation Format*

2400 The default is **false** (the service is initialized lazily).

2401 **Applies to:** Service

2402 Example:

2403 Implementation:

2404 `/* @EagerInit */`

2405

2406 Component definition:

2407 `<component name="LoanService">`

2408 `<implementation.c module="loan" componentType="LoanService"`

2409 `eagerInit="true" />`

2410 `</component>`

2411 *Snippet A-30: Example of @EagerInit Annotation*

A.3.8 @AllowsPassByReference

2413 Annotation on service implementation or operation to indicate that a service or operation allows pass by reference semantics.

2415 **Corresponds to:** `@allowsPassByReference="true"` attribute of an *implementation.c* element or a *function* child element of an *implementation.c* element.

2417 **Format:**

2418 `/* @AllowsPassByReference */`

2419 *Snippet A-31: @AllowsPassByReference Annotation Format*

2420 The default is **false** (the service does not allow by reference parameters).

2421 **Applies to:** Service or Function

2422 Example:

2423 Implementation:

2424 `/* @Service(name="LoanService")`

2425 `* @AllowsPassByReference`

2426 `*/`

2427

2428 Component definition:

2429 `<component name="LoanService">`

2430 `<implementation.c module="loan" componentType="LoanService"`

2431 `allowsPassByReference="true" />`

2432 `</component>`

2433 *Snippet A-32: Example of @AllowsPassByReference Annotation*

A.4 Base Annotation Grammar

2435 While annotations are defined using the `/* ... */` format for comments, if the `// ...` format is supported by a C compiler, the `// ...` format MAY be supported by an SCA implementation annotation processor.

[CA0006]

2438

2439 `<annotation> ::= /* @<baseAnnotation> */`

```
2441 <baseAnnotation> ::= <name> [ (<params>) ]  
2442  
2443 <params> ::= <paramNameValue>[, <paramNameValue>]* |  
2444     <paramValue>[, <paramValue>]*  
2445  
2446 <paramNameValue> ::= <name>=<value>  
2447  
2448 <paramValue> ::= "<value>"  
2449  
2450 <name> ::= NCName  
2451  
2452 <value> ::= string
```

2453 *Snippet A-33: Base Annotation Grammar*

- 2454 • Adjacent string constants are concatenated
- 2455 • NCName is as defined by XML schema **[XSD]**
- 2456 • Whitespace including newlines between tokens is ignored.
- 2457 • Annotations with parameters can span multiple lines within a comment, and are considered complete
2458 when the terminating ")" is reached.

2459 B C SCA Policy Annotations

2460 SCA provides facilities for the attachment of policy-related metadata to SCA assemblies, which influence
2461 how implementations, services and references behave at runtime. The policy facilities are described in
2462 [POLICY]. In particular, the facilities include Intents and Policy Sets, where intents express abstract, high-
2463 level policy requirements and policy sets express low-level detailed concrete policies.

2464 Policy metadata can be added to SCA assemblies through the means of declarative statements placed
2465 into Composite documents and into Component Type documents. These annotations are completely
2466 independent of implementation code, allowing policy to be applied during the assembly and deployment
2467 phases of application development.

2468 However, it can be useful and more natural to attach policy metadata directly to the code of
2469 implementations. This is particularly important where the policies concerned are relied on by the code
2470 itself. An example of this from the Security domain is where the implementation code expects to run
2471 under a specific security Role and where any service operations invoked on the implementation have to
2472 be authorized to ensure that the client has the correct rights to use the operations concerned. By
2473 annotating the code with appropriate policy metadata, the developer can rest assured that this metadata
2474 is not lost or forgotten during the assembly and deployment phases.

2475 The SCA C policy annotations provide the capability for the developer to attach policy information to C
2476 implementation code. The annotations provide both general facilities for attaching SCA Intents and Policy
2477 Sets to C code and annotations for specific policy intents. Policy annotation can be used in files for
2478 service interfaces or component implementations.

2479 B.1 General Intent Annotations

2480 SCA provides the annotation **@Requires** for the attachment of any intent to a C function, to a C function
2481 declaration or to sets of functions implementing a service or sets of function declarations defining a
2482 service interface.

2483 The **@Requires** annotation can attach one or multiple intents in a single statement. Each intent is
2484 expressed as a string. Intents are XML QNames, which consist of a Namespace URI followed by the
2485 name of the Intent. The precise form used is:

```
2486     "{" + Namespace URI + "}" + intentname
```

2487 *Snippet B-1: Intent Format*

2488

2489 Intents can be qualified, in which case the string consists of the base intent name, followed by a ".",
2490 followed by the name of the qualifier. There can also be multiple levels of qualification.

2491 This representation is quite verbose, so we expect that reusable constants will be defined for the
2492 namespace part of this string, as well as for each intent that is used by C code. SCA defines constants for
2493 intents such as the following:

```
2494
2495     /* @Define SCA_PREFIX "{http://docs.oasis-pen.org/ns/opencsa/sca/200912}"
2496     */
2497     /* @Define CONFIDENTIALITY SCA PREFIX ## "confidentiality" */
2498     /* @Define CONFIDENTIALITY_MESSAGE CONFIDENTIALITY ## ".message" */
```

2499 *Snippet B-2: Example Intent Constants*

2500

2501 Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,
2502 separated by an underscore. These intent constants are defined in the file that defines an annotation for
2503 the intent (annotations for intents, and the formal definition of these constants, are covered in a following
2504 section).

2505 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.
2506 **Corresponds to:** @requires attribute of an *interface.c*, *implementation.c*, *function* or *callbackFunction*
2507 element.

2508 **Format:**

```
2509        /* @Requires("qualifiedIntent" | {"qualifiedIntent" [, "qualifiedIntent"]})) */
```

2510 where

```
2511        qualifiedIntent ::= QName | QName.qualifier | QName.qualifier1.qualifier2
```

2512 *Snippet B-3: @Requires Annotation Format*

2513 **Applies to:** Interface, Service, Function, Function Prototype

2514 Examples:

2515 Attaching the intents "confidentiality.message" and "integrity.message".

```
2516        /* @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE}) */
```

2517 *Snippet B-4: Example @Requires Annotation*

2518 A reference requiring support for confidentiality:

```
2519        /* @Requires(CONFIDENTIALITY)
2520           * @Reference(interfaceHeader="SetBar.h") */
2521           void setBar(struct barType *bar);
```

2522 *Snippet B-5: @Requires Annotation applied with an @Reference Annotation*

2523

2524 Users can also choose to only use constants for the namespace part of the QName, so that they can add
2525 new intents without having to define new constants. In that case, this definition would instead look like
2526 this:

2527

```
2528        /* @Requires(SCA_PREFIX "confidentiality")
2529           * @Reference(interfaceHeader="SetBar.h") */
2530           void setBar(struct barType *bar);
```

2531 *Snippet B-6: @Requires Annotation Using Mixed Constants and Literals*

2532 **B.2 Specific Intent Annotations**

2533 In addition to the general intent annotation supplied by the @Requires annotation described above, there
2534 are C annotations that correspond to specific policy intents.

2535 The general form of these specific intent annotations is an annotation with a name derived from the name
2536 of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation
2537 in the form of a string or an array of strings.

2538 For example, the SCA confidentiality intent described in General Intent Annotations using the
2539 @Requires(CONFIDENTIALITY) intent can also be specified with the specific @Confidentiality intent
2540 annotation. The specific intent annotation for the "integrity" security intent is:

2541

```
2542        /* @Integrity */
```

2543 *Snippet B-7: Example Specific Intent Annotation*

2544

2545 **Corresponds to:** @requires=<Intent>" attribute of an *interface.c*, *implementation.c*, *function* or
2546 *callbackFunction* element.

2547 **Format:**

```
2548        /* @<Intent>[(qualifiers)] */
```

2549 where Intent is an NCName that denotes a particular type of intent.

```
2550     Intent ::= NCName
2551     qualifiers ::= "qualifier" | {"qualifier" [, "qualifier"] }
2552     qualifier ::= NCName | NCName/qualifier
```

2553 *Snippet B-8: @<Intent> Annotation Format*

2554 **Applies to:** Interface, Service, Function, Function Prototype – but see specific intents for restrictions

2555 Example:

```
2556     /* @ClientAuthentication( {"message", "transport"} ) */
```

2557 *Snippet B-9 Example @<Intent> Annotation*

2558

2559 This annotation attaches the pair of qualified intents: *authentication.message* and *authentication.transport*
2560 (the sca: namespace is assumed in both of these cases – "http://docs.oasis-
2561 open.org/ns/opencsa/sca/200912").

2562 The Policy Framework **[POLICY]** defines a number of intents and qualifiers. Security Interaction –
2563 Miscellaneous define the annotations for those intents.

2564 **B.2.1 Security Interaction**

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

2565 *Table B-1: Security Interaction Intent Annotations*

2566

2567 These three intents can be qualified with

- transport
- message

2570 **B.2.2 Security Implementation**

Intent	Annotation	Qualifiers
authorization	@Authorization	fine_grain

2571 *Table B-2: Security Implementation Intent Annotations*

2572 **B.2.3 Reliable Messaging**

Intent	Annotation
atLeastOnce	@AtLeastOnce
atMostOnce	@AtMostOnce
ordered	@Ordered

exactlyOnce	@ExactlyOnce
-------------	--------------

2573 *Table B-3: Reliable Messaging Intent Annotations*

2574 B.2.4 Transactions

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

2575 *Table B-4: Transaction Intent Annotations*

2576 B.2.5 Miscellaneous

Intent	Annotation	Qualifiers
SOAP	@SOAP	v1_1 v1_2

2577 *Table B-5: Miscellaneous Intent Annotations*

2578 B.3 Policy Set Annotations

2579 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies (for example,
2580 a concrete policy is the specific encryption algorithm to use when encrypting messages when using a
2581 specific communication protocol to link a reference to a service).

