



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

Committee Draft 05 / Public Review Draft 03

4 March 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-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)

Previous Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd04.html>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd04.doc>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd04.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](#)

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-cd05.zip>

Abstract:

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

The SCA C implementation model describes how to implement SCA components in C. A component implementation itself can also be a client to other services provided by other components or external services. The document describes how a 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	Conventions.....	9
1.3.1	Naming Conventions.....	9
1.3.2	Typographic Conventions.....	9
2	Basic Component Implementation Model.....	10
2.1	Implementing a Service.....	10
2.1.1	Implementing a Remotable Service.....	11
2.1.2	AllowsPassByReference.....	11
2.1.3	Implementing a Local Service.....	12
2.2	Component and Implementation 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	SCAReleaseCallback	38
6.1.11	SCAInvokeAsync	39
6.1.12	SCAInvokePoll	39
6.1.13	SCACheckResponse	40
6.1.14	SCACancelInvoke	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 Statments	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

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.

1.1 Terminology

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

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

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"	

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

1.2 Normative References

- [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>
- [ASSEMBLY] OASIS Committee Draft 05, *Service Component Architecture Assembly Model Specification Version 1.1*, January 2010. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd05.pdf>
- [POLICY] OASIS Committee Draft 02, *SCA Policy Framework Version 1.1*, March 2009. <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf>
- [SDO21] OSOA, *Service Data Objects For C Specification*, September 2007. http://www.osoa.org/download/attachments/36/SDO_Specification_C_V2.1.pdf
- [WSDL11] World Wide Web Consortium, *Web Service Description Language (WSDL)*, March 2001. <http://www.w3.org/TR/wsdl>
- [XSD] World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*, October 2004. <http://www.w3.org/TR/xmlschema-2/>
- [JAXWS21] Doug. Kohlert and Arun Gupta, *The Java API for XML-Based Web Services (JAX-WS) 2.1*, JSR, JCP, May 2007. <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>

35 1.3 Conventions

36 1.3.1 Naming Conventions

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

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

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

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

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

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

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

48 1.3.2 Typographic Conventions

49 This specification follows typographic conventions for specific constructs:

- 50 • Conformance points are **highlighted**, **[numbered]** and cross-referenced to Normative Statement
51 Summary

52 • XML attributes are identified in text as `@attribute`

53 • Language identifiers used in text are in `courier`

54 • Literals in text are in *italics*

55 2 Basic Component Implementation Model

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

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

61 2.1 Implementing a Service

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

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

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

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

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

72

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

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

76

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

83 *Snippet 2-2: C Service Implementation*

84

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

86

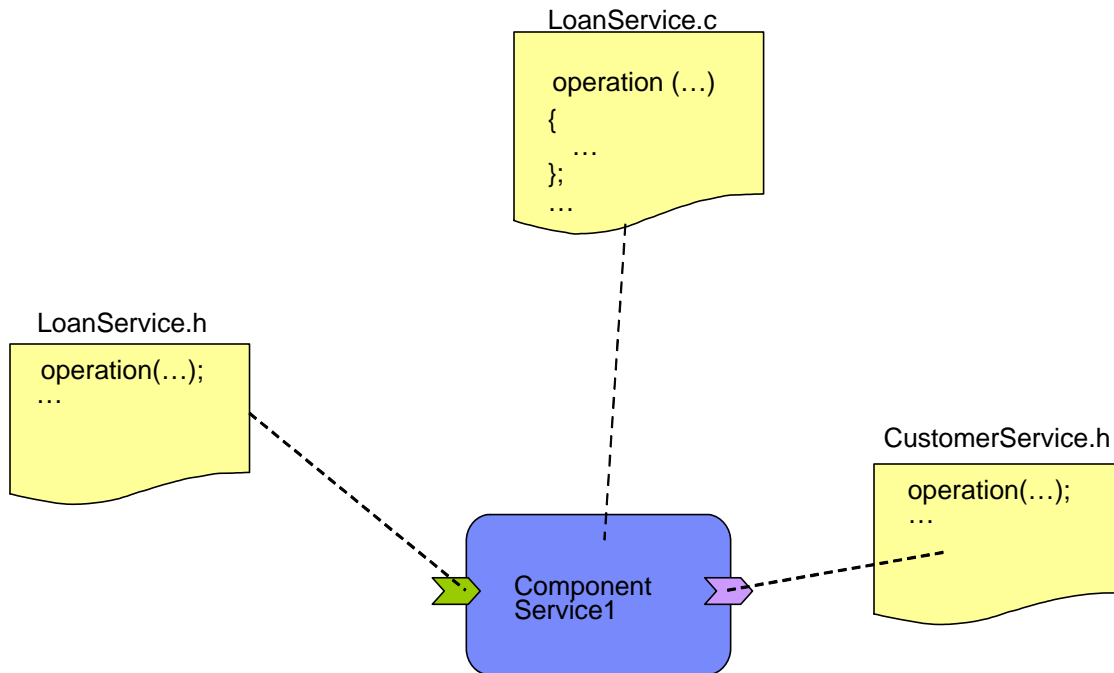
```
87 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
88     <service name="LoanService">  
89         <interface.c header="LoanService.h"/>  
90     </service>  
91 </componentType>
```

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

93

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

96



97
98 *Figure 2-1: Relationship of C Implementation Artifacts*

99 **2.1.1 Implementing a Remotable Service**

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

```
103
104 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
105   <service name="LoanService">
106     <interface.c header="LoanService.h" remotable="true"/>
107   </service>
108 </componentType>
```

109 *Snippet 2-4: ComponentType for a Remotable Service*

110 **2.1.2 AllowsPassByReference**

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

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

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

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

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

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

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

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

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

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

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

152 2.1.3 Implementing a Local Service

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

154 2.2 Component and Implementation Lifecycles

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

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

170 2.3 Implementing a Configuration Property

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

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

175

```
176 #include "SCA.h"
177
178 void clientFunction()
179 {
180
181     ...
182
183     int32_t loanRating;
184     int values, compCode, reason;
185
186     ...
187
188     SCAPropertyInt(L"maxLoanValue", &loanRating, &values, &compCode, &reason);
189
190     ...
191
192 }
```

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

194

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

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

209 2.4 Component Type and Component

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

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

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

217

```
218 /* LoanService interface */
219 char approveLoan(long customerNumber, long loanAmount);
```

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

221
222
223
224
225
226
227
228

```
#include "LoanService.h"

char approveLoan(long customerNumber, long loanAmount)
{
    ...
}
```

229 *Snippet 2-7: C Service Implementation*

230

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

232

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

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

239

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

241

```
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  name="LoanComposite" >
  ...
  <component name="LoanService">
    <implementation.c module="loan" componentType="LoanService" />
  </component>
  ...
</composite>
```

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

255 **2.4.1 Interface.c**

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

258

```
<!-- interface.c schema snippet -->
<interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  header="string" remotable="boolean"? callbackHeader="string"?
  requires="listOfQNames"? policySets="listOfQNames"? >

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

</interface.c>
```

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

271

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

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

294 2.4.2 Function and CallbackFunction

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

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

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

309

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



```

324     <policySetAttachment/>*
325     </callbackFunction> *
326
327 </interface.c>

```

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

329

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

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

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

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

352 2.4.3 Implementation.c

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

355

```

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

```

368 *Snippet 2-12: Pseudo-schema for C Implementation Element*

369

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

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

399 The *interface.c* element has the **child elements**:

- 400 • **function : CImplementationFunction (0..n)** – see Implementation Function
- 401 • **requires : requires (0..n)** - See the Policy Framework specification [POLICY] for a description of this
402 element.
- 403 • **policySetAttachment : policySetAttachment (0..n)** - See the Policy Framework specification
404 [POLICY] for a description of this element.

405 2.4.4 Implementation Function

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

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

410

```

411 <!-- ImplementationFunction schema snippet -->
412 <implementation.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"... >
413
414     <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
415         allowsPassByReference="Boolean"? init="Boolean"?
416         destroy="Boolean"? >
417         <requires/>*
418         <policySetAttachment/>*
419     </function> *
420

```

```
421 </implementation.c>
```

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

423

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

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

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

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

443 2.5 Implementing a Service with a Program

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

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

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

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

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

466

```
467 #include "SCA.h"  
468 #include "myInterface.h"
```

```

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

```

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

496 3 Basic Client Model

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

499 3.1 Accessing Services from Component Implementations

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

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

502

```
503 void SCAGetReference(wchar_t *referenceName, SCAREF *referenceToken,  
504                    int *compCode, int *reason);  
505 void SCAInvoke(SCAREF referenceToken, wchar_t *operationName,  
506              int inputMsgLen, void *inputMsg,  
507              int outputMsgLen, void *outputMsg, int *compCode, int *reason);
```

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

509

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

511

```
512 #include "SCA.h"  
513  
514 void clientFunction()  
515 {  
516  
517     SCAREF serviceToken;  
518     int compCode, reason;  
519     long custNum = 1234;  
520     short rating;  
521  
522     ...  
523     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);  
524     SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),  
525              (void *)&custNum, sizeof(rating), (void *)&rating,  
526              &compCode, &reason);  
527  
528 }
```

529 *Snippet 3-2: Using SCAGetReference*

530

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

533

```
534 SCAREF *tokens;  
535 int num_targets;  
536 ...  
537 myFunction(...) {  
538     int compCode, reason;  
539     ...  
540     SCAGetReferences(L"myReference", &tokens, &num_targets, &compCode,  
541                    &reason);  
542     for (i = 0; i < num_targets; i++)  
543     {  
544         SCAInvoke(tokens[i], L"myOperation", sizeof(inputMsg),  
545                 (void *)&inputMsg, 0, NULL, &compCode, &reason);  
546     }
```

```
546     };
547     };
```

548 *Snippet 3-3: Using SCAGetReferences*

549

550 **3.2 Accessing Services from non-SCA Component Implementations**

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

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

554

```
555 #include "SCA.h"
556
557 void externalFunction()
558 {
559     SCAREF serviceToken;
560     int compCode, reason;
561     long custNum = 1234;
562     short rating;
563
564     SCAEntryPoint(L"customerService", L"http://example.com/mydomain",
565                 &serviceToken, &compCode, &reason);
566     SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),
567              (void *)&custNum, sizeof(rating), (void *)&rating,
568              &compCode, &reason);
569 }
```