2582 Policy Sets can be applied directly to C implementations using the **@PolicySets** annotation. The
2583 PolicySets annotation either takes the QName of a single policy set as a string or the name of two or
2584 more policy sets as an array of strings.

2585 **Corresponds to:** @policySets attribute of an *interface.c*, *implementation.c*, *function* or *callbackFunction*
2586 element.

2587 **Format:**

```
2588 /* @PolicySets( "<policy set QName>" |
2589   { "<policy set QName>" [, "<policy set QName>"] }) */
```

2590 *Snippet B-10: @PolicySets Annotation Format*

2591 As for intents, PolicySet names are QNames – in the form of “{Namespace-URI}localPart”.

2592 **Applies to:** Interface, Service, Function, Function Prototype

2593 **Example:**

```
2594 /* @Reference(name="helloService", interfaceHeader="helloService.h",
2595   *           required=true)
2596   * @PolicySets({ MY_NS "WS_Encryption_Policy",
2597   *               MY_NS "WS_Authentication_Policy" }) */
2598 HelloService* helloService;
2599 ...
```

```

2600      }
2601 Snippet B-11: Example @PolicySets Annotation
2602
2603 In this case, the Policy Sets WS_Encryption_Policy and WS_Authentication_Policy are applied, both
2604 using the namespace defined for the constant MY_NS.
2605 PolicySets satisfy intents expressed for the implementation when both are present, according to the rules
2606 defined in [POLICY].

```

2607 B.4 Policy Annotation Grammar Additions

```

2608 <annotation> ::= /* @<baseAnnotation> | @<requiresAnnotation> |
2609           @<intentAnnotation> | @<policySetAnnotation> */
2610
2611 <requiresAnnotation> ::= Requires(<intents>)
2612
2613 <intents> ::= "<qualifiedIntent>" |
2614           {"<qualifiedIntent>"[, "<qualifiedIntent>"]*})
2615
2616 <qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |
2617           <intentName>.<qualifier>.qualifier>
2618
2619 <intentName> ::= {anyURI}NCName
2620
2621 <intentAnnotation> ::= <intent>[(<qualifiers>)]
2622
2623 <intent> ::= NCName[ (param) ]
2624
2625 <qualifiers> ::= "<qualifier>" | {"<qualifier>"[, "<qualifier>"]*}
2626
2627 <qualifier> ::= NCName | NCName/<qualifier>
2628
2629 <policySetAnnotation> ::= policySets(<policysets>)
2630
2631 <policySets> ::= "<policySetName>" | {"<policySetName>"[, "<policySetName>"]*}
2632
2633 <policySetName> ::= {anyURI}NCName

```

2634 Snippet B-12: Annotation Grammar Additions for Policy Annotations

- 2635 • anyURI is as defined by XML schema [XSD]

2636 B.5 Annotation Constants

```

2637 <annotationConstant> ::= /* @Define <identifier> <token string> */
2638
2639 <identifier> ::= token
2640
2641 <token string> ::= "string" | "string"[ ## <token string>]

```

2642 Snippet B-13: Annotation Constants Grammar

- 2643 • Constants are immediately expanded

2644 C C WSDL Annotations

2645 To allow developers to control the mapping of C to WSDL, a set of annotations is defined. If WSDL
2646 mapping annotations are supported by an implementation, the annotations defined here MUST be
2647 supported and MUST be mapped to WSDL as described. [CC0005]

2648 C.1 Interface Header Annotations

2649 C.1.1 @WebService

2650 Annotation on a C header file indicating that it represents a web service. A second or subsequent
2651 instance of this annotation in a file, or a first instance after any function declarations indicates the start of
2652 a new service and has to contain a name value. An SCA implementation MUST treat any instance of a
2653 @Remotable annotation and without an explicit @WebService annotation as if a @WebService
2654 annotation with a name value equal to the name value of the @Interface annotation, if specified, and no
2655 other parameters was specified. [CC0001]

2656 **Corresponds to:** javax.jws.WebService annotation in the JAX-WS specification (7.11.1)

2657 **Format:**

```
2658 /* @WebService(name="portTypeName", targetNamespace="namespaceURI",  
2659 * serviceNamespace="WSDLServiceName", portName="WSDLPortName") */
```

2660 *Snippet C-1: @WebService Annotation Format*

2661 where

- **name : NCName (0..1)** – specifies the name of the web service portType. The default is the root name of the header file containing the annotation. The name of the associated binding is also determined by the portType. The binding name is the name of the portType suffixed with “Binding”.
- **targetNamespace : anyURI (0..1)** – specifies the target namespace for the web service. The default namespace is determined by the implementation.
- **serviceName : NCName (0..1)** – specifies the name for the associated WSDL service. The default service name is the name of the header file containing the annotation suffixed with “Service”.
- **portName : NCName (0..1)** – specifies the name for the associated WSDL port for the service. If portName is not specified, the name of the WSDL port is the name of the portType suffixed with “Port”. See [CF0032]

2672 **Applies to:** Header file

2673 Example:

2674 Input C header file (stockQuote.h):

```
2675 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
2676 * serviceNamespace="StockQuoteService") */  
2677 ...
```

2679

2680 Generated WSDL file:

```
2681 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
2682   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
2683   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
2684   xmlns:tns="http://www.example.org/"  
2685   targetNamespace="http://www.example.org/">  
2686  
2687   <portType name="StockQuote">  
2688     <sca-c:bindings>
```

```

2689         <sca-c:prefix name="stockQuote"/>
2690     </sca-c:bindings>
2691 </portType>
2692
2693     <binding name="StockQuoteServiceSoapBinding">
2694         <soap:binding style="document"
2695             transport="http://schemas.xmlsoap.org/soap/http"/>
2696     </binding>
2697
2698     <service name="StockQuoteService">
2699         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2700             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2701         </port>
2702     </service>
2703 </definitions>

```

2704 *Snippet C-2: Example @WebService Annotation*

2705 C.1.2 @WebFunction

2706 Annotation on a C function indicating that it represents a web service operation. An SCA implementation
2707 **MUST** treat a function annotated with an `@Function` annotation and without an explicit `@WebFunction`
2708 annotation as if a `@WebFunction` annotation with an `operationName` value equal to the name value
2709 of the `@Function` annotation, an `exclude` value equal to the `exclude` value of the `@Function` annotation
2710 and no other parameters was specified. [CC0002]

2711 **Corresponds to:** `javax.jws.WebMethod` annotation in the JAX-WS specification (7.11.2)

2712 **Format:**

```

2713 /* @WebFunction(operationName="operation",    action="SOAPAction",
2714 *                  exclude="false") */

```

2715 *Snippet C-3: @WebFunction Annotation Format*

2716 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`”.

2724 **Applies to:** Function.

2725 Example:

2726 Input C header file:

```

2727 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2728 *                  serviceName="StockQuoteService") */
2729
2730 /* @WebFunction(operationName="GetLastTradePrice",
2731 *                  action="urn:GetLastTradePrice") */
2732 float getLastTradePrice(const char *tickerSymbol);
2733
2734 /* @WebFunction(exclude="true") */
2735 void setLastTradePrice(const char *tickerSymbol, float value);

```

2736

2737 Generated WSDL file:

```

2738 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2739   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2740   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"〉

```

```

2741      xmlns:tns="http://www.example.org/"
2742      targetNamespace="http://www.example.org/">
2743
2744      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2745          xmlns:tns="http://www.example.org/"
2746          attributeFormDefault="unqualified"
2747          elementFormDefault="unqualified"
2748          targetNamespace="http://www.example.org/">
2749          <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2750          <xs:element name="GetLastTradePriceResponse"
2751              type="tns:GetLastTradePriceResponse"/>
2752          <xs:complexType name="GetLastTradePrice">
2753              <xs:sequence>
2754                  <xs:element name="tickerSymbol" type="xs:string"/>
2755              </xs:sequence>
2756          </xs:complexType>
2757          <xs:complexType name="GetLastTradePriceResponse">
2758              <xs:sequence>
2759                  <xs:element name="return" type="xs:float"/>
2760              </xs:sequence>
2761          </xs:complexType>
2762      </xs:schema>
2763
2764      <message name="GetLastTradePrice">
2765          <part name="parameters" element="tns:GetLastTradePrice">
2766          </part>
2767      </message>
2768
2769      <message name="GetLastTradePriceResponse">
2770          <part name="parameters" element="tns:GetLastTradePriceResponse">
2771          </part>
2772      </message>
2773
2774      <portType name="StockQuote">
2775          <sca-c:bindings>
2776              <sca-c:prefix name="stockQuote"/>
2777          </sca-c:bindings>
2778          <operation name="GetLastTradePrice">
2779              <sca-c:bindings>
2780                  <sca-c:function name="getLastTradePrice"/>
2781              </sca-c:bindings>
2782              <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2783              </input>
2784              <output name="GetLastTradePriceResponse"
2785                  message="tns:GetLastTradePriceResponse">
2786              </output>
2787          </operation>
2788      </portType>
2789
2790      <binding name="StockQuoteServiceSoapBinding">
2791          <soap:binding style="document"
2792              transport="http://schemas.xmlsoap.org/soap/http"/>
2793          <operation name="GetLastTradePrice">
2794              <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2795              <input name="GetLastTradePrice">
2796                  <soap:body use="literal"/>
2797              </input>
2798              <output name="GetLastTradePriceResponse">
2799                  <soap:body use="literal"/>
2800              </output>
2801          </operation>
2802      </binding>
2803
2804      <service name="StockQuoteService">

```

```
2805     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2806         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2807     </port>
2808 </service>
2809 </definitions>
```

2810 *Snippet C-4: Example @WebFunction Annotation*

2811 **C.1.3 @WebOperation**

2812 Annotation on a C request message struct indicating that it represents a web service operation. An SCA
2813 implementation MUST treat a struct annotated with an @Operation annotation without an explicit
2814 @WebOperation annotation as if a @WebOperation annotation with an operationName value equal
2815 to the name value of the @Operation annotation, a response value equal to the response value of the
2816 @Operation annotation, an exclude value equal to the exclude value of the @Operation annotation and
2817 no other parameters was specified. [CC0003]

2818 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2819 **Format:**

```
2820 /* @WebOperation(operationName="operation", response="responseStruct",
2821 *                  action="SOAPAction", exclude="false") */
```

2822 *Snippet C-5: @WebOperation Annotation Format*

2823 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”.