570 *Snippet 3-4: Using SCAEntryPoint*

571

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

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

- 575
- componentName/serviceName/bindingName

576 **3.3 Calling Service Operations**

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

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

582 **3.3.1 Proxy Functions**

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

588 For SCA, proxy functions have the form:

```
589 <functionReturn> SCA_<functionName>( SCAREF referenceToken,
590                                     <functionParameters> )
```

591 where:

- 592
- <functionName> is the name of interface function

593 • <functionParameters> are the parameters of the interface function

594 • <functionReturn> is the return type of the interface function

595 Snippet : Proxy Function Format

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

597 • `ENOENT` if a remote service is unavailable

598 • `EFAULT` if a fault is returned by the operation

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

600

```
601 #include "SCA.h"
602
603 void clientFunction()
604 {
605
606     SCAREF serviceToken;
607     int compCode, reason;
608     long custNum = 1234;
609     short rating;
610
611     ...
612     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
613     errno = 0;
614     rating = SCA_getCreditRating(serviceToken, custNum);
615     if (errno) {
616         /* handle error or fault */
617     }
618     else {
619         ...
620     }
621 }
622 }
```

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

624

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

626 3.4 Long Running Request-Response Operations

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

636 3.4.1 Asynchronous Invocation

637 The asynchronous style of invocation uses `SCAInvokeAsync()` which has the same signature as
638 `SCAInvoke()` without the `outputMsgLen` or `outputMsg` parameters but with a parameter taking the
639 address of a handler function. This API sends the operation request. The handler function has the
640 signature

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

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

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

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

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

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

651 the asynchronous invocation style includes a proxy function

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

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

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

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

```
657  
658 #include "SCA.h"  
659 #include "TravelService.h"  
660  
661 SCAREF serviceToken;  
662 int compCode, reason;  
663  
664 void makeReservationsHandler(short rspType)  
665 {  
666     struct confirmationData cd;  
667     wchar_t *fault, *faultDetails;  
668  
669     if (rspType == SCA_REPLY_MESSAGE) {  
670         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);  
671         ...  
672     }  
673     else {  
674         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,  
675                             &faultDetails, &compCode, &reason);  
676         if (wcscmp(*fault, L"noFlight") {  
677             ...  
678         }  
679         else {  
680             ...  
681         }  
682     }  
683  
684     return;  
685 }  
686  
687 void clientFunction()  
688 {  
689     struct itineraryData id;  
690     ...  
691  
692     void (*ah)(short) = &makeReservationsHandler;  
693  
694     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);  
695  
696     SCAInvokeAsync(serviceToken, L"makeReservations", sizeof(itineraryData),  
697                   (void *)&id, ah, &compCode, &reason);  
698  
699
```

```
700
701     return;
702 }
```

703 *Snippet 3-8: Using Asynchronous Invocation*

704 3.4.2 Polling Invocation

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

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

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

711 the polling invocation style includes a proxy function

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

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

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

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

```
717
718 #include "SCA.h"
719 #include "TravelService.h"
720
721 void pollingClientFunction()
722 {
723     SCAREF serviceToken;
724     int compCode, reason;
725     short rspType;
726
727     struct itineraryData id;
728     struct confirmationData cd;
729     wchar_t *fault, *faultDetails;
730
731     ...
732
733     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
734
735     SCAInvokePoll(serviceToken, L"makeReservations", sizeof(itineraryData),
736                 (void *) &id, &compCode, &reason);
737
738     SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
739     while (!rspType) {
740         // do something, then wait for some time...
741         SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
742     }
743     if (rspType == SCA_REPLY_MESSAGE) {
744         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
745         ...
746     }
747     else {
748         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
749                           &faultDetails, &compCode, &reason);
750         if (wcscmp(*fault, L"noFlight") {
751             ...
752         }
753         else {
754             ...

```



```

755     }
756     }
757
758     return;
759 }

```

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

761 **3.4.3 Synchronous Invocation**

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

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

```

769
770 #include "SCA.h"
771 #include "TravelService.h"
772
773 void synchronousClientFunction()
774 {
775     SCAREF serviceToken;
776     int compCode, reason;
777
778     struct itineraryData id;
779     struct confirmationData *cd;
780     wchar_t *fault, *faultDetails;
781
782     ...
783
784     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
785
786     SCAInvoke(serviceToken, L"makeReservations", sizeof(itineraryData),
787             (void *)&id, sizeof(confirmationData), (void *)&cd,
788             &compCode, &reason);
789     if (compCode == SCA_FAULT) {
790         ...
791     }
792     else {
793         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
794             &faultDetails, &compCode, &reason);
795         if (wcscmp(*fault, L"noFlight") {
796             ...
797         }
798         else {
799             ...
800         }
801     }
802
803     return;
804 }

```

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

806 4 Asynchronous Programming

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

814 4.1 Non-blocking Calls

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

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

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

```
824  
825 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
826   <service name="LoanService">  
827     <interface.c header="LoanService.h">  
828       <function name="reportEvent" oneWay="true" />  
829     </interface.c>  
830   </service>  
831 </componentType>
```

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

833
834 SCA does not currently define a mechanism for making non-blocking calls to functions that return values.
835 It is considered to be a best practice that service designers define one-way operations as often as
836 possible, in order to give the greatest degree of binding flexibility to deployers.

837 4.2 Callbacks

838 Callbacks services are used by *bidirectional services* as defined in the Assembly Specification
839 **[ASSEMBLY]:**

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

```
843  
844 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" >  
845   <service name="MyService">  
846     <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h"/>  
847   </service>  
848 </componentType>
```

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

850 4.2.1 Using Callbacks

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

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

860

```
861 double requestQuotation(char *productCode,int quantity);  
862  
863 char *getState();  
864 char *getZipCode();  
865 char *getCreditRating();
```

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

867

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

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

875

```
876 #include "QuotationCallback.h"  
877 #include "SCA.h"  
878  
879 double requestQuotation(char *productCode,int quantity) {  
880     double price, discount = 0;  
881     char state[3], creditRating[4];  
882     SCAREF callbackRef;  
883     int compCode, reason;  
884  
885     price = getPrice(productCode, quantity);  
886  
887     SCAGetCallback(L"", &callbackRef, &compCode, &reason);  
888     SCAInvoke(callbackRef, L"getState", 0, NULL, sizeof(state), state,  
889             &compCode, &reason);  
890     if (quantity > 1000 && strcmp(state,"FL") == 0)  
891         discount = 0.05;  
892     SCAInvoke(callbackRef, L"getCreditRating", 0, NULL, sizeof(creditRating),  
893             creditRating, &compCode, &reason);  
894     if (quantity > 10000 && creditRating[0] == 'A')  
895         discount += 0.05;  
896     SCAReleaseCallback(callbackRef, &compCode, &reason);  
897     return price * (1-discount);  
898 }
```

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

900

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

```

904
905 #include "QuotationCallback.h"
906 #include "SCA.h"
907
908 char state[3] = "TX", zipCode[6] = "78746", creditRating[3] = "AA";
909
910 ClientFunction() {
911     SCAREF serviceToken;
912     int compCode, reason;
913
914     SCAGetReference(L"quotationService", &serviceToken, &compCode, &reason);
915
916     SCA_requestQuotation(serviceToken, "AB123", 2000);
917 }
918
919 char *getState() {
920     return state;
921 }
922 char *getZipCode() {
923     return zipCode;
924 }
925 char *getCreditRating() {
926     return creditRating;
927 }

```

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

929

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

934 4.2.2 Callback Instance Management

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

940 4.2.3 Implementing Multiple Bidirectional Interfaces

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

944 5 Error Handling

945 Clients calling service operations will experience business logic errors, and SCA runtime errors.

946 Business logic errors are generated by the implementation of the called service operation. They are
947 handled by client the invoking the operation of the service.

948 SCA runtime errors are generated by the SCA runtime and signal problems in the management of the
949 execution of components, and in the interaction with remote services. The SCA C API includes two return
950 codes on every function, a completion code and a reason code. The reason code is used to provide more
951 detailed information if a function does not complete successfully.

952

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

972 *Snippet 5-1: SCA Constant Defintions*

973

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

976

6 C API

977

6.1 SCA Programming Interface

978

The SCA API definition is:

979

980

```
typedef void *SCAREF;
```

981

```
void SCAGetReference(wchar_t *referenceName,  
                    SCAREF *referenceToken,  
                    int *compCode,  
                    int *reason);
```

986

```
void SCAGetReferences(wchar_t *referenceName,  
                     SCAREF **referenceTokens,  
                     int *num_targets,  
                     int *compCode,  
                     int *Reason);
```

992

```
void SCAInvoke(SCAREF token,  
               wchar_t *operationName,  
               int inputMsgLen,  
               void *inputMsg,  
               int *outputMsgLen,  
               void *outputMsg,  
               int *compCode,  
               int *reason);
```

1000

1001

```
void SCAPropertyBoolean(wchar_t *propertyName,  
                        char *value,  
                        int *num_values,  
                        int *compCode,  
                        int *reason);
```

1007

```
void SCAPropertyByte(wchar_t *propertyName,  
                     int8_t *value,  
                     int *num_values,  
                     int *compCode,  
                     int *reason);
```

1013

```
void SCAPropertyBytes(wchar_t *propertyName,  
                      int8_t **value,  
                      int *size,  
                      int *num_values,  
                      int *compCode,  
                      int *reason);
```

1019

1020

```
void SCAPropertyChar(wchar_t *propertyName,  
                     wchar_t *value,  
                     int *num_values,  
                     int *compCode,  
                     int *reason);
```

1026

```
void SCAPropertyChars(wchar_t *propertyName,  
                      wchar_t **value,  
                      int *size,  
                      int *num_values,  
                      int *compCode,  
                      int *reason);
```

1032

```

1033
1034 void SCAPPropertyCChar(wchar_t *propertyName,
1035                        char *value,
1036                        int *num_values,
1037                        int *compCode,
1038                        int *reason);
1039
1040 void SCAPPropertyCChars(wchar_t *propertyName,
1041                        char **value,
1042                        int *size,
1043                        int *num_values,
1044                        int *compCode,
1045                        int *reason);
1046
1047 void SCAPPropertyShort(wchar_t *propertyName,
1048                       int16_t *value,
1049                       int *num_values,
1050                       int *compCode,
1051                       int *reason);
1052
1053 void SCAPPropertyInt(wchar_t *propertyName,
1054                    int32_t *value,
1055                    int *num_values,
1056                    int *compCode,
1057                    int *reason);
1058
1059 void SCAPPropertyLong(wchar_t *propertyName,
1060                     int64_t *value,
1061                     int *num_values,
1062                     int *compCode,
1063                     int *reason);
1064
1065 void SCAPPropertyFloat(wchar_t *propertyName,
1066                      float *value,
1067                      int *num_values,
1068                      int *compCode,
1069                      int *reason);
1070
1071 void SCAPPropertyDouble(wchar_t *propertyName,
1072                       double *value,
1073                       int *num_values,
1074                       int *compCode,
1075                       int *reason);
1076
1077 void SCAPPropertyString(wchar_t *propertyName,
1078                       wchar_t **value,
1079                       int *size,
1080                       int *num_values,
1081                       int *compCode,
1082                       int *reason);
1083
1084 void SCAPPropertyCString(wchar_t *propertyName,
1085                        char **value,
1086                        int *size,
1087                        int *num_values,
1088                        int *compCode,
1089                        int *reason);
1090
1091 void SCAPPropertyStruct(wchar_t *propertyName,
1092                       void *value,
1093                       int *num_values,
1094                       int *compCode,
1095                       int *reason);
1096

```

```

1097 void SCAGetReplyMessage(SCAREF token,
1098                         int *bufferLen,
1099                         void *buffer,
1100                         int *compCode,
1101                         int *reason);
1102
1103 void SCAGetFaultMessage(SCAREF token,
1104                        int *bufferLen,
1105                        wchar_t **faultName,
1106                        void *buffer,
1107                        int *compCode,
1108                        int *reason);
1109
1110 void SCASetFaultMessage(wchar_t *serviceName,
1111                        wchar_t *operationName,
1112                        wchar_t *faultName,
1113                        int bufferLen,
1114                        void *buffer,
1115                        int *compCode,
1116                        int *reason);
1117
1118 void SCASelf(wchar_t *serviceName,
1119             SCAREF *serviceToken,
1120             int *compCode,
1121             int *reason);
1122
1123 void SCAGetCallback(wchar_t *serviceName,
1124                    SCAREF *serviceToken,
1125                    int *compCode,
1126                    int *reason);
1127
1128 void SCAReleaseCallback(SCAREF serviceToken,
1129                        int *compCode,
1130                        int *reason);
1131
1132 void SCAInvokeAsync(SCAREF token,
1133                    wchar_t *operationName,
1134                    int inputMsgLen,
1135                    void *inputMsg,
1136                    void (*handler)(short),
1137                    int *compCode,
1138                    int *reason);
1139
1140 void SCAInvokePoll(SCAREF token,
1141                   wchar_t *operationName,
1142                   int inputMsgLen,
1143                   void *inputMsg,
1144                   int *compCode,
1145                   int *reason);
1146
1147 void SCACheckResponse(SCAREF token,
1148                      short *responseType,
1149                      int *compCode,
1150                      int *reason);
1151
1152 void SCACancelInvoke(SCAREF token,
1153                     int *compCode,
1154                     int *reason);
1155
1156 void SCAEntryPoint(wchar_t *serviceURI,
1157                   wchar_t *domainURI,
1158                   SCAREF *serviceToken,
1159                   int *compCode,
1160                   int *reason);

```


1161 *Snippet 6-1: SCA API Definition*

1162 **6.1.1 SCAGetReference**

1163 A C component implementation uses `SCAGetReference()` to initialize a Reference before invoking any
 1164 operations of the Reference.

Precondition	C component instance is running	
Input Parameter	<code>referenceName</code>	Name of the Reference to initialize
Output Parameters	<code>referenceToken</code>	Token to be used in subsequent <code>SCAInvoke()</code> calls. This will be NULL if <code>referenceName</code> is not defined for the component.
	<code>compCode</code>	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	<code>reason</code>	SCA_SERVICE_UNAVAILABLE if no suitable service exists in the domain SCA_MULTIPLE_SERVICES if the reference is bound to multiple services
Post Condition	If an operational Service exists for the reference, the component instance has a valid token to use for subsequent runtime calls.	

1165 *Table 6-1: SCAGetReference Details*

1166 **6.1.2 SCAGetReferences**

1167 A C component implementation uses `SCAGetReferences()` to initialize a Reference that might be
 1168 bound to multiple Services before invoking any operations of the Reference.

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

1169 *Table 6-2: SCAGetReferencse Details*

1170 **6.1.3 SCAInvoke**

1171 A C component implementation uses `SCAInvoke()` to invoke an operation of an interface.

Precondition	C component instance is running and has a valid token	
Input Parameters	Token	Token returned by prior <code>SCAGetReference()</code> or <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
	operationName	Name of the operation to invoke
	inputMsgLen	Length of the request message buffer
	inputMsg	Request message
In/Out Parameter	outputMsgLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	outputMsg	Response message
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the response data was truncated. The buffer size needs to be increased and <code>SCAGetReplyMessage()</code> called with the larger buffer. SCACC_FAULT, if the operation returned a business fault. <code>SCAGetFaultMessage()</code> 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 <code>SCA_SERVICE_UNAVAILABLE</code> reason is returned, the token remains valid for subsequent calls.	

1172 Table 6-3: *SCAInvoke Details*

1173 6.1.4 SCAProperty<T>

1174 A C component implementation uses `SCAProperty<T>()` to get the configured value for a Property.

1175 This API is available for Boolean, Byte, Bytes, Char, Chars, CChar, CChars, Short, Int, Long, Float,
1176 Double, String, CString and Struct. The Char, Chars, and String variants return `wchar_t` based data while
1177 the CChar, CChars, and CString variants return char based data. The Bytes, Chars, and CChars variants
1178 return a buffer of data. The String and CString variants return a null terminated string.

1179 An SCA runtime MAY additionally provide a `DataObject` variant of this API for handling properties with
1180 complex XML types. The type of the value parameter in this variant is `DATAOBJECT`. [C60002]

1181 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.	

1182 Table 6-4: SCAProperty<T> Details for fixed length types

1183

1184 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	Input: Maximum number of bytes or characters that can be returned for each configured value Output: Actual number of bytes or characters returned or size needed to hold a configured value. 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 of 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.
	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 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 was 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.	

1185 *Table 6-5: SCAProperty<T> Details for variable length types*

1186 6.1.5 SCAGetReplyMessage

1187 A C component implementation uses SCAGetReplyMessage() to retrieve the reply message of an
 1188 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 SCAInvoke() returned a SCACC_WARNING compCode or has a valid serviceToken and an SCACallback() returned a SCACC_WARNING compCode	
Input Parameter	token	Token returned by prior SCAGetReference(), SCAGetReferences(), SCASelf(), or SCAGetCallback() call.
In/Out Parameter	bufferLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	buffer	Response message
	compCode	<p>SCACC_OK, if the call is successful</p> <p>SCACC_WARNING, if the fault data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer.</p> <p>SCACC_ERROR, otherwise – see reason for details</p>
	reason	SCA_DATA_TRUNCATED if the fault data was truncated.
Post Condition	The referenceToken remains valid for subsequent calls.	

1189 *Table 6-6: SCAGetReplyMessage Details*

1190 6.1.6 SCAGetFaultMessage

1191 A C component implementation uses SCAGetFaultMessage() to retrieve the details of a business fault
 1192 received in response to an operation invocation.

Precondition	C component instance is running, has a valid token and an <code>SCAInvoke()</code> returned a <code>SCACC_FAULT compCode</code>	
Input Parameter	<code>token</code>	Token returned by prior <code>SCAGetReference()</code> , <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
In/Out Parameter	<code>bufferLen</code>	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	<code>faultName</code>	Name of the business fault
	<code>buffer</code>	Fault message
	<code>compCode</code>	<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
	<code>reason</code>	<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.	

1193 *Table 6-7: SCAGetFaultMessage Details*

1194 6.1.7 SCASetFaultMessage

1195 A C component implementation uses `SCASetFaultMessage()` to return a business fault in response to
1196 a request.

Precondition	C component instance is running	
Input Parameters	<code>serviceName</code>	Name of the Service of the component for which the fault is being returned
	<code>operationName</code>	Name of the operation of the Service for which the fault is being returned
	<code>faultName</code>	Name of the business fault
	<code>bufferLen</code>	Length of the fault message buffer
	<code>buffer</code>	Fault message
Output Parameters	<code>compCode</code>	<code>SCACC_OK</code> , if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
	<code>reason</code>	<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	

1197 *Table 6-8: SCASetFaultMessage Details*

1198 **6.1.8 SCASelf**

1199 A C component implementation uses `SCASelf()` to access a Service it provides.

Precondition	C component instance is running	
Input Parameter	<code>serviceName</code>	Name of the Service to access. If a component only provides one service, this string can be empty.
Output Parameters	<code>serviceToken</code>	Token to be used in subsequent <code>SCAInvoke()</code> calls. This will be NULL if <code>serviceName</code> is not defined for the component.
	<code>compCode</code>	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.	

1200 *Table 6-9: SCASelf Details*

1201 **6.1.9 SCAGetCallback**

1202 A C component implementation uses `SCAGetCallback()` to initialize a Service before invoking any
1203 callback operations of the Service.

Precondition	C component instance is running	
Input Parameter	<code>serviceName</code>	Name of the Service to initialize. If a component only provides one service, this string can be empty.
Output Parameters	<code>serviceToken</code>	Token to be used in subsequent <code>SCAInvoke()</code> calls. This will be NULL if <code>serviceName</code> is not defined for the component.
	<code>compCode</code>	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.	

1204 *Table 6-10: SCAGetCallback Details*

1205 **6.1.10 SCAReleaseCallback**

1206 A C component implementation uses `SCAReleaseCallback()` to tell the SCA runtime it has completed
1207 callback processing and the `EndPointReference` can be released.

Precondition	C component instance is running and has a valid <code>serviceToken</code>	
Input Parameter	<code>serviceToken</code>	Token returned by prior <code>SCAGetCallback()</code> call.
Output Parameters	<code>compCode</code>	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.	

1208 *Table 6-11: SCAReleaseCallbacke Details*

1209 6.1.11 SCAInvokeAsync

1210 A C component implementation uses SCAInvokeAsync() to invoke a long running operation of an
 1211 interface using the asynchronous style.

Precondition	C component instance is running and has a valid token	
Input Parameters	token	Token returned by prior SCAGetReference(), SCAGetReferences(), SCASelf() or SCAGetCallback() 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	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 for the 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.	

1212 *Table 6-12: SCAInvokeAsynch Details*

1213 6.1.12 SCAInvokePoll

1214 A C component implementation uses SCAInvokePoll() to invoke a long running operation of a
 1215 Reference using the polling style.