2833 **Applies to:** Struct.

2834 Example:

2835 Input C header file:

```
2836 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2837 *                  serviceName="StockQuoteService") */
2838
2839 /* @WebOperation(operationName="GetLastTradePrice",
2840 *                  response="getLastTradePriceResponseMsg"
2841 *                  action="urn:GetLastTradePrice") */
2842 struct getLastTradePriceMsg {
2843     char tickerSymbol[10];
2844 } getLastTradePrice;
2845
2846 struct getLastTradePriceResponseMsg {
2847     float return;
2848 } getLastTradePriceResponse;
```

2849

2850 Generated WSDL file:

```
2851 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2852   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2853   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
2854   xmlns:tns="http://www.example.org/"
2855   targetNamespace="http://www.example.org/">
```

```

2856
2857     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2858         xmlns:tns="http://www.example.org/"
2859         attributeFormDefault="unqualified"
2860         elementFormDefault="unqualified"
2861         targetNamespace="http://www.example.org/">
2862     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2863     <xs:element name="GetLastTradePriceResponse"
2864         type="tns:GetLastTradePriceResponse"/>
2865     <xs:simpleType name="TickerSymbolType">
2866         <xs:restriction base="xs:string">
2867             <xsd:maxlength value="9"/>
2868         </xs:restriction>
2869     </xs:simpleType>
2870     <xs:complexType name="GetLastTradePrice">
2871         <xs:sequence>
2872             <xs:element name="tickerSymbol" type="TickerSymbolType"/>
2873         </xs:sequence>
2874     </xs:complexType>
2875     <xs:complexType name="GetLastTradePriceResponse">
2876         <xs:sequence>
2877             <xs:element name="return" type="xs:float"/>
2878         </xs:sequence>
2879     </xs:complexType>
2880 </xs:schema>
2881
2882 <message name="GetLastTradePrice">
2883     <sca-c:bindings>
2884         <sca-c:struct name="getLastTradePrice"/>
2885     </sca-c:bindings>
2886     <part name="parameters" element="tns:GetLastTradePrice">
2887     </part>
2888 </message>
2889
2890 <message name="GetLastTradePriceResponse">
2891     <sca-c:bindings>
2892         <sca-c:struct name="getLastTradePriceResponse"/>
2893     </sca-c:bindings>
2894     <part name="parameters" element="tns:GetLastTradePriceResponse">
2895     </part>
2896 </message>
2897
2898 <portType name="StockQuote">
2899     <sca-c:bindings>
2900         <sca-c:prefix name="stockQuote"/>
2901     </sca-c:bindings>
2902     <operation name="GetLastTradePrice">
2903         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2904         </input>
2905         <output name="GetLastTradePriceResponse"
2906             message="tns:GetLastTradePriceResponse">
2907             </output>
2908         </operation>
2909     </portType>
2910
2911 <binding name="StockQuoteServiceSoapBinding">
2912     <soap:binding style="document"
2913         transport="http://schemas.xmlsoap.org/soap/http"/>
2914     <operation name="GetLastTradePrice">
2915         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2916         <input name="GetLastTradePrice">
2917             <soap:body use="literal"/>
2918         </input>
2919         <output name="GetLastTradePriceResponse">
```

```

2920             <soap:body use="literal"/>
2921         </output>
2922     </operation>
2923 </binding>
2924
2925     <service name="StockQuoteService">
2926         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2927             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2928         </port>
2929     </service>
2930 </definitions>
```

2931 *Snippet C-6: Example @WebOperation Annotation*

2932 C.1.4 @OneWay

2933 Annotation on a C function indicating that it represents a one-way request. The @OneWay annotation
 2934 also affects the service interface. See @OneWay.

2935 **Corresponds to:** javax.jws.OneWay annotation in the JAX-WS specification (7.11.3)

2936 **Format:**

```
2937 /* @OneWay */
```

2938 *Snippet C-7: @OneWay Annotation Format*

2939 **Applies to:** Function.

2940 Example:

2941 Input C header file:

```

2942 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2943 *                 serviceName="StockQuoteService") */
2944
2945 /* @WebFunction(operationName="SetTradePrice",
2946 *                 action="urn:SetTradePrice")
2947 * @OneWay */
2948 void setTradePrice(const char *tickerSymbol, float price);
```

2949

2950 Generated WSDL file:

```

2951 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2952   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2953   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
2954   xmlns:tns="http://www.example.org/"
2955   targetNamespace="http://www.example.org/">
2956
2957   <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2958     xmlns:tns="http://www.example.org/"
2959     attributeFormDefault="unqualified"
2960     elementFormDefault="unqualified"
2961     targetNamespace="http://www.example.org/">
2962     <xselement name="SetTradePrice" type="tns:SetTradePrice"/>
2963     <xsccomplexType name="SetTradePrice">
2964       <xsssequence>
2965         <xselement name="tickerSymbol" type="xs:string"/>
2966         <xselement name="price" type="xs:float"/>
2967       </xsssequence>
2968     </xsccomplexType>
2969   </xsschema>
2970
2971   <message name="SetTradePrice">
2972     <part name="parameters" element="tns:SetTradePrice">
2973     </part>
2974   </message>
```

```

2975
2976     <portType name="StockQuote">
2977         <sca-c:bindings>
2978             <sca-c:prefix name="stockQuote"/>
2979         </sca-c:bindings>
2980         <operation name="SettTradePrice">
2981             <sca-c:bindings>
2982                 <sca-c:function name="setTradePrice"/>
2983             </sca-c:bindings>
2984             <input name="SetTradePrice" message="tns:SetTradePrice">
2985                 </input>
2986             </operation>
2987         </portType>
2988
2989     <binding name="StockQuoteServiceSoapBinding">
2990         <soap:binding style="document"
2991             transport="http://schemas.xmlsoap.org/soap/http"/>
2992         <operation name="SetTradePrice">
2993             <soap:operation soapAction="urn:SetTradePrice" style="document"/>
2994             <input name="SetTradePrice">
2995                 <soap:body use="literal"/>
2996             </input>
2997         </operation>
2998     </binding>
2999
3000     <service name="StockQuoteService">
3001         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3002             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3003         </port>
3004     </service>
3005 </definitions>

```

3006 *Snippet C-8: Example @OneWay Annotation*

3007 C.1.5 @WebParam

3008 Annotation on a C function indicating the mapping of a parameter to the associated input and output
 3009 WSDL messages. Or on a C struct indicating the mapping of a member to the associated WSDL
 3010 message.

3011 **Corresponds to:** javax.jws.WebParam annotation in the JAX-WS specification (7.11.4)

3012 **Format:**

```

3013     /* @WebParam(paramName="parameter", name="WSDLElement",
3014     *             targetNamespace="namespaceURI", mode="IN" | "OUT" | "INOUT",
3015     *             header="false", partName="WSDLPart", type="xsdType") */

```

3016 *Snippet C-9: @WebParam Annotation Format*

3017 where:

- **paramName : NCName (1..1)** – specifies the name of the parameter that this annotation applies to. The value of the paramName of a @WebParam annotation MUST be the name of a parameter of the function the annotation is applied to. [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 : QName (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 either one of the simpleTypes defined in namespace <http://www.w3.org/2001/XMLSchema> or, if the type of the parameter is a struct, the QName of a XSD complex type following the mapping specified in Complex Content Binding. [CC0006] The default type is determined by the mapping defined in Data Binding.

Applies to: Function parameter or struct member.

Example:

Input C header file:

```
/* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
 *             serviceName="StockQuoteService") */  
  
/* @WebFunction(operationName="GetLastTradePrice",  
 *               action="urn:GetLastTradePrice")  
 * @WebParam(paramName="tickerSymbol", name="symbol", mode="IN") */  
float getLastTradePrice(char *tickerSymbol);
```

Generated WSDL file:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
    xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
    xmlns:tns="http://www.example.org/"  
    targetNamespace="http://www.example.org/">  
  
    <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"  
        xmlns:tns="http://www.example.org/"  
        attributeFormDefault="unqualified"  
        elementFormDefault="unqualified"  
        targetNamespace="http://www.example.org/">  
        <xss:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
        <xss:element name="GetLastTradePriceResponse"  
            type="tns:GetLastTradePriceResponse"/>  
        <xss:complexType name="GetLastTradePrice">  
            <xss:sequence>  
                <xss:element name="symbol" type="xs:string"/>  
            </xss:sequence>  
        </xss:complexType>  
        <xss:complexType name="GetLastTradePriceResponse">  
            <xss:sequence>  
                <xss:element name="return" type="xs:float"/>  
            </xss:sequence>  
        </xss:complexType>  
    </xsschema>  
  
    <message name="GetLastTradePrice">  
        <part name="parameters" element="tns:GetLastTradePrice">  
        </part>  
    </message>  
  
    <message name="GetLastTradePriceResponse">  
        <part name="parameters" element="tns:GetLastTradePriceResponse">
```