Precondition	C component instance is running and has a valid token	
Input Parameters	token	Token returned by prior SCAGetReference(), SCAGetReferences(), SCASelf() or SCAGetCallback() 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.	

1216 Table 6-13: SCAInvokePoll Details

1217 6.1.13 SCACheckResponse

1218 A C component implementation uses SCACheckResponse() to determine if a response to a long
1219 running operation request has been received.

Precondition	C component instance is running, has a valid token and has made a SCAInvokePoll() 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	

1220 Table 6-14: SCACheckResponse Details

1221 6.1.14 SCACancelInvoke

1222 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 SCAInvokeAsync() or SCAInvokePoll() 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.
----------------	---

1223 *Table 6-15 : SCACancelInvokee Details*

1224 6.1.15 SCAEntryPoint

1225 Non-SCA C code uses `SCAEntryPoint()` to access a Service before invoking any operations of the
1226 Service.

Precondition	None	
Input Parameter	<code>serviceURI</code>	URI of the Service to access
	<code>domainURI</code>	URI of the SCA domain
Output Parameters	<code>serviceToken</code>	Token to be used in subsequent <code>SCAInvoke()</code> calls. This will be NULL if the Service cannot be found.
	<code>compCode</code>	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	<code>reason</code>	SCA_SERVICE_UNAVAILABLE if the domain does not exist of 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.	

1227 *Table 6-16: SCAEntryPoint Details*

1228 6.2 Program-Based Implementation Support

1229 Support for components implemented via C programs is provided by the functions `SCAService()`,
1230 `SCAOperation()`, `SCAMessageIn()` and `SCAMessageOut()`.

1231

```

1232 void SCAService(wchar_t *serviceName, int *compCode, int *reason);
1233
1234 void SCAOperation(wchar_t *operationName, int *compCode, int *reason);
1235
1236 void SCAMessageIn(wchar_t *serviceName,
1237                  wchar_t *operationName,
1238                  int *bufferLen,
1239                  void *buffer,
1240                  int *compCode,
1241                  int *reason);
1242
1243 void SCAMessageOut(wchar_t *serviceName,
1244                   wchar_t *operationName,
1245                   int bufferLen,
1246                   void *buffer,
1247                   int *CompCode,
1248                   int *Reason);

```

1249 *Snippet 6-2: SCA API for Program Implementations Definition*

1250 6.2.1 SCAService

1251 A program-based C component implementation uses `SCAService()` to determine which service was
1252 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	

1253 *Table 6-17: SCAService Details*

1254 6.2.2 SCAOperation

1255 A program-based C component implementation uses `SCAOperation()` to determine which operation of
 1256 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.	

1257 *Table 6-18: SCAOperation Details*

1258 6.2.3 SCAMessageIn

1259 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.	

1260 *Table 6-19: SCAaMessgeln Details*

1261 6.2.4 SCAMessageOut

1262 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 SCAService().
	operationName	Name returned by SCAOperation().
	bufferLen	Length of the reply message buffer
	buffer	Reply message
Output Parameters	compCode	SCACC_OK
	reason	
Post Condition	The component normally ends processing.	

1263 Table 6-20: SCAMessgeOut Details

1264 7 C Contributions

1265 Contributions are defined in the Assembly specification [ASSEMBLY] C contributions are typically, but
1266 not necessarily contained in .zip files. In addition to SCDL and potentially WSDL artifacts, C contributions
1267 include binary executable files, componentType files and potentially C interface headers. No additional
1268 discussion is needed for header files, but there are additional considerations for executable and
1269 componentType files discussed.

1270 7.1 Executable files

1271 Executable files containing the C implementations for a contribution can be contained in the contribution,
1272 contained in another contribution or external to any contribution. In some cases, it could be desirable to
1273 have contributions share an executable. In other cases, an implementation deployment policy might
1274 dictate that executables are placed in specific directories in a file system.

1275 7.1.1 Executable in contribution

1276 When the executable file containing a C implementation is in the same contribution, the *@path* attribute of
1277 the *implementation.c* element is used to specify the location of the executable. The specific location of an
1278 executable within a contribution is not defined by this specification.

1279 Snippet 7-1 shows a contribution containing a DLL.

1280

```
1281 META-INF/  
1282   sca-contribution.xml  
1283 bin/  
1284   autoinsurance.dll  
1285 AutoInsurance/  
1286   AutoInsurance.composite  
1287   AutoInsuranceService/  
1288     AutoInsurance.h  
1289     AutoInsurance.componentType  
1290 include/  
1291   Customers.h  
1292   Underwriting.h  
1293   RateUtils.h
```

1294 *Snippet 7-1: Contribution Containing a DLL*

1295

1296 The SCDL for the AutoInsuranceService component of Snippet 7-1 is:

1297

```
1298 <component name="AutoInsuranceService">  
1299   <implementation.c module="autoinsurance" path="bin/"  
1300     componentType="AutoInsurance" />  
1301 </component>
```

1302 *Snippet 7-2: Component Definition Using Implementation in a Common DLL*

1303 7.1.2 Executable shared with other contribution(s) (Export)

1304 If a contribution contains an executable that also implements C components found in other contributions,
1305 the contribution has to export the executable. An executable in a contribution is made visible to other
1306 contributions by adding an *export.c* element to the contribution definition as shown in Snippet 7-3.

1307

```
1308 <contribution>  
1309   <deployable composite="myNS:RateUtilities"
```

```
1310     <export.c name="contribNS:rates" >
1311     </contribution>
```

1312 *Snippet 7-3: Exporting a Contribution*

1313

1314 It is also possible to export only a subtree of a contribution. For a contribution:

1315

```
1316     META-INF/
1317         sca-contribution.xml
1318     bin/
1319         rates.dll
1320     RateUtilities/
1321         RateUtilities.composite
1322         RateUtilitiesService/
1323             RateUtils.h
1324             RateUtils.componentType
```

1325 *Snippet 7-4: Contribution with a Subdirectory to be Shared*

1326

1327 An export of the form:

1328

```
1329     <contribution>
1330         <deployable composite="myNS:RateUtilities"
1331         <export.c name="contribNS:ratesbin" path="bin/" >
1332     </contribution>
```

1333 *Snippet 7-5: Exporting a Subdirectory of a Contribution*

1334

1335 only makes the contents of the bin directory visible to other contributions. By placing all of the executable
1336 files of a contribution in a single directory and exporting only that directory, the amount of information
1337 available to a contribution that uses the exported executable files is limited. This is considered a best
1338 practice.

1339 7.1.3 Executable outside of contribution (Import)

1340 When the executable that implements a C component is located outside of a contribution, the contribution
1341 has to import the executable. If the executable is located in another contribution, the **import.c** element of
1342 the contribution definition uses a *@location* attribute that identifies the name of the export as defined in
1343 the contribution that defined the export as shown in Snippet 7-6.

1344

```
1345     <contribution>
1346         <deployable composite="myNS:Underwriting"
1347         <import.c name="rates" location="contribNS:rates">
1348     </contribution>
```

1349 *Snippet 7-6: Contribution with an Import*

1350

1351 The SCDL for the UnderwritingService component of Snippet 7-6 is:

1352

```
1353     <component name="UnderwritingService">
1354         <implementation.c module="rates" path="rates:bin/"
1355         componentType="Underwriting" />
1356     </component>
```

1357 *Snippet 7-7: Component Definition Using Implementation in an External DLL*

1358

1359 If the executable is located in the file system, the *@location* attribute identifies the location in the files
1360 system used as the root of the import as shown in Snippet 7-8.

1361

```
1362 <contribution>  
1363   <deployable composite="myNS:CustomerUtilities"  
1364   <import.c name="usr-bin" location="/usr/bin/" >  
1365 </contribution>
```

1366 *Snippet 7-8: Component Definition Using Implementation in a File System*

1367 7.2 componentType files

1368 As stated in Component Type and Component, each component implemented in C has a corresponding
1369 componentType file. This componentType file is, by default, located in the root directory of the composite
1370 containing the component or a subdirectory of the composite root with a name specified on the
1371 *@componentType* attribute as shown in the Snippet 7-9.

1372

```
1373 META-INF/  
1374   sca-contribution.xml  
1375 bin/  
1376   autoinsurance.dll  
1377 AutoInsurance/  
1378   AutoInsurance.composite  
1379   AutoInsuranceService/  
1380     AutoInsurance.h  
1381     AutoInsurance.componentType
```

1382 *Snippet 7-9: Contribution with ComponentType*

1383

1384 The SCDL for the AutoInsuranceService component of Snippet 7-9 is:

1385

```
1386 <component name="AutoInsuranceService">  
1387   <implementation.c module="autoinsurance" path="bin/"  
1388     componentType="AutoInsurance" />  
1389 </component>
```

1390 *Snippet 7-10: Component Definition with Local ComponentType*

1391

1392 Since there is a one-to-one correspondence between implementations and componentTypes, when an
1393 implementation is shared between contributions, it is desirable to also share the componentType file.
1394 ComponentType files can be exported and imported in the same manner as executable files. The location
1395 of a *.componentType* file can be specified using the *@componentType* attribute of the *implementation.c*
1396 element.

1397

```
1398 <component name="UnderwritingService">  
1399   <implementation.c library="rates" path="rates:bin/"  
1400     componentType="rates:types/Underwriting" />  
1401 </component>
```

1402 *Snippet 7-11: Component Definition with Imported ComponentType*

1403 7.3 C Contribution Extensions

1404 7.3.1 Export.c

1405 Snippet 7-12 shows the pseudo-schema for the C export element used to make an executable or
1406 componentType file visible outside of a contribution.

1407

1408

1409

1410

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

1411

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

1412

1413

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

1414

- **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]**

1415

1416

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

1417

1418

7.3.2 Import.c

1419

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

1420

1421

1422

1423

1424

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

1425

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

1426

1427

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

1428

- **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]**

1429

1430

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

1431

1432 8 C Interfaces

1433 A service interface can be defined by a set of C function and/or struct declarations.

1434 When mapping a C interface to WSDL or when comparing two C interfaces for compatibility, as defined
1435 by the Assembly specification **[ASSEMBLY]**, it is necessary for an SCA implementation to determine the
1436 signature (return type, name, and the names and types of the parameters) of every function or the type of
1437 every member of every struct in the service interface definition. An SCA implementation **MUST translate**
1438 **declarations to tokens as part of conversion to WSDL or compatibility testing.** **[C80001]** Snippet 8-1
1439 shows a case where a macro has to be processed to understand the return type of a function.

1440

```
1441 #if LIB_BUILD  
1442 # define DECLSPEC_FUNC(ReturnType) __declspec(dllimport) ReturnType  
1443 #else  
1444 # define DECLSPEC_FUNC(ReturnType) __declspec(dllexport) ReturnType  
1445 #endif  
1446  
1447 DECLSPEC_FUNC(int) fooFunc(void) {}
```

1448 *Snippet 8-1: Example Macro Impacting Function Signature*

1449

1450 Macros and typedefs in function or struct declarations might lead to portability problems. Complete
1451 function or struct declarations within a macro are discouraged. The processing of typedefs needs to be
1452 aware of the types that impact mapping to WSDL (see Table 9-1 and Table 9-2)

1453 8.1 Types Supported in Service Interfaces

1454 Not all service interfaces support the complete set of the types available in C.

1455 8.1.1 Local Service

1456 Any fundamental or compound type defined by C can be used in the interface of a local service.

1457 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:

- 1460 • Any of the C types specified in Simple Content Binding and Complex Content Binding. These types
1461 may be passed by-value or by-pointer. Unless the function and client indicate that they allow by-
1462 reference semantics (see `AllowsPassByReference`), a copy will be explicitly created by the runtime
1463 for any parameters passed by-pointer.
- 1464 • An SDO `DATAOBJECT`. This type may be passed by-value or by-pointer. Unless the function and
1465 client indicate that they allow by-reference semantics (see `AllowsPassByReference`), a deep-copy of
1466 the `DATAOBJECT` will be created by the runtime for any parameters passed by-value or by-pointer.
1467 When by-reference semantics are allowed, the `DATAOBJECT` handle will be passed. **[C80002]**

1468 8.2 Restrictions on C header files

1469 A C header file used to define an interface **MUST** declare at least one function or message format struct
1470 **[C80003]**

1471 Function definitions in a header file are not considered part of a service definition.

1472 9 WSDL to C and C to WSDL Mapping

1473 The SCA Client and Implementation Model for C applies the principles of the WSDL to Java and Java to
1474 WSDL mapping rules (augmented and interpreted for C as detailed in the following section) defined in the
1475 JAX-WS specification [JAXWS21] for generating remotable C interfaces from WSDL portTypes and vice
1476 versa. Use of the JAX-WS specification as a guideline for WSDL to C and C to WSDL mappings does not
1477 imply that any support for the Java language is mandated by this specification.

1478 A detailed mapping of C to WSDL types and WSDL to C types is covered in Data Binding.

1479 General rules for the application of JAX-WS to C:

- 1480 • References to Java are considered references to C.
- 1481 • References to Java classes are considered references to a collection of C functions or programs
1482 that implement an interface.
- 1483 • References to Java methods are considered references to C functions or message format struct
1484 declarations.
- 1485 • References to Java interfaces are considered references to a collection of C function or message
1486 format struct declarations used to define an interface.

1487 9.1 Interpretations for WSDL to C Mapping

1488 External binding files are not supported.

1489 For dispatching functions or invoking programs and marshalling data, an implementation can choose to
1490 interpret the WSDL document, possibly containing mapping customizations, at runtime or interpret the
1491 document as part of the deployment process generating implementation specific artifacts that represent
1492 the mapping.

1493 9.1.1 Definitions

1494 Since C has no namespace or package construct, the targetNamespace of a WSDL document is ignored
1495 by the mapping.

1496 MIME binding is not supported.

1497 9.1.2 PortType

1498 A portType maps to a set of declarations that form the C interface for the service. The form of these
1499 declarations depends on the type of the service implementation.

1500 If the implementation is a library, the declarations are one or more function declarations and potentially
1501 any necessary struct declarations corresponding to any complex XML schema types needed by
1502 messages used by operations of the portType. See Complex Content Binding for options for complex
1503 type mapping.

1504 If the implementation is contained in a program, the declarations are all struct declarations. See the next
1505 section for details.

1506 An SCA implementation **MUST** map a WSDL portType to a remotable C interface definition. [C100023]

1507 In the absence of customizations, an SCA implementation **SHOULD** map each portType to separate
1508 header file. An SCA implementation **MAY** use any sca-c:prefix binding declarations to control this
1509 mapping. [C100001] For example, all portTypes in a WSDL document with a common sca-c:prefix binding
1510 declaration could be mapped to a single header file.

1511 Header file naming is implementation dependent.

1512 9.1.3 Operations

1513 Asynchronous mapping is not supported.

1514 9.1.3.1 Operation Names

1515 WSDL operation names are only guaranteed to be unique with a portType. C requires function and struct
1516 names loaded into an address space to be distinct. The mapping of operation names to function or struct
1517 names have to take this into account.

1518 For components implemented in libraries, in the absence of customizations, an SCA implementation
1519 MUST map an operation name, with the first character converted to lower case, to a function name. If
1520 necessary, to avoid name collisions, an SCA implementation MAY prepend the portType name, with the
1521 first character converted to lower case, and the operation name, with the first character converted to
1522 upper case, to form the function name. [C100002]

1523 An application can customize this mapping using the sca-c:prefix and/or sca-c:function binding
1524 declarations.

1525 For program-based service implementations:

- 1526 • If the number of **In** parameters plus the number of **In/Out** parameters is greater than one there will be
1527 a request struct.
- 1528 • If the number of **Out** parameters plus the number of **In/Out** parameters is greater than one there will
1529 be a response struct.

1530 For components implemented in a program, in the absence of customizations, an SCA implementation
1531 MUST map an operation name, with the first character converted to lowercase to a request struct name. If
1532 necessary, to avoid name collisions, an SCA implementation MAY concatenate the portType name, with
1533 the first character converted to lower case, and the operation name, with the first character converted to
1534 upper case, to form the request struct name. Additionally an SCA implementation MUST append
1535 “Response” to the request struct name to form the response struct name. [C100005]

1536 An application can customize this mapping using the sca-c:prefix and/or sca-c:struct binding declarations.

1537 9.1.3.2 Message and Part

1538 In the absence of any customizations for a WSDL operation that does not meet the requirements for the
1539 wrapped style, the name of a mapped function parameter or struct member MUST be the value of the
1540 name attribute of the wsdl:part element with the first character converted to lower case. [C100003]

1541 In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped
1542 style, the name of a mapped function parameter or struct member MUST be the value of the local name
1543 of the wrapper child with the first character converted to lower case. [C100004]

1544 An application can customize this mapping using the sca-c:parameter binding declaration.

1545 For library-based service implementations, an SCA implementation MUST map **In** parameters as pass
1546 by-value or const and **In/Out** and **Out** parameters as pass via pointers. [C100019]

1547 For program-based service implementations, an SCA implementation MUST map all values in the input
1548 message as pass by-value and the updated values for **In/Out** parameters and all **Out** parameters in the
1549 response message as pass by-value. [C100020]

1550 9.1.4 Types

1551 As per section Data Binding (based on SDO type mapping).

1552 MTOM/XOP content processing is left to the application.

1553 9.1.5 Fault

1554 C has no exceptions so an API is provided for getting and setting fault messages (see
1555 SCAGetFaultMessage and SCASetFaultMessage). Fault messages are mapped in same manner as
1556 input and output messages.

1557 In the absence of customizations, an SCA implementation MUST map the name of the message element
1558 referred to by a fault element to the name of the struct describing the fault message content. If necessary,
1559 to avoid name collisions, an implementation MAY append “*Fault*” to the name of the message element
1560 when mapping to the struct name. [C100006]

1561 An application can customize this mapping using the `sca-c:struct` binding declaration.

1562 9.1.6 Service and Port

1563 This mapping does not define generation of client side code.

1564 9.1.7 XML Names

1565 See comments in Operations

1566 9.2 Interpretations for C to WSDL Mapping

1567 Where annotations are discussed as a means for an application to control the mapping to WSDL, an
1568 implementation-specific means of controlling the mapping can be used instead.

1569 9.2.1 Package

1570 Not relevant.

1571 An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be
1572 configurable. [C100007]

1573 9.2.2 Class

1574 Not relevant since mapping is only based on declarations.

1575 9.2.3 Interface

1576 The declarations in a header file are used to define an interface. A header file can be used to define an
1577 interface if it satisfies either (for components implemented in libraries):

- 1578 • Contains one or more function declarations
- 1579 • Any of these functions declarations might carry a `@WebFunction` annotation
- 1580 • The parameters and return types of these function declarations are compatible with the C to XML
1581 Schema mapping in Data Binding

1582 or (for components implemented in programs):

- 1583 • Contains one request message struct declarations
- 1584 • Any of the request message struct declarations might carry a `@WebOperation` annotation
- 1585 • Any of the request message struct declarations can have a corresponding response message struct,
1586 identified by either having a name with “*Response*” appended to the request message struct name or
1587 identified in a `@WebOperation` annotation
- 1588 • Members of these struct declarations are compatible with the C to XML Schema mapping in Data
1589 Binding

1590 An SCA implementation MUST map a C interface definition to WSDL as if it has a `@WebService`
1591 annotation with all default values. [C100024]

1592 In the absence of customizations, an SCA implementation MUST map the header file name to the
1593 `portType` name. An implementation MAY append “*PortType*” to the header file name in the mapping to the
1594 `portType` name. [C100008]

1595 An application can customize this mapping using the `@WebService` annotation.

1596 9.2.4 Method

1597 For components implemented in libraries, functions map to operations.

1598 In the absence of customizations, an SCA implementation MUST map a function name to an operation
1599 name, stripping the portType name, if present and any namespace prefix from the front of function name
1600 before mapping it to the operation name. [C100009]

1601 An application can customize function to operation mapping or exclude a function from an interface using
1602 the @WebFunction annotation.

1603 For components implemented in programs, operations are mapped from request structs.

1604 In the absence of customizations, a struct with a name that does not end in "Response" or "Fault" is
1605 considered to be a request message struct and an SCA implementation MUST map the struct name to
1606 the operation name, stripping the portType name, if present, and any namespace prefix from the front of
1607 the struct name before mapping it to the operation name. [C100010]

1608 An application can customize struct to operation mapping or exclude a struct from an interface using the
1609 @WebOperation annotation.

1610 9.2.5 Method Parameters and Return Type

1611 For components implemented in libraries, function parameters and return type map to either message or
1612 global element components.

1613 In the absence of customizations, an SCA implementation MUST map a parameter name, if present, to a
1614 part or global element component name. If the parameter does not have a name the SCA implementation
1615 MUST use argN as the part or global element child name. [C100011]

1616 An application can customize parameter to message or global element component mapping using the
1617 @WebParam annotation.