```

3087     </part>
3088   </message>
3089
3090   <portType name="StockQuote">
3091     <sca-c:bindings>
3092       <sca-c:prefix name="stockQuote"/>
3093     </sca-c:bindings>
3094     <operation name="GetLastTradePrice">
3095       <sca-c:bindings>
3096         <sca-c:function name="getLastTradePrice"/>
3097         <sca-c:parameter name="tickerSymbol"
3098           part="tns:GetLastTradePrice/parameter"
3099           childElementName="symbol"/>
3100       </sca-c:bindings>
3101     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3102     </input>
3103     <output name="GetLastTradePriceResponse"
3104       message="tns:GetLastTradePriceResponse">
3105     </output>
3106   </operation>
3107 </portType>
3108
3109 <binding name="StockQuoteServiceSoapBinding">
3110   <soap:binding style="document"
3111     transport="http://schemas.xmlsoap.org/soap/http"/>
3112   <operation name="GetLastTradePrice">
3113     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3114     <input name="GetLastTradePrice">
3115       <soap:body use="literal"/>
3116     </input>
3117     <output name="GetLastTradePriceResponse">
3118       <soap:body use="literal"/>
3119     </output>
3120   </operation>
3121 </binding>
3122
3123 <service name="StockQuoteService">
3124   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3125     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3126   </port>
3127 </service>
3128 </definitions>

```

3129 *Snippet C-10: Example @WebParam Annotation*

3130 **C.1.6 @WebResult**

3131 Annotation on a C function indicating the mapping of the function's return type to the associated output
 3132 WSDL message.

3133 **Corresponds to:** javax.jws.WebResult annotation in the JAX-WS specification (7.11.5)

3134 **Format:**

```

3135 /* @WebResult(name="WSDLElement", targetNamespace="namespaceURI",
3136   *             header="false", partName="WSDLPart", type="xsdType") */

```

3137 *Snippet C-11: @WebResult Annotation Format*

3138 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

- 3143 ignored unless the binding style is document, and the binding parameterStyle is bare. (See
 3144 @SOAPBinding).
- 3145 • **header : boolean (0..1)** – specifies whether the result is associated with a SOAP header element.
 3146 The default value is “false”.
 - 3147 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The
 3148 default value is the value of name.
 - 3149 • **type : NCName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with
 3150 this parameter. The value of the type property of a @WebResult annotation MUST be one of the
 3151 simpleTypes defined in namespace <http://www.w3.org/2001/XMLSchema>. [CC0007] The default type
 3152 is determined by the mapping defined in 11.3.1.

3153 **Applies to:** Function.

3154 Example:

3155 Input C header file:

```
3156 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  

3157 *                    serviceName="StockQuoteService") */  

3158  

3159 /* @WebFunction(operationName="GetLastTradePrice",  

3160 *                    action="urn:GetLastTradePrice")  

3161 * @WebResult(name="price") */  

3162 float getLastTradePrice(const char *tickerSymbol);
```

3163

3164 Generated WSDL file:

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

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

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

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

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

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

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

  attributeFormDefault="unqualified"  

  elementFormDefault="unqualified"  

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

<xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  

<xs:element name="GetLastTradePriceResponse"  

  type="tns:GetLastTradePriceResponse"/>  

<xs:complexType name="GetLastTradePrice">  

  <xs:sequence>  

    <xs:element name="tickerSymbol" type="xs:string"/>  

  </xs:sequence>  

</xs:complexType>  

<xs:complexType name="GetLastTradePriceResponse">  

  <xs:sequence>  

    <xs:element name="price" type="xs:float"/>  

  </xs:sequence>  

</xs:complexType>  

</xs:schema>  

<message name="GetLastTradePrice">  

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

  </part>  

</message>  

<message name="GetLastTradePriceResponse">  

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

  </part>  

</message>
```

```

3201 <portType name="StockQuote">
3202     <sca-c:bindings>
3203         <sca-c:prefix name="stockQuote"/>
3204     </sca-c:bindings>
3205     <operation name="GetLastTradePrice">
3206         <sca-c:bindings>
3207             <sca-c:function name="getLastTradePrice"/>
3208         </sca-c:bindings>
3209         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3210             </input>
3211         <output name="GetLastTradePriceResponse"
3212             message="tns:GetLastTradePriceResponse">
3213             </output>
3214         </operation>
3215     </portType>
3216
3217     <binding name="StockQuoteServiceSoapBinding">
3218         <soap:binding style="document"
3219             transport="http://schemas.xmlsoap.org/soap/http"/>
3220         <operation name="GetLastTradePrice">
3221             <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3222             <input name="GetLastTradePrice">
3223                 <soap:body use="literal"/>
3224             </input>
3225             <output name="GetLastTradePriceResponse">
3226                 <soap:body use="literal"/>
3227             </output>
3228         </operation>
3229     </binding>
3230
3231     <service name="StockQuoteService">
3232         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3233             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3234         </port>
3235     </service>
3236 </definitions>

```

3237 *Snippet C-12: Example @WebResult Annotation*

3238 C.1.7 @SOAPBinding

3239 Annotation on a C WebService or function specifying the mapping of the web service onto the SOAP
3240 message protocol.

3241 **Corresponds to:** javax.jws.SOAPBinding annotation in the JAX-WS specification (7.11.6)

3242 **Format:**

```

3243 /* @SOAPBinding(style="DOCUMENT" | "RPC", use="LITERAL" | "ENCODED",
3244 *                  parameterStyle="BARE" | "WRAPPED") */

```

3245 *Snippet C-13: @SOAPBinding Annotation Format*

3246 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”.

3251 **Applies to:** WebService, Function.

3252 Example:

3253 Input C header file:

```

3254       /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
```

```

3255             serviceName="StockQuoteService") */
3256             * @SOAPBinding(style="RPC") */
3257 ...
3258 ...
3259
3260     Generated WSDL file:
3261 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3262   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3263   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3264   xmlns:tns="http://www.example.org/"
3265   targetNamespace="http://www.example.org/">
3266
3267     <portType name="StockQuote">
3268       <sca-c:bindings>
3269         <sca-c:prefix name="stockQuote"/>
3270       </sca-c:bindings>
3271     </portType>
3272
3273     <binding name="StockQuoteServiceSoapBinding">
3274       <soap:binding style="rpc"
3275         transport="http://schemas.xmlsoap.org/soap/http"/>
3276     </binding>
3277
3278     <service name="StockQuoteService">
3279       <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3280         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3281       </port>
3282     </service>
3283   </definitions>

```

3284 *Snippet C-14: Example @SOAPBinding Annotation*

3285 C.1.8 @WebFault

3286 Annotation on a C struct indicating that it format of a fault message.

3287 **Corresponds to:** javax.xml.ws.WebFault annotation in the JAX-WS specification (7.2)

3288 **Format:**

```

3289 /* @WebFault(name="WSDLElement", targetNamespace="namespaceURI") */

```

3290 *Snippet C-15: @WebFault Annotation Format*

3291 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.

3295 **Applies to:** struct.

3296 **Example:**

3297 Input C header file:

```

3298 /* @WebFault(name="UnknownSymbolFault",
3299   *           targetNamespace="http://www.example.org/")
3300 struct UnkSymMsg {
3301   char faultInfo[10];
3302 } unkSymInfo;
3303
3304 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3305   *               serviceName="StockQuoteService") */
3306
3307 /* @WebFunction(operationName="GetLastTradePrice",

```

```

3308         * action="urn:GetLastTradePrice")
3309         * @WebThrows(faults="unkSymMsg") */
3310 float getLastTradePrice(const char *tickerSymbol);

```

3311

3312 Generated WSDL file:

```

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

  <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.example.org/"
    attributeFormDefault="unqualified"
    elementFormDefault="unqualified"
    targetNamespace="http://www.example.org/">
    <xselement name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
    <xselement name="GetLastTradePriceResponse"
      type="tns:GetLastTradePriceResponse"/>
    <xsccomplexType name="GetLastTradePrice">
      <xsssequence>
        <xselement name="tickerSymbol" type="xs:string"/>
      </xsssequence>
    </xsccomplexType>
    <xsccomplexType name="GetLastTradePriceResponse">
      <xsssequence>
        <xselement name="return" type="xs:float"/>
      </xsssequence>
    </xsccomplexType>
    <xssimpleType name="UnknownSymbolFaultType">
      <xsrrestriction base="xs:string">
        <xsd:maxlength value="9"/>
      </xsrrestriction>
    </xssimpleType>
    <xselement name="UnknownSymbolFault" type="UnknownSymbolFaultType"/>
  </xsschema>

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

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

  <message name="UnknownSymbol">
    <sca-c:bindings>
      <sca-c:struct name="unkSymMsg"/>
    </sca-c:bindings>
    <part name="parameters" element="tns:UnknownSymbolFault">
    </part>
  </message>

  <portType name="StockQuote">
    <sca-c:bindings>
      <sca-c:prefix name="stockQuote"/>
    </sca-c:bindings>
    <operation name="GetLastTradePrice">
      <sca-c:bindings>
        <sca-c:function name="getLastTradePrice"/>
      </sca-c:bindings>
    </operation>
  </portType>

```

```

3371         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3372             </input>
3373             <output name="GetLastTradePriceResponse"
3374                 message="tns:GetLastTradePriceResponse">
3375                 </output>
3376                 <fault name="UnknownSymbol" message="tns:UnknownSymbol">
3377                     </fault>
3378             </operation>
3379         </portType>
3380
3381     <binding name="StockQuoteServiceSoapBinding">
3382         <soap:binding style="document"
3383             transport="http://schemas.xmlsoap.org/soap/http"/>
3384         <operation name="GetLastTradePrice">
3385             <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3386             <input name="GetLastTradePrice">
3387                 <soap:body use="literal"/>
3388             </input>
3389             <output name="GetLastTradePriceResponse">
3390                 <soap:body use="literal"/>
3391             </output>
3392             <fault>
3393                 <soap:fault name="UnknownSymbol" use="literal"/>
3394             </fault>
3395         </operation>
3396     </binding>
3397
3398     <service name="StockQuoteService">
3399         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3400             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3401         </port>
3402     </service>
3403 </definitions>

```

3404 *Snippet C-16: Example @WebFault Annotation*

3405 **C.1.9 @WebThrows**

3406 Annotation on a C function or operation indicating which faults might be thrown by this function or
 3407 operation.