1618 In the absence of customizations, an SCA implementation MUST map the return type to a part or global
1619 element child named "return". [C100012]

1620 An application can customize return type to message or global element component mapping using the
1621 @WebReturn annotation.

1622 An SCA implementation MUST map:

- 1623 • a function's return value as an **out** parameter.
- 1624 • by-value and const parameters as **in** parameters.
- 1625 • in the absence of customizations, pointer parameters as **in/out** parameters. [C100017]

1626 An application can customize parameter classification using the @WebParam annotation.

1627 Program based implementation SHOULD use the Document-Literal style and encoding. [C100013]

1628 In the absence of customizations, an SCA implementation MUST map the struct member name to the
1629 part or global element child name. [C100014]

1630 An application can customize struct member to message or global element component mapping using the
1631 @WebParam annotation.

- 1632 • Members of the request struct that are not members of the response struct are **in** parameters
- 1633 • Members of the response struct that are not members of the request struct are **out** parameters
- 1634 • Members of both the request and response structs are **in/out** parameters. Matching is done by
1635 member name. An SCA implementation MUST ensure that **in/out** parameters have the same type in
1636 the request and response structs. [C100015]

1637 9.2.6 Service Specific Exception

1638 C has no exceptions. A struct can be annotated as a fault message type. A function or operation
1639 declaration can be annotated to indicate that it potentially generates a specific fault.

1640 An application can define a fault message format using the @WebFault annotation.

1641 An application can indicate that a WSDL fault might be generated by a function or operation using the
1642 @WebThrows annotation.

1643 9.2.7 Generics

1644 Not relevant.

1645 9.3 Data Binding

1646 The data in wsdl:parts or wrapper children is mapped to and from C function parameters and return
1647 values (for library-based component implementations), or struct members (for program-based component
1648 implementations and fault messages).

1649 9.3.1 Simple Content Binding

1650 The mapping between XSD simple content types and C types follows the convention defined in the SDO
1651 specification [SDO21]. Table 9-1 summarizes that mapping as it applies to SCA services.
1652

<i>XSD Schema Type →</i>	<i>C Type</i>	<i>→ XSD Schema Type</i>
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

1653 *Table 9-1: XSD simple type to C type mapping*

1654

1655 Table 9-2 defines the mapping of C++ types to XSD schema types that are not covered in Table 9-1.

1656

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

1657 *Table 9-2: C type to XSD type mapping*

1658

1659 The C standard does not define value ranges for integer types so it is possible that on a platform
 1660 parameters or return values could have values that are out of range for the default XSD schema type. In
 1661 these circumstances, the mapping would need to be customized, using @WebParam or @WebResult if
 1662 supported, or some other implementation-specific mechanism.

1663 An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.

1664 [C100021]

1665 An SCA implementation MAY map boolean to _Bool by default. [C100022]

1666 9.3.1.1 WSDL to C Mapping Details

1667 In general, when `xsd:string` and types derived from `xsd:string` map to a struct member, the
 1668 mapping is to a combination of a `wchar_t *` and a separately allocated data array. If either the `length`
 1669 or `maxLength` facet is used, then a `wchar_t[]` is used. If the `pattern` facet is used, this might allow
 1670 the use of `char` and/or also constrain the length.

1671 Example:

```
1672 <xsd:element name="myString" type="xsd:string"/>
```

1673 maps to:

```
1674 wchar_t *myString;  
1675 /* this points to a dynamically allocated buffer with the data */
```

1676 *Snippet 9-1: Unbounded String Mapping*

1677

```
1678 <xsd:simpleType name="boundedString25">  
1679 <xsd:restriction base="xsd:string">  
1680 <xsd:length value="25"/>  
1681 </xsd:restriction>  
1682 </xsd:simpleType>
```

1683 ...

1684 `<xsd:element name="myString" type="boundedString25" />`

1685 maps to:

1686 `wchar_t myString[26];`

1687 *Snippet 9-2: Bounded String Mapping*

1688

- 1689 • When unbounded binary data maps to a struct member, the mapping is to a `char *` that points to
1690 the location where the actual data is located. Like strings, if the binary data is bounded in length, a
1691 `char[]` is used.

1692 Examples:

1693 `<xsd:element name="myData" type="xsd:hexBinary" />`

1694 maps to:

1695 `char *myData;`

1696 `/* this points to a dynamically allocated buffer with the data */`

1697 *Snippet 9-3: Unbounded Binary Data Mapping*

1698

```
1699 <xsd:simpleType name="boundedData25">  
1700 <xsd:restriction base="xsd:hexBinary">  
1701 <xsd:length value="25"/>  
1702 </xsd:restriction>  
1703 </xsd:simpleType>
```

```
1704 ...  
1705 <xsd:element name="myData" type="boundedData25" />
```

1706 maps to:

1707 `char myData[26];`

1708 *Snippet 9-4: Bounded Binary Data Mapping*

1709

- 1710 • Since C does not have a way of representing unset values, when elements with `minOccurs !=`
1711 `maxOccurs` and lists with `minLength != maxLength`, which have a variable, but bounded, number
1712 of instances, map to a struct, the mapping is to a count of the number of occurrences and an array. If
1713 the count is 0, then the content of the array is undefined.

1714 Examples:

1715 `<xsd:element name="counts" type="xsd:int" maxOccurs="5" />`

1716 maps to:

1717 `size_t counts_num;`

1718 `int counts[5];`

1719 *Snippet 9-5: minOccurs != maxOccurs Mapping*

1720

```
1721 <xsd:simpleType name="lineNumList">  
1722 <xsd:list itemType="xsd:int" />  
1723 </xsd:simpleType>  
1724 <xsd:simpleType name="lineNumList6">  
1725 <xsd:restriction base="lineNumList ">  
1726 <xsd:minLength value="1"/>  
1727 <xsd:maxLength value="6"/>  
1728 </xsd:restriction>  
1729 </xsd:simpleType>  
1730 ...  
1731 <xsd:element name="lineNums" type="lineNumList6" />
```

1732 maps to:

1733 `size_t lineNums_num;`

1734 `long lineNums[6];`

1735 *Snippet 9-6: minLength != maxLength Mapping*

1736

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

1740 Examples:

1741 `<xsd:element name="counts" type="xsd:int" maxOccurs="unbounded" />`

1742 maps to:

1743 `size_t counts_num;`

1744 `int *counts;`

1745 `/* this points to a dynamically allocated array of longs */`

1746 *Snippet 9-7: Unbounded Array Mapping*

1747

- Union Types are not supported.

1749 9.3.1.2 C to WSDL Mapping Details

- `wchar_t[]` and `char[]` map to `xsd:string` with a `maxLength` facet.

- C arrays map as normal elements but with multiplicity allowed via the `minOccurs` and `maxOccurs` facets.

1753 Example:

1754 `int idList[];`

1755 maps to:

1756 `<xsd:element name="idList" type="xsd:int"`

1757 `minOccurs="0" maxOccurs="unbounded" />`

1758 *Snippet 9-8: Array Mapping*

1759

- Multi-dimensional arrays map into nested elements.

1761 Example:

1762 `int multiIdArray[4][2];`

1763 maps to:

1764 `<xsd:element name="multiIdArray"`

1765 `minOccurs="0" maxOccurs="4" />`

1766 `<xsd:complexType>`

1767 `<xsd:sequence>`

1768 `<xsd:element name="multiIdArray" type="xsd:int"`

1769 `minOccurs="2" maxOccurs="2" />`

1770 `</xsd:sequence>`

1771 `</xsd:complexType>`

1772 `</xsd:element>`

1773 *Snippet 9-9: Multi-Dimensional Array Mapping*

1774

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

1777 9.3.2 Complex Content Binding

1778 When mapping between XSD complex content types and C, either instances of SDO DataObjects or
1779 structs are used. An SCA implementation MUST support mapping message parts or global elements with
1780 complex types and parameters, return types and struct members with a type defined by a struct. The
1781 mapping from WSDL MAY be to DataObjects and/or structs. The mapping to and from structs MUST
1782 follow the rules defined in WSDL to C Mapping Details. [C100016]

1783 9.3.2.1 WSDL to C Mapping Details

- 1784 • Complex types and groups mapped to static DataObjects follow the rules defined in [SDO21].
- 1785 • Complex types and groups mapped to structs have the attributes and elements of the type mapped
1786 to members of the struct.
 - 1787 – The name of the struct is the name of the type or group.
 - 1788 – Attributes appear in the struct before elements.
 - 1789 – Simple types are mapped to members as described above.
 - 1790 – The same rules for variable number of instances of a simple type element apply to complex type
1791 elements.
 - 1792 – A sequence group is mapped as either a simple type or a complex type as appropriate.

1793 Example:

```
1794 <xsd:complexType name="myType">  
1795   <xsd:sequence>  
1796     <xsd:element name="name">  
1797       <xsd:simpleType>  
1798         <xsd:restriction base="xsd:string">  
1799           <xsd:length value="25"/>  
1800         </xsd:restriction>  
1801       </xsd:simpleType>  
1802     </xsd:element>  
1803     <xsd:element name="idList" type="xsd:int"  
1804               minOccurs="0" maxOccurs="unbounded"/>  
1805     <xsd:element name="value" type="xsd:double"/>  
1806   </xsd:sequence>  
1807 </xsd:complexType>
```

1808 maps to:

```
1809 struct myType {  
1810     wchar_t name[26];  
1811     size_t idList_num;  
1812     long *idList;  
1813     /* this points to a dynamically allocated array of longs */  
1814     double value;  
1815 };
```

1816 *Snippet 9-10: Sequence Group Mapping*

1817

- 1818 • While XML Schema allow the elements of an all group to appear in any order, the order is fixed in
1819 the C mapping. Each child of an all group is mapped as pointer to the value and value itself. If the
1820 child is not present, the pointer is NULL and the value is undefined.

1821 Example:

```
1822 <xsd:element name="myVariable">  
1823   <xsd:complexType name="myType">  
1824     <xsd:all>  
1825       <xsd:element name="name" type="xsd:string"/>  
1826       <xsd:element name="idList" type="xsd:int"  
1827                 minOccurs="0" maxOccurs="unbounded"/>
```

```
1828     <xsd:element name="value" type="xsd:double"/>
1829     </xsd:all>
1830 </xsd:complexType>
1831 </xsd:element>
```

1832 maps to:

```
1833 struct myType {
1834     wchar_t *name;
1835     /* this points to a dynamically allocated string */
1836     size_t idList_num;
1837     long *idList;
1838     /* this points to a dynamically allocated array of longs */
1839     double *value;
1840     /* this points to a dynamically allocated long */
1841 } *pmyVariable, myVariable;
```

1842 *Snippet 9-11: All Group Mapping*

1843

- 1844 • Handing of choice groups is not defined by this mapping, and is implementation dependent. For
1845 portability, choice groups are discouraged in service interfaces.
- 1846 • Nillable elements are mapped to a pointer to the value and the value itself. If the element is not
1847 present, the pointer is NULL and the value is undefined.

1848 Example:

```
1849 <xsd:element name="priority" type="xsd:short" nillable="true"/>
```

1850 maps to:

```
1851 int16_t *pprioiry, priority;
```

1852 *Snippet 9-12: Nillable Mapping*

1853

- 1854 • Mixed content and open content (Any Attribute and Any Element) is supported via DataObjects.

1855 9.3.2.2 C to WSDL Mapping Details

- 1856 • C structs that contain types that can be mapped, are themselves mapped to complex types.

1857 Example:

```
1858 struct DataStruct {
1859     char *name;
1860     double value;
1861 };
```

1862 maps to:

```
1863 <xsd:complexType name="DataStruct">
1864     <xsd:sequence>
1865         <xsd:element name="name" type="xsd:string"/>
1866         <xsd:element name="value" type="xsd:double"/>
1867     </xsd:sequence>
1868 </xsd:complexType>
```

1869 *Snippet 9-13: Struct Mapping*

1870

- 1871 • char and wchar_t arrays inside of structs are mapped to a restricted subtype of xsd:string that
1872 limits the length the space allowed in the array.

1873 Example:

```
1874 struct DataStruct {
1875     char name[256];
1876     double value;
```

```

1877     };
1878 maps to:
1879     <xsd:complexType name="DataStruct">
1880     <xsd:sequence>
1881     <xsd:element name="name">
1882     <xsd:simpleType>
1883     <xsd:restriction base="xsd:string">
1884     <xsd:maxLength value="255"/>
1885     </xsd:restriction>
1886     </xsd:simpleType>
1887     </xsd:element>
1888     <xsd:element name="value" type="xsd:double"/>
1889     </xsd:sequence>
1890 </xsd:complexType>

```

1891 *Snippet 9-14: Character Array in Struct Mapping*

1892

- 1893 • C enums define a list of named symbols that map to values. If a function uses an enum type, this is
- 1894 mapped to a restricted element in the WSDL schema.

1895 Example:

```

1896 enum ParameterType {
1897     UNSET = 1,
1898     TYPEA,
1899     TYPEB,
1900     TYPEC
1901 };

```

1902 maps to:

```

1903 <xsd:simpleType name="ParameterType">
1904 <xsd:restriction base="xsd:int">
1905 <xs:minInclusive value="1"/>
1906 <xs:maxInclusive value="4"/>
1907 </xsd:restriction>
1908 </xsd:simpleType>

```

1909 *Snippet 9-15: Enum Mapping*

1910

1911 The restriction used will have to be appropriate to the values of the enum elements.

1912 Example:

```

1913 enum ParameterType {
1914     UNSET = 'u',
1915     TYPEA = 'A',
1916     TYPEB = 'B',
1917     TYPEC = 'C'
1918 };

```

1919 maps to:

```

1920 <xsd:simpleType name="ParameterType">
1921 <xsd:restriction base="xsd:int">
1922 <xsd:enumeration value="86"/> <!-- Character 'u' -->
1923 <xsd:enumeration value="65"/> <!-- Character 'A' -->
1924 <xsd:enumeration value="66"/> <!-- Character 'B' -->
1925 <xsd:enumeration value="67"/> <!-- Character 'C' -->
1926 </xsd:restriction>
1927 </xsd:simpleType>

```

1928 *Snippet 9-16: Non-contiguous Value Enum Mapping*

1929

- 1930 • If a struct or enum contains other structs or enums, the mapping rules are applied recursively.

1931 Example:

1932 `struct DataStruct data;`
1933 with types defined as follows:

```
1934 struct DataStruct {  
1935     char name[30];  
1936     double values[20];  
1937     ParameterType type;  
1938 };  
1939  
1940 enum ParameterType {  
1941     UNSET = 1,  
1942     TYPEA,  
1943     TYPEB,  
1944     TYPEC  
1945 };
```

1946 maps to:

```
1947 <xsd:complexType name="DataStruct">  
1948     <xsd:sequence>  
1949         <xsd:element name="name">  
1950             <xsd:simpleType>  
1951                 <xsd:restriction base="xsd:string">  
1952                     <xsd:maxLength value="29"/>  
1953                 </xsd:restriction>  
1954             </xsd:simpleType>  
1955         </xsd:element>  
1956         <xsd:element name="values" type="xsd:double" minOccurs=20 maxOccurs=20/>  
1957         <xsd:element name="type" type="ParameterType"/>  
1958     </xsd:sequence>  
1959 </xsd:complexType>  
1960  
1961 <xsd:simpleType name="ParameterType">  
1962     <xsd:restriction base="xsd:int">  
1963         <xs:minInclusive value="1"/>  
1964         <xs:maxInclusive value="4"/>  
1965     </xsd:restriction>  
1966 </xsd:simpleType>
```

1967 *Snippet 9-17: Nested Struct Mapping*

1968

- 1969 • Mapping of C unions is not supported by this specification.
- 1970 • Typedefs are resolved when evaluating parameter and return types. Typedefs are resolved before
1971 the mapping to Schema is done.

1972 10 Conformance

1973 The XML schema pointed to by the RDDDL document at the SCA namespace URI, defined by the
1974 Assembly specification [ASSEMBLY] and extended by this specification, are considered to be
1975 authoritative and take precedence over the XML schema in this document.

1976 The XML schema pointed to by the RDDDL document at the SCA C namespace URI, defined by this
1977 specification, is considered to be authoritative and takes precedence over the XML schema in this
1978 document.

1979 Normative code artifacts related to this specification are considered to be authoritative and take
1980 precedence over specification text.

1981 An SCA implementation MUST reject a composite file that does not conform to [http://docs.oasis-
1983 open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd](http://docs.oasis-
1982 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](http://docs.oasis-
open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd). [C110001]

1984 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](http://docs.oasis-
1985 open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd). [C110002]

1986 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](http://docs.oasis-
1987 open.org/opencsa/sca/200912/sca-contribution-c-1.1.xsd). [C110003]

1988 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-wsdlex-c-1.1.xsd](http://docs.oasis-
1989 open.org/opencsa/sca-c-cpp/c/200901/sca-wsdlex-c-1.1.xsd). [C110004]

1990 10.1 Conformance Targets

1991 The conformance targets of this specification are:

- 1992 • **SCA implementations**, which provide a **runtime** for SCA components and potentially **tools** for
1993 authoring SCA artifacts, component descriptions and/or runtime operations.
- 1994 • **SCA documents**, which describe SCA artifacts, and specific **elements** within these documents.
- 1995 • **C files**, which define SCA service interfaces and implementations.
- 1996 • **WSDL files**, which define SCA service interfaces.

1997 10.2 SCA Implementations

1998 An implementation conforms to this specification if it meets these conditions:

- 1999 1. It MUST conform to the SCA Assembly Model Specification [ASSEMBLY] and the SCA Policy
2000 Framework [POLICY].
- 2001 2. It MUST comply with all statements in Table F-1 and Table F-5 related to an SCA implementation,
2002 notably all mandatory statements have to be implemented.
- 2003 3. It MUST implement the SCA C API defined in section SCA Programming Interface.
- 2004 4. It MAY support program-based component implementations. If program-based component
2005 implementations are supported, the implementation MUST implement the Program-Based
2006 Implementation Support API defined in Program-Based Implementation Support and MUST comply
2007 with all statements in Table F-2 related to an SCA implementation, notably all mandatory statements
2008 in that section have to be implemented.
- 2009 5. It MUST implement the mapping between C and WSDL 1.1 [WSDL11] defined in WSDL to C and C
2010 to WSDL Mapping.
- 2011 6. It MUST support <interface.c/> and <implementation.c/> elements as defined in Component Type
2012 and Component in composite and componentType and documents.
- 2013 7. It MUST support <export.c/> and <import.c/> elements as defined in C Contributions in contribution
2014 documents.

- 2015 8. It MAY support source file annotations as defined in, C SCA Annotations, C SCA Policy Annotations
2016 and C WSDL Annotations. If source file annotations are supported, the implementation MUST comply
2017 with all statements in Table F-3 related to an SCA implementation, notably all mandatory statements
2018 in that section have to be implemented.
- 2019 9. It MAY support WSDL extensions as defined in C WSDL Mapping Extensions. If WSDL extensions
2020 are supported, the implementation MUST comply with all statements in Table F-4 related to an SCA
2021 implementation, notably all mandatory statements in that section have to be implemented.

2022 10.3 SCA Documents

2023 An SCA document conforms to this specification if it meets these conditions:

- 2024 1. It MUST conform to the SCA Assembly Model Specification [**ASSEMBLY**] and, if appropriate, the
2025 SCA Policy Framework [**POLICY**].

- 2026 2. If it is a composite document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd)
2027 [open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd) and [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd)
2028 [open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd) schema and MUST comply with the
2029 additional constraints on the document contents as defined in Table F-1.

2030 If it is a componentType document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd)
2031 [open.org/opencsa/sca/200912/sca-interface-c-1.1.xsd](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-](http://docs.oasis-open.org/opencsa/sca/200912/sca-contribution-c-1.1.xsd)
2034 [open.org/opencsa/sca/200912/sca-contribution-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200912/sca-contribution-c-1.1.xsd) schema and MUST comply with the
2035 additional constraints on the document contents as defined in Table F-1.

2036 10.4 C Files

2037 A C file conforms to this specification if it meets the conditions:

- 2038 1. It MUST comply with all statements in, Table F-1, Table F-3 and Table F-5 related to C contents and
2039 annotations, notably all mandatory statements have to be satisfied.

2040 10.5 WSDL Files

2041 A WSDL conforms to this specification if it meets these conditions:

- 2042 1. It is a valid WSDL 1.1 [**WSDL11**] document.
- 2043 2. It MUST comply with all statements in Table F-1, Table F-4 and Table F-5 related to WSDL contents
2044 and extensions, notably all mandatory statements have to be satisfied.