3408 **Corresponds to:** No equivalent in JAX-WS.

3409 **Format:**

```
3410 /* @WebThrows(faults="faultMsg1"[, "faultMsgn"]*) */
```

3411 *Snippet C-17: @WebThrows Annotation Format*

3412 where:

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

3416 **Applies to:** Function or Operation

3417 Example:

3418 See @WebFault.

3419 **D C WSDL Mapping Extensions**

3420 The following WSDL extensions are used to augment the conversion process from WSDL to C. All of
3421 these extensions are defined in the namespace <http://docs.oasis-open.org/ns/opencsa/sca-c->
3422 <http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901>. For brevity, all definitions of these extensions will be fully qualified, and all references to
3423 the “sca-c” prefix are associated with the namespace above. If WSDL extensions are supported by an
3424 implementation, all the extensions defined here MUST be supported and MUST be mapped to C as
3425 described. [CD0001]

3426 **D.1 <sca-c:bindings>**

3427 <sca-c:bindings> is a container type which can be used as a WSDL extension. All other SCA wsdl
3428 extensions will be specified as children of a <sca-c:bindings> element. An <sca-c:bindings> element can
3429 be used as an extension to any WSDL type that accepts extensions.

3430 **D.2 <sca-c:prefix>**

3431 <sca-c:prefix> provides a mechanism for defining an alternate prefix for the functions or structs
3432 implementing the operations of a portType.

3433 **Format:**

3434

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

3435 *Snippet D-1: <sca-c:prefix> Element Format*

3436 where:

- 3437 • **prefix/@name : string (1..1)** – specifies the string to prepend to an operation name when generating
3438 a C function or structure name.

3439 **Applicable WSDL element(s):**

- 3440 • wsdl:portType

3441 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:prefix> child element. [CD0003]

3442 Example:

3443 Input WSDL file:

```
3444 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3445   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3446   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
3447   xmlns:tns="http://www.example.org/"  
3448   targetNamespace="http://www.example.org/">  
3449  
3450   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
3451     xmlns:tns="http://www.example.org/"  
3452     attributeFormDefault="unqualified"  
3453     elementFormDefault="unqualified"  
3454     targetNamespace="http://www.example.org/">  
3455     <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
3456     <xsd:element name="GetLastTradePriceResponse"  
3457       type="tns:GetLastTradePriceResponse"/>  
3458     <xsd:complexType name="GetLastTradePrice">  
3459       <xsd:sequence>  
3460         <xsd:element name="tickerSymbol" type="xsd:string"/>  
3461       </xsd:sequence>  
3462     </xsd:complexType>  
3463     <xsd:complexType name="GetLastTradePriceResponse">  
3464       <xsd:sequence>  
3465         <xsd:element name="return" type="xsd:float"/>
```

```

3466      </xs:sequence>
3467    </xs:complexType>
3468  </xs:schema>
3469
3470  <message name="GetLastTradePrice">
3471    <part name="parameters" element="tns:GetLastTradePrice">
3472    </part>
3473  </message>
3474
3475  <message name="GetLastTradePriceResponse">
3476    <part name="parameters" element="tns:GetLastTradePriceResponse">
3477    </part>
3478  </message>
3479
3480  <portType name="StockQuote">
3481    <sca-c:bindings>
3482      <sca-c:prefix name="stockQuote"/>
3483    </sca-c:bindings>
3484    <operation name="GetLastTradePrice">
3485      <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3486      </input>
3487      <output name="GetLastTradePriceResponse"
3488        message="tns:GetLastTradePriceResponse">
3489      </output>
3490    </operation>
3491  </portType>
3492
3493  <binding name="StockQuoteServiceSoapBinding">
3494    <soap:binding style="document"
3495      transport="http://schemas.xmlsoap.org/soap/http"/>
3496    <operation name="GetLastTradePrice">
3497      <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3498      <input name="GetLastTradePrice">
3499        <soap:body use="literal"/>
3500      </input>
3501      <output name="GetLastTradePriceResponse">
3502        <soap:body use="literal"/>
3503      </output>
3504    </operation>
3505  </binding>
3506
3507  <service name="StockQuoteService">
3508    <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3509      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3510    </port>
3511  </service>
3512</definitions>

```

3513

3514 Generated C header file:

```

3515 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3516 *               serviceName="StockQuoteService") */
3517
3518 /* @WebFunction(operationName="GetLastTradePrice",
3519 *               action="urn:GetLastTradePrice") */
3520 float stockQuoteGetLastTradePrice(const char *tickerSymbol);

```

3521 *Snippet D-2: Example <sca-c:prefix> Element*

3522 D.3 <sca-c:enableWrapperStyle>

3523 <sca-c:enableWrapperStyle> indicates whether or not the wrapper style for messages is applied, when
 3524 otherwise applicable. If false, the wrapper style will never be applied.

3525 **Format:**

3526 <sca-c:enableWrapperStyle>value</sca-c:enableWrapperStyle>

3527 *Snippet D-3: <sca-c:enableWrapperStyle> Element Format*

3528 where:

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

3529 **Applicable WSDL element(s):**

3530

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

3531 A <sca-c:bindings/> element MUST NOT have more than one < ssc-c:enableWrapperStyle/> child element. [CD0004]

3532 Example:

3533 Input WSDL file:

```

3534 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3535   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3536   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3537   xmlns:tns="http://www.example.org/"
3538   targetNamespace="http://www.example.org/">
3539
3540   <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"
3541     xmlns:tns="http://www.example.org/"
3542     attributeFormDefault="unqualified"
3543     elementFormDefault="unqualified"
3544     targetNamespace="http://www.example.org/">
3545     <xselement name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3546     <xselement name="GetLastTradePriceResponse"
3547       type="tns:GetLastTradePriceResponse"/>
3548     <xsccomplexType name="GetLastTradePrice">
3549       <xsssequence>
3550         <xselement name="tickerSymbol" type="xss:string"/>
3551       </xsssequence>
3552     </xsccomplexType>
3553     <xsccomplexType name="GetLastTradePriceResponse">
3554       <xsssequence>
3555         <xselement name="return" type="xss:float"/>
3556       </xsssequence>
3557     </xsccomplexType>
3558   </xsschema>
3559
3560   <message name="GetLastTradePrice">
3561     <part name="parameters" element="tns:GetLastTradePrice">
3562       </part>
3563     </message>
3564
3565   <message name="GetLastTradePriceResponse">
3566     <part name="parameters" element="tns:GetLastTradePriceResponse">
3567       </part>
3568     </message>
3569
3570   <portType name="StockQuote">
3571     <sca-c:bindings>
3572       <sca-c:prefix name="stockQuote"/>
3573       <sca-c:enableWrapperStyle>false</sca-c:enableWrapperStyle>
3574     </sca-c:bindings>
3575     <operation name="GetLastTradePrice">
3576       <input name="GetLastTradePrice">
3577         <param name="tickerSymbol" type="xss:string"/>
3578       </input>
3579     </operation>
3580   </portType>
3581 
```

```

3582         <sca-c:bindings>
3583             <sca-c:function name="getLastTradePrice"/>
3584         </sca-c:bindings>
3585         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3586             </input>
3587             <output name="GetLastTradePriceResponse"
3588                 message="tns:GetLastTradePriceResponse">
3589             </output>
3590         </operation>
3591     </portType>
3592 </definitions>
```

3593

3594 Generated C header file:

```

3595 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3596 *               serviceName="StockQuoteService") */
3597
3598 /* @WebFunction(operationName="GetLastTradePrice",
3599 *               action="urn:GetLastTradePrice") */
3600 DATAOBJECT getLastTradePrice(DATAOBJECT parameters);
```

3601 *Snippet D-4: Example <sca-c:enableWrapperStyle> Element*

3602 D.4 <sca-c:function>

3603 <sca-c:function> specifies the name of the C function that the associated WSDL operation is associated
 3604 with. If <sca-c:function> is used, the portType prefix, either default or a specified with <sca-c:prefix> is not
 3605 prepended to the function name.

3606 **Format:**

```

3607     <sca-c:function name="myFunction"/>
```

3608 *Snippet D-5: <sca-c:function> Element Format*

3609 where:

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

3612 **Applicable WSDL element(s):**

- wsdl:portType/wsdl:operation

3614 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:function/> child element. [CD0005]

3615 **Example:**

3616 Input WSDL file:

```

3617 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3618   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3619   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3620   xmlns:tns="http://www.example.org/"
3621   targetNamespace="http://www.example.org/">
3622
3623   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3624     xmlns:tns="http://www.example.org/"
3625     attributeFormDefault="unqualified"
3626     elementFormDefault="unqualified"
3627     targetNamespace="http://www.example.org/">
3628     <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3629     <xsd:element name="GetLastTradePriceResponse"
3630       type="tns:GetLastTradePriceResponse"/>
3631     <xsd:complexType name="GetLastTradePrice">
3632       <xsd:sequence>
3633         <xsd:element name="tickerSymbol" type="xsd:string"/>
3634       </xsd:sequence>
```

```

3635      </xs:complexType>
3636      <xs:complexType name="GetLastTradePriceResponse">
3637          <xs:sequence>
3638              <xs:element name="return" type="xs:float"/>
3639          </xs:sequence>
3640      </xs:complexType>
3641  </xs:schema>
3642
3643  <message name="GetLastTradePrice">
3644      <part name="parameters" element="tns:GetLastTradePrice">
3645          </part>
3646  </message>
3647
3648  <message name="GetLastTradePriceResponse">
3649      <part name="parameters" element="tns:GetLastTradePriceResponse">
3650          </part>
3651  </message>
3652
3653  <portType name="StockQuote">
3654      <sca-c:bindings>
3655          <sca-c:prefix name="stockQuote"/>
3656      </sca-c:bindings>
3657      <operation name="GetLastTradePrice">
3658          <sca-c:bindings>
3659              <sca-c:function name="getTradePrice"/>
3660          </sca-c:bindings>
3661          <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3662              </input>
3663          <output name="GetLastTradePriceResponse"
3664              message="tns:GetLastTradePriceResponse">
3665              </output>
3666          </operation>
3667      </portType>
3668  </definitions>

```

3669

3670 Generated C header file:

```

3671  /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3672   *           serviceName="StockQuoteService") */
3673
3674  /* @WebFunction(operationName="GetLastTradePrice",
3675   *           action="urn:GetLastTradePrice") */
3676  float getTradePrice(const wchar_t *tickerSymbol);

```

3677 *Snippet D-6: Example <sca-c:function> Element*

3678 **D.5 <sca-c:struct>**

3679 <sca-c:struct> specifies the name of the C struct that the associated WSDL message is associated with. If
 3680 <sca-c:struct> is used for an operation request or response message, the portType prefix, either default
 3681 or a specified with <sca-c:prefix> is not prepended to the struct name.

3682 **Format:**

```

3683  <sca-c:struct name="myStruct"/>

```

3684 *Snippet D-7: <sca-c:struct> Element Format*

3685 where:

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

3688 **Applicable WSDL element(s):**

- wsdl:message

3690 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:struct/> child element. [CD0006]

3691 Example:

3692 Input WSDL file:

```
3693 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3694   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3695   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
3696   xmlns:tns="http://www.example.org/"  
3697   targetNamespace="http://www.example.org/">  
3698  
3699   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
3700     xmlns:tns="http://www.example.org/"  
3701     attributeFormDefault="unqualified"  
3702     elementFormDefault="unqualified"  
3703     targetNamespace="http://www.example.org/">  
3704     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
3705     <xs:element name="GetLastTradePriceResponse"  
3706       type="tns:GetLastTradePriceResponse"/>  
3707     <xs:complexType name="GetLastTradePrice">  
3708       <xs:sequence>  
3709         <xs:element name="tickerSymbol" type="xs:string"/>  
3710       </xs:sequence>  
3711     </xs:complexType>  
3712     <xs:complexType name="GetLastTradePriceResponse">  
3713       <xs:sequence>  
3714         <xs:element name="return" type="xs:float"/>  
3715       </xs:sequence>  
3716     </xs:complexType>  
3717   </xs:schema>  
3718  
3719   <message name="GetLastTradePrice">  
3720     <sca-c:bindings>  
3721       <sca-c:struct name="getTradePrice"/>  
3722     </sca-c:bindings>  
3723     <part name="parameters" element="tns:GetLastTradePrice">  
3724     </part>  
3725   </message>  
3726  
3727   <message name="GetLastTradePriceResponse">  
3728     <sca-c:bindings>  
3729       <sca-c:struct name="getTradePriceResponse"/>  
3730     </sca-c:bindings>  
3731     <part name="parameters" element="tns:GetLastTradePriceResponse">  
3732     </part>  
3733   </message>  
3734  
3735   <portType name="StockQuote">  
3736     <sca-c:bindings>  
3737       <sca-c:prefix name="stockQuote"/>  
3738     </sca-c:bindings>  
3739     <operation name="GetLastTradePrice">  
3740       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">  
3741       </input>  
3742       <output name="GetLastTradePriceResponse"  
3743         message="tns:GetLastTradePriceResponse">  
3744       </output>  
3745     </operation>  
3746   </portType>  
3747 </definitions>
```

3748

3749 Generated C header file:

```
3750 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
```

```

3751         *             serviceName="StockQuoteService") */
3752
3753     /* @WebOperation(operationName="GetLastTradePrice",
3754      *             response="getLastTradePriceResponse"
3755      *             action="urn:GetLastTradePrice") */
3756     struct getLastTradePrice {
3757         wchar_t *tickerSymbol; /* Since the length of the element is not
3758                               * restricted, a pointer is returned with the
3759                               * actual value held by the SCA runtime. */
3760     };
3761
3762     struct getLastTradePriceResponse {
3763         float return;
3764     };

```

3765 *Snippet D-8: Example <sca-c:struct> Element*

3766 D.6 <sca-c:parameter>

3767 <sca-c:parameter> specifies the name of the C function parameter or struct member associated with a
3768 specific WSDL message part or wrapper child element.

3769 **Format:**

```

3770     <sca-c:parameter name="CParameter" part="WSDLPart"
3771       childElementName="WSDLElement" type="CType"/>

```

3772 *Snippet D-9: <sca-c:parameter> Element Format*

3773 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.
- **parameter/@type : string (0..1)** – specifies the type of the parameter or struct member or return type. The @type attribute of a <parameter/> element MUST be either a C type specified in Simple Content Binding or, if the message part has complex content, a struct following the mapping specified in Complex Content Binding. [CD0002] The default type is determined by the mapping defined in Data Binding.

3785 **Applicable WSDL element(s):**

- wsdl:portType/wsdl:operation

3787 **Example:**

3788 Input WSDL file:

```

3789 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3790   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3791   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3792   xmlns:tns="http://www.example.org/"
3793   targetNamespace="http://www.example.org/">
3794
3795   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3796     xmlns:tns="http://www.example.org/"
3797     attributeFormDefault="unqualified"
3798     elementFormDefault="unqualified"
3799     targetNamespace="http://www.example.org/">
3800     <xsd:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3801     <xsd:element name="GetLastTradePriceResponse"
3802       type="tns:GetLastTradePriceResponse"/>

```

```

3803 <xs:complexType name="GetLastTradePrice">
3804     <xs:sequence>
3805         <xs:element name="symbol" type="xs:string"/>
3806     </xs:sequence>
3807 </xs:complexType>
3808 <xs:complexType name="GetLastTradePriceResponse">
3809     <xs:sequence>
3810         <xs:element name="return" type="xs:float"/>
3811     </xs:sequence>
3812 </xs:complexType>
3813 </xs:schema>
3814
3815 <message name="GetLastTradePrice">
3816     <part name="parameters" element="tns:GetLastTradePrice">
3817     </part>
3818 </message>
3819
3820 <message name="GetLastTradePriceResponse">
3821     <part name="parameters" element="tns:GetLastTradePriceResponse">
3822     </part>
3823 </message>
3824
3825 <portType name="StockQuote">
3826     <sca-c:bindings>
3827         <sca-c:prefix name="stockQuote"/>
3828     </sca-c:bindings>
3829     <operation name="GetLastTradePrice">
3830         <sca-c:bindings>
3831             <sca-c:function name="getLastTradePrice"/>
3832             <sca-c:parameter name="tickerSymbol"
3833                 part="tns:GetLastTradePrice/parameters"
3834                 childElementName="symbol"/>
3835         </sca-c:bindings>
3836         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3837         </input>
3838         <output name="GetLastTradePriceResponse"
3839             message="tns:GetLastTradePriceResponse">
3840         </output>
3841     </operation>
3842 </portType>
3843
3844 <binding name="StockQuoteServiceSoapBinding">
3845     <soap:binding style="document"
3846         transport="http://schemas.xmlsoap.org/soap/http"/>
3847     <operation name="GetLastTradePrice">
3848         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3849         <input name="GetLastTradePrice">
3850             <soap:body use="literal"/>
3851         </input>
3852         <output name="GetLastTradePriceResponse">
3853             <soap:body use="literal"/>
3854         </output>
3855     </operation>
3856 </binding>
3857
3858 <service name="StockQuoteService">
3859     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3860         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3861     </port>
3862 </service>
3863 </definitions>

```

3864

3865 Generated C header file:

```

3866     /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3867      *             serviceName="StockQuoteService") */
3868
3869     /* @WebFunction(operationName="GetLastTradePrice",
3870      *             action="urn:GetLastTradePrice")
3871      * @WebParam(paramName="tickerSymbol", name="symbol") */
3872      float getLastTradePrice(const wchar_t *tickerSymbol);

```

3873 *Snippet D-10: Example <sca-c:parameter> Element*

3874 D.7 JAX-WS WSDL Extensions

3875 An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL
 3876 extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.
 3877 Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted, and their corresponding SCA
 3878 WSDL extension. [CD0007]

3879

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

3880 *Table D-1: Allowed JAX-WS Extensions*

3881 D.8 sca-wsdlext-c-1.1.xsd

```

3882 <?xml version="1.0" encoding="UTF-8"?>
3883 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3884   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3885   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3886   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3887   elementFormDefault="qualified">
3888
3889   <element name="bindings" type="sca-c:BindingsType" />
3890   <complexType name="BindingsType">
3891     <choice minOccurs="0" maxOccurs="unbounded">
3892       <element ref="sca-c:prefix" />
3893       <element ref="sca-c:enableWrapperStyle" />
3894       <element ref="sca-c:function" />
3895       <element ref="sca-c:struct" />
3896       <element ref="sca-c:parameter" />
3897     </choice>
3898   </complexType>
3899
3900   <element name="prefix" type="sca-c:PrefixType" />
3901   <complexType name="PrefixType">
3902     <attribute name="name" type="xsd:string" use="required" />
3903   </complexType>
3904
3905   <element name="function" type="sca-c:FunctionType" />
3906   <complexType name="FunctionType">
3907     <attribute name="name" type="xsd:NCName" use="required" />
3908   </complexType>
3909
3910   <element name="struct" type="sca-c:StructType" />

```

```
3911 <complexType name="StructType">
3912   <attribute name="name" type="xsd:NCName" use="required" />
3913 </complexType>
3914
3915   <element name="parameter" type="sca-c:ParameterType" />
3916   <complexType name="ParameterType">
3917     <attribute name="part" type="xsd:string" use="required" />
3918     <attribute name="childElementName" type="xsd:QName" use="required" />
3919     <attribute name="name" type="xsd:NCName" use="required" />
3920     <attribute name="type" type="xsd:string" use="optional" />
3921   </complexType>
3922
3923   <element name="enableWrapperStyle" type="xsd:boolean" />
3924
3925 </schema>
```

3926 *Snippet D-11: SCA C WSDL Extension Schema*

3927

E XML Schemas

3928

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

```

3929 <?xml version="1.0" encoding="UTF-8"?>
3930 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3931     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3932     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3933     elementFormDefault="qualified">
3934
3935     <include schemaLocation="sca-core.xsd"/>
3936
3937     <element name="interface.c" type="sca:CInterface"
3938         substitutionGroup="sca:interface"/>
3939
3940     <complexType name="CInterface">
3941         <complexContent>
3942             <extension base="sca:Interface">
3943                 <sequence>
3944                     <element name="function" type="sca:CFunction"
3945                         minOccurs="0" maxOccurs="unbounded" />
3946                     <element name="callbackFunction" type="sca:CFunction"
3947                         minOccurs="0" maxOccurs="unbounded" />
3948                     <any namespace="#other" processContents="lax"
3949                         minOccurs="0" maxOccurs="unbounded"/>
3950                 </sequence>
3951                 <attribute name="header" type="string" use="required"/>
3952                 <attribute name="callbackHeader" type="string" use="optional"/>
3953             </extension>
3954         </complexContent>
3955     </complexType>
3956
3957     <complexType name="CFunction">
3958         <sequence>
3959             <choice minOccurs="0" maxOccurs="unbounded">
3960                 <element ref="sca:requires"/>
3961                 <element ref="sca:policySetAttachment"/>
3962             </choice>
3963             <any namespace="#other" processContents="lax" minOccurs="0"
3964                 maxOccurs="unbounded" />
3965         </sequence>
3966         <attribute name="name" type="NCName" use="required"/>
3967         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3968         <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3969         <attribute name="oneWay" type="boolean" use="optional"/>
3970         <attribute name="exclude" type="boolean" use="optional"/>
3971         <attribute name="input" type="NCName" use="optional"/>
3972         <attribute name="output" type="NCName" use="optional"/>
3973         <anyAttribute namespace="#other" processContents="lax"/>
3974     </complexType>
3975
3976 </schema>
```

3977

Snippet E-1: SCA *<interface.c>* Schema

3978

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

```

3979 <?xml version="1.0" encoding="UTF-8"?>
3980 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3981     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
```

```

3982      xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3983      elementFormDefault="qualified">
3984
3985      <include schemaLocation="sca-core.xsd"/>
3986
3987      <element name="implementation.c" type="sca:CImplementation"
3988          substitutionGroup="sca:implementation" />
3989
3990      <complexType name="CImplementation">
3991          <complexContent>
3992              <extension base="sca:Implementation">
3993                  <sequence>
3994                      <element name="operation" type="sca:CImplementationFunction"
3995                          minOccurs="0" maxOccurs="unbounded" />
3996                      <any namespace="#other" processContents="lax"
3997                          minOccurs="0" maxOccurs="unbounded"/>
3998                  </sequence>
3999                  <attribute name="module" type="NCName" use="required"/>
4000                  <attribute name="path" type="string" use="optional"/>
4001                  <attribute name="library" type="boolean" use="optional"/>
4002                  <attribute name="componentType" type="string" use="required"/>
4003                  <attribute name="eagerInit" type="boolean" use="optional"/>
4004                  <attribute name="init" type="boolean" use="optional"/>
4005                  <attribute name="destoy" type="boolean" use="optional"/>
4006                  <attribute name="allowsPassByReference" type="boolean"
4007                      use="optional"/>
4008              </extension>
4009          </complexContent>
4010      </complexType>
4011
4012      <complexType name="CImplementationFunction">
4013          <sequence>
4014              <choice minOccurs="0" maxOccurs="unbounded">
4015                  <element ref="sca:requires"/>
4016                  <element ref="sca:policySetAttachment"/>
4017              </choice>
4018              <any namespace="#other" processContents="lax" minOccurs="0"
4019                  maxOccurs="unbounded" />
4020          </sequence>
4021          <attribute name="name" type="NCName" use="required"/>
4022          <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4023          <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4024          <attribute name="allowsPassByReference" type="boolean"
4025              use="optional"/>
4026          <attribute name="init" type="boolean" use="optional"/>
4027          <attribute name="destoy" type="boolean" use="optional"/>
4028          <anyAttribute namespace="#other" processContents="lax"/>
4029      </complexType>
4030
4031  </schema>

```

4032 *Snippet E-2: SCA <implementation.c> Schema*

4033 **E.3 sca-contribution-c-1.1.xsd**

```

4034  <?xml version="1.0" encoding="UTF-8"?>
4035  <schema xmlns="http://www.w3.org/2001/XMLSchema"
4036      targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4037      xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4038      elementFormDefault="qualified">
4039
4040      <include schemaLocation="sca-contributions.xsd"/>
4041

```

```
4042 <element name="export.c" type="sca:CExport"
4043     substitutionGroup="sca:Export"/>
4044
4045 <complexType name="CExport">
4046     <complexContent>
4047         <attribute name="name" type="QName" use="required"/>
4048         <attribute name="path" type="string" use="optional"/>
4049     </complexContent>
4050 </complexType>
4051
4052 <element name="import.c" type="sca:CImport"
4053     substitutionGroup="sca:Import"/>
4054
4055 <complexType name="CImport">
4056     <complexContent>
4057         <attribute name="name" type="QName" use="required"/>
4058         <attribute name="location" type="string" use="required"/>
4059     </complexContent>
4060 </complexType>
4061
4062 </schema>
```

4063 *Snippet E-3: SCA <export.c> and <import.c> Schema*

4064

F Normative Statement Summary

4065

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

Conformance ID	Description
[C20001]	A C implementation MUST implement all of the operation(s) of the service interface(s) of its componentType.
[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 <interface.c/> element contains function or struct declarations that are not operations of the interface, then the functions or structs that are not operations of the interface MUST be excluded using <function/> child elements of the <interface.c/> element with @exclude="true".
[C20007]	If the header file identified by the @callbackHeader attribute of an <interface.c/> element contains function or struct declarations that are not operations of the callback interface, then the functions or structs that are not operations of the callback interface MUST be excluded using <callbackFunction/> child elements of the <interface.c/> element with @exclude="true".
[C20009]	The @name attribute of a <function/> child element of a <interface.c/> MUST be unique amongst the <function/> elements of that <interface.c/>.
[C20010]	The @name attribute of a <callbackFunction/> child element of a <interface.c/> MUST be unique amongst the <callbackFunction/> elements of that <interface.c/>.
[C20013]	The @name attribute of a <function/> child element of a <implementation.c/> MUST be unique amongst the <function/> elements of that <implementation.c/>.
[C20015]	An SCA runtime MUST NOT perform any synchronization of access to component implementations.
[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.

Conformance ID	Description
[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]	An SCA implementation MUST translate declarations to tokens as part of conversion to WSDL or compatibility testing.
[C80002]	<p>The return type and types of the parameters of a function of a remotable service interface MUST be one of:</p> <ul style="list-style-type: none"> Any of the C types specified in Simple Content Binding and Complex Content Binding. These types may be passed by-value or by-pointer. Unless the function and client indicate that they allow by-reference semantics (see AllowsPassByReference), a copy will be explicitly created by the runtime for any parameters passed by-pointer. An SDO DATAOBJECT. This type may be passed by-value or by-pointer. Unless the function and client indicate that they allow by-reference semantics (see AllowsPassByReference), a deep-copy of the DATAOBJECT will be created by the runtime for any parameters passed by-value or by-pointer. When by-reference semantics are allowed, the DATAOBJECT handle will be passed.
[C80003]	A C header file used to define an interface MUST declare at least one function or message format struct
[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 map an operation name, with the first character converted to lower case, to a function name. If necessary, to avoid name collisions, an SCA implementation MAY prepend 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 name.
[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.
[C100006]	In the absence of customizations, an SCA implementation MUST map the name of the message element referred to by a fault element to the 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.

Conformance ID	Description
[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 a function name to an operation name, stripping the portType name, if present and any namespace prefix from the front of function name before mapping it to the operation name.
[C100011]	In the absence of customizations, an SCA implementation MUST map a parameter name, if present, to a 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”.
[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 struct. The mapping from WSDL MAY be to DataObjects and/or structs. The mapping to and from structs 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 or const and In/Out and Out parameters as pass via pointers.
[C100021]	An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.
[C100022]	An SCA implementation MAY map boolean to _Bool by default.
[C100023]	An SCA implementation MUST map a WSDL portType to a remotable C interface definition.
[C100024]	An SCA implementation MUST map a C interface definition to WSDL as if it has a @WebService annotation with all default values.
[C110001]	An SCA implementation MUST reject a composite file that does not conform to http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd or http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd .
[C110002]	An SCA implementation MUST reject a componentType file that does not conform to http://docs.oasis-open.org/opencsa/sca/200912/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/200912/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 .

4066 Table F-1: SCA C Core Normative Statements

4067 F.1 Program-Based Normative Statements Summary

4068 This section contains a list of normative statements related to program-based component
 4069 implementations for this specification.

Conformance ID	Description
[C100005]	For components implemented in a program, in the absence of customizations, an SCA implementation MUST map an operation name, with the first character converted to lowercase to a request struct name. If necessary, to avoid name collisions, an SCA implementation MAY 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.
[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.
[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.
[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.

4070 Table F-2: SCA C Program-Based Normative Statements

4071 F.2 Annotation Normative Statement Summary

4072 This section contains a list of normative statements related to source file annotations for this specification.

Conformance ID	Description
[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 @WebService annotation specified as if @Remotable and @Interface were specified with the name value of the @WebService annotation used as the name value of the @Interface annotation.
[CA0004]	An SCA implementation MUST treat a function with a @WebFunction annotation specified as if @Function was specified with the operationName value of the @WebFunction annotation used as the name value of the @Function annotation and the exclude value of the @WebFunction annotation used as the exclude value of the @Function annotation.

Conformance ID	Description
[CA0005]	An SCA implementation MUST treat a struct with a @WebOperation annotation specified as if @Operation was specified with the operationName value of the @WebOperation annotation used as the name value of the @Operation annotation, the response value of the @WebOperation annotation used as the response value of the @Operation annotation and the exclude value of the @WebFunction annotation used as the exclude value of the @Operation annotation.
[CA0006]	While annotations are defined using the /* ... */ format for comments, if the // ... format is supported by a C compiler, the // ... format MAY be supported by an SCA implementation annotation processor.
[CA0007]	An SCA implementation MUST ensure that all variables in a component implementation with the same name and annotated with @Property have the same type.
[CC0001]	An SCA implementation MUST treat any instance of a @Remotable 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, if specified, and no other parameters was specified.
[CC0002]	An SCA implementation MUST treat a function annotated with an @Function annotation and without an explicit @WebFunction annotation as if a @WebFunction annotation with an operationName value equal to the name value of the @Function annotation, an exclude value equal to the exclude value of the @Function annotation and no other parameters was specified.
[CC0003]	An SCA implementation MUST treat a struct annotated with 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 response value of the @Operation annotation, an exclude value equal to the exclude value of the @Operation annotation and no other parameters was specified.
[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 either one of the simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema or, if the type of the parameter is a struct, the QName of a XSD complex type following the mapping specified in Complex Content Binding.
[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 .
[CC0009]	The value of the paramName of a @WebParam annotation MUST be the name of a parameter of the function the annotation is applied to.

4073 Table F-3: SCA C Annotation Normative Statements

4074 **F.3 WSDL Extension Normative Statement Summary**

4075 This section contains a list of normative statements related to WSDL extensions for this specification.

Conformance ID	Description
----------------	-------------

[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 either a C type specified in Simple Content Binding or, if the message part has complex content, a struct following the mapping specified in Complex Content Binding.
[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. Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted, and their corresponding SCA WSDL extension.

4076 *Table F-4: SCA C WSDL Extension Normative Statements*4077

F.4 JAX-WS Normative Statements

4078 The JAX-WS 2.1 specification [JAXWS21] defines normative statements for various requirements defined
4079 by that specification. Table F-5 outlines those normative statements which apply to the WSDL mapping
4080 described in this specification.

Number	Conformance Point	Notes	Conformance ID
2.1	WSDL 1.1 support	[A]	[CF0001]
2.2	Customization required	[CD0001] The reference to the JAX-WS binding language is treated as a reference to the C WSDL extensions defined in C WSDL Mapping Extensions	
2.3	Annotations on generated classes		[CF0002]
2.5	WSDL and XML Schema import directives		[CF0003]
2.6	Optional WSDL extensions		[CF0004]
2.7	SEI naming	[C100001]	
2.8	javax.jws.WebService required	[B] References to javax.jws.WebService in the conformance statement are treated as the C annotation @WebService.	[CF0005]
2.10	Method naming	[C100002] and [C100005]	

Number	Conformance Point	Notes	Conformance ID
2.11	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]
2.12	Transmission primitive support		[CF0007]
2.13	Using javax.jws.OneWay	[A], [B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0008]
2.14	Using javax.jws.SOAPBinding	[A], [B] References to javax.jws.SOAPBinding in the conformance statement are treated as the C annotation @SOAPBinding.	[CF0009]
2.15	Using javax.jws.WebParam	[A], [B] References to javax.jws.WebParam in the conformance statement are treated as the C annotation @WebParam.	[CF0010]
2.16	Using javax.jws.WebResult	[A], [B] References to javax.jws.WebResult in the conformance statement are treated as the C annotation @WebResult.	[CF0011]
2.18	Non-wrapped parameter naming	[C100003]	
2.19	Default mapping mode		[CF0012]
2.20	Disabling wrapper style	[B] References to jaxws:enableWrapperStyle in the conformance statement are treated as the C annotation sca-c:enableWrapperStyle.	[CF0013]
2.21	Wrapped parameter naming	[C100004]	
2.22	Parameter name clash	[A]	[CF0014]
2.38	javax.xml.ws.WebFault required	[B] References to javax.jws.WebFault in the conformance statement are treated as the C annotation @WebFault.	[CF0015]
2.39	Exception naming	[C100006]	

Number	Conformance Point	Notes	Conformance ID
2.40	Fault equivalence	[A] References to fault exception classes are treated as references to fault message structs.	[CF0016]
2.42	Required WSDL extensions	MIME Binding not necessary	[CF0018]
2.43	Unbound message parts	[A]	[CF0019]
2.44	Duplicate headers in binding		[CF0020]
2.45	Duplicate headers in message		[CF0021]
3.1	WSDL 1.1 support	[A]	[CF0022]
3.2	Standard annotations	[A] [CC0005]	
3.3	Java identifier mapping	[A]	[CF0023]
3.6	WSDL and XML Schema import directives		[CF0024]
3.8	portType naming	[C100008]	
3.11	Operation naming	[C100009] and [C100010]	
3.12	One-way mapping	[B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0025]
3.13	One-way mapping errors		[CF0026]
3.15	Parameter classification	[C100017]	
3.16	Parameter naming	[C100011] and [C100014]	
3.17	Result naming	[C100012]	
3.18	Header mapping of parameters and results	References to javax.jws.WebParam in the conformance statement are treated as the C annotation @WebParam. References to javax.jws.WebResult in the conformance statement are treated as the C annotation @WebResult.	[CF0027]
3.24	Exception naming	[CC0004]	
3.27	Binding selection	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CF0029]
3.28	SOAP binding support	[A]	[CF0030]

Number	Conformance Point	Notes	Conformance ID
3.29	SOAP binding style required		[CF0031]
3.31	Port selection		[CF0032]
3.32	Port binding	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CF0033]

4081 [A] All references to Java in the conformance point are treated as references to C.

4082 [B] Annotation generation is only necessary if annotations are supported by an SCA implementation.

4083 *Table F-5: JAX-WS Normative Statements that are Applicable to SCA C*

4084 F.4.1 Ignored Normative Statements

Number	Conformance Point
2.4	Definitions mapping
2.9	javax.xml.bind.XmlSeeAlso required
2.17	use of JAXB annotations
2.23	Using javax.xml.ws.RequestWrapper
2.24	Using javax.xml.ws.ResponseWrapper
2.25	Use of Holder
2.26	Asynchronous mapping required
2.27	Asynchronous mapping option
2.28	Asynchronous method naming
2.29	Asynchronous parameter naming
2.30	Failed method invocation
2.31	Response bean naming
2.32	Asynchronous fault reporting
2.33	Asynchronous fault cause
2.34	JAXB class mapping
2.35	JAXB customization use
2.36	JAXB customization clash
2.37	javax.xml.ws.wsaddressing.W3CEndpointReference
2.41	Fault Equivalence
2.46	Use of MIME type information
2.47	MIME type mismatch
2.48	MIME part identification

Number	Conformance Point
2.49	Service superclass required
2.50	Service class naming
2.51	javax.xml.ws.WebServiceClient required
2.52	Default constructor required
2.53	2 argument constructor required
2.54	Failed getPort Method
2.55	javax.xml.ws.WebEndpoint required
3.4	Method name disambiguation
3.5	Package name mapping
3.7	Class mapping
3.9	Inheritance flattening
3.10	Inherited interface mapping
3.14	use of JAXB annotations
3.19	Default wrapper bean names
3.20	Default wrapper bean package
3.21	Null Values in rpc/literal
3.25	java.lang.RuntimeExceptions and java.rmi.RemoteExceptions
3.26	Fault bean name clash
3.30	Service creation

4085 *Table F-6: JAX-WS Normative Statements that Are Not Applicable to SCA C*

4086 **G Migration**

4087 To aid migration of an implementation or clients using an implementation based the version of the Service
4088 Component Architecture for C defined in [SCA C Client and Implementation V1.00](#), this appendix identifies
4089 the relevant changes to APIs, annotations, or behavior defined in V1.00.

4090 **G.1 Implementation.c attributes**

4091 `@location` has been replaced with `@path`.

4092 **G.2 SCALocate and SCALocateMultiple**

4093 `SCALocate()` and `SCALocateMultiple()` have been renamed to `SCAGetReference()`
4094 `SCAGetReferences()` respectively.

4095

H Acknowledgements

4096 The following individuals have participated in the creation of this specification and are gratefully
4097 acknowledged:

4098 **Participants:**

4099

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

4100

I Revision History

4101

[optional; should not be included in OASIS Standards]

4102

Revision	Date	Editor	Changes Made
			•

4103