2045

A C SCA Annotations

2046 To allow developers to define SCA related information directly in source files, without having to separately
2047 author SCDL files, a set of annotations is defined. If SCA annotations are supported by an
2048 implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as
2049 described. The SCA runtime MUST only process the SCDL files and not the annotations. [CA0001]

2050

A.1 Application of Annotations to C Program Elements

2051 In general an annotation immediately precedes the program element it applies to. If multiple annotations
2052 apply to a program element, all of the annotations SHOULD be in the same comment block. [CA0002]

2053

- Function or Function Prototype

2054

The annotation immediately precedes the function definition or declaration.

2055

Example:

2056

2057

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

2058

Snippet A-1: Example Function Annotation

2059

- Variable

2060

The annotation immediately precedes the variable definition.

2061

Example:

2062

2063

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

2064

Snippet A-2: Example Variable Annotation

2065

- Set of Functions Implementing a Service

2066

A set of functions implementing a service begins with an @Service annotations. Any annotations

2067

applying to this service as a whole immediately precede the @Service annotation. These annotations

2068

SHOULD be in the same comment block as the @Service annotation.

2069

Example:

2070

2071

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

2072

Snippet A-3: Example Set of Functions Annotation

2073

- Set of Function Prototypes Defining an Interface

2074

To avoid any ambiguity about the application of an annotation to a specific function or the set of

2075

functions defining an interface, if an annotation is to apply to the interface as a whole, then the

2076

@Interface annotation is used, even in the case where there is just one interface defined in a header

2077

file. Any annotations applying to the interface immediately precede the @Interface annotation.

2078

2079

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

2080

Snippet A-4: Example Set of Function Declarations Annotation

2081

A.2 Interface Header Annotations

2082

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

2083

A.2.1 @Interface

2084

Annotation that indicates the start of a new interface definition.

2085 **Corresponds to:** *interface.c* element

2086 **Format:**

```
2087 /* @Interface(name="serviceName") */
```

2088 *Snippet A-5: @Interface Annotation Format*

2089 where

- 2090 • **name : NCName (0..1)** – specifies the name of a service using this interface. The default is the root
2091 name of the header file containing the annotation.

2092 **Applies to:** Set of functions defining an interface.

2093 Function declarations following this annotation form the definition of this interface. This annotation also
2094 serves to bound the scope of the remaining annotations in this section,

2095 **Example:**

2096 Interface header:

```
2097 /* @Interface(name="LoanService") */
```

2098

2099 Service definition:

```
2100 <service name="LoanService">  
2101   <interface.c header="loans.h" />  
2102 </service>
```

2103 *Snippet A-6: Example of @Interface Annotation*

2104 **A.2.2 @Function**

2105 Annotation that indicates that a function defines an operation of a service. There are two formats for this
2106 annotation depending on if the service is implemented as a set of subroutines or in a program. An SCA
2107 implementation MUST treat a function with a @WebFunction annotation specified as if @Function was
2108 specified with the operationName value of the @WebFunction annotation used as the name value of the
2109 @Function annotation and the exclude value of the @WebFunction annotation used as the exclude value
2110 of the @Function annotation. [CA0004]

2111 **Corresponds to:** *function* or *callbackFunction* child element of an *interface.c* element. If the file the
2112 function is contained in is being processed because it was identified via either
2113 *interface.c/@callbackHeader* or a @Callback annotation, then the @Function annotation corresponds to
2114 a *callbackFunction* element, otherwise it corresponds to a *function* element.

2115 **Format:**

```
2116 /* @Function(name="operationName", exclude="true") */
```

2117 *Snippet A-7: @Operation Annotation Format for Functions*

2118 where

- 2119 • **name : NCName (0..1)** – specifies the name of the operation. The default operation name is the
2120 function name.
- 2121 • **exclude : boolean (0..1)** – specifies whether this function is to be excluded from the SCA interface.
2122 Default is **false**.

2123 **Applies to:** Function declaration

2124 **Example:**

2125 Interface header (loans.h):

```
2126 short internalFcn(char *param1, short param2);  
2127  
2128 /* @Function(name="getRate") */  
2129 void rateFcn(char *cust, float *rate);
```

2130

2131 Interface definition:

```
2132 <interface.c header="loans.h">
2133   <function name="getRate" />
2134 </interface.c>
```

2135 *Snippet A-8: Example of @Operation Annotation for Functions*

2136 A.2.3 @Operation

2137 Annotation that indicates a struct declaration defines a request message format of an operation of a
2138 service. An SCA implementation MUST treat a struct with a @WebOperation annotation specified as if
2139 @Operation was specified with the operationName value of the @WebOperation annotation used as the
2140 name value of the @Operation annotation, the response value of the @WebOperation annotation used
2141 as the response value of the @Operation annotation and the exclude value of the @WebFunction
2142 annotation used as the exclude value of the @Operation annotation. [CA0005]

2143 **Corresponds to:** *function* or *callbackFunction* child element of an *interface.c* element. If the file the struct
2144 is contained in is being processed because it was identified via either *interface.c/@callbackHeader* or a
2145 @Callback annotation, then the @Operation annotation corresponds to a *callbackFunction* element,
2146 otherwise it corresponds to a *function* element.

2147 **Format:**

```
2148 /* @Operation(name="operationName", response="outStruct", exclude="true") */
```

2149 *Snippet A-9: @Operation Annotation Format for Structs*

2150 where

- 2151 • **name: NCName (1..1)** – specifies the name of the operation. The default operation name is the name
2152 of the request message struct.
- 2153 • **response : NCName (0..1)** – specifies the name of a struct that defined the format of the response
2154 message if one is used.
- 2155 • **exclude : boolean (0..1)** – specifies whether this struct is to be excluded from the SCA interface.
2156 Default is **false**.

2157 **Applies to:** struct declarations

2158 Example:

2159 Interface header (loans.h):

```
2160 /* @Operation(name="getRate", response="rateOutput") */
2161 struct rateInput {
2162     char cust[25];
2163     int term;
2164 };
2165 struct rateOutput {
2166     float rate;
2167     int rateClass;
2168 };
```

2169

2170 Interface definition:

```
2171 <interface.c header="loans.h">
2172   <function name="getRate" input="rateInput" output="rateOutput" />
2173 </interface.c>
```

2174 *Snippet A-10: Example of @Operation Annotation for Structs*

2175 A.2.4 @Remotable

2176 Annotation on service interface to indicate that a service is remotable and implies an @Interface
2177 annotation applies as well. An SCA implementation MUST treat a file with a @WebService annotation
2178 specified as if @Remotable and @Interface were specified with the name value of the @WebService
2179 annotation used as the name value of the @Interface annotation. [CA0003]

2180 **Corresponds to:** @remotable="true" attribute of an *interface.c* element.

2181 **Format:**

```
2182 /* @Remotable */
```

2183 *Snippet A-11: @Remotable Annotation Format*

2184 The default is **false** (not remotable).

2185 **Applies to:** Interface

2186 **Example:**

2187 Interface header (LoanService.h):

```
2188 /* @Remotable */
```

2189

2190 Service definition:

```
2191 <service name="LoanService">  
2192   <interface.c header="LoanService.h" remotable="true" />  
2193 </service>
```

2194 *Snippet A-12: Example of @Remotable Annotation*

2195 A.2.5 @Callback

2196 Annotation on a service interface to specify the callback interface.

2197 **Corresponds to:** @callbackHeader attribute of an *interface.c* element.

2198 **Format:**

```
2199 /* @Callback(header="headerName") */
```

2200 *Snippet A-13: @Callback Annotation Format*

2201 where

2202 • **header : Name (1..1)** – specifies the name of the header defining the callback service interface.

2203 **Applies to:** Interface

2204 **Example:**

2205 Interface header (MyService.h):

```
2206 /* @Callback(header="MyServiceCallback.h") */
```

2207

2208 Service definition:

```
2209 <service name="MyService">  
2210   <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h" />  
2211 </service>
```

2212 *Snippet A-14: Example of @Callback Annotation*

2213 A.2.6 @OneWay

2214 Annotation on a service interface function declaration to indicate the function is one way. The @OneWay
2215 annotation also affects the representation of a service in WSDL. See @OneWay.

2216 **Corresponds to:** @oneWay="true" attribute of function element of an *interface.c* element.

2217 **Format:**

```
2218 /* @OneWay */
```

2219 *Snippet A-15: @OneWay Annotation Format*

2220 The default is **false** (not OneWay).

2221 **Applies to:** Function declaration

2222 **Example:**

2223 Interface header:

```
2224 /* @OneWay */
2225 reportEvent(int eventID);
```

2226

2227 Service definition:

```
2228 <service name="LoanService">
2229   <interface.c header="LoanService.h">
2230     <function name="reportEvent" oneWay="true" />
2231   </interface.c>
2232 </service>
```

2233 *Snippet A-16: Example of @OneWay Annotation*

2234 **A.3 Implementation Annotations**

2235 This section lists the annotations that can be used in the file that implements a service.

2236 **A.3.1 @ComponentType**

2237 Annotation used to indicate the start of a new componentType.

2238 **Corresponds to:** *@componentType* attribute of an *implementation.c* element.

2239 **Format:**

```
2240 /* @ComponentType */
```

2241 *Snippet A-17: @ComponetType Annotation Format*

2242 **Applies to:** Set of services, references and properties

2243 **Example:**

2244 Implementation:

```
2245 /* @ComponentType */
```

2246

2247 Component definition:

```
2248 <component name="LoanService">
2249   <implementation.c module="loan" componentType="LoanService" />
2250 </component>
```

2251 *Snippet A-18: Example of @ComponentType Annotation*

2252 **A.3.2 @Service**

2253 Annotation that indicates the start of a new service implementation.

2254 **Corresponds to:** *implementation.c* element

2255 **Format:**

```
2256 /* @Service(name="serviceName", interfaceHeader="headerFile") */
```

2257 *Snippet A-19: @Service Annotation Format*

2258 where

- 2259 • **name : NCName (0..1)** – specifies the name of the service. The default is the service name for the
2260 interface.
- 2261 • **interfaceHeader : Name (1..1)** – specifies the C header defining the interface.

2262 **Applies to:** Set of functions implementing a service

2263 Function definitions following this annotation form the implementation of this service. This annotation also
2264 serves to bound the scope of the remaining annotations in this section,

2265 Example:

2266 Implementation:

```
2267 /* @Service(name="LoanService", interfaceHeader="loan.h") */
```

2268

2269 ComponentType definition:

```
2270 <componentType name="LoanService">  
2271   <service name="LoanService">  
2272     <interface.c header="loans.h" />  
2273   </service>  
2274 </componentType>
```

2275 *Snippet A-20: Example of @Service Annotation*

2276 **A.3.3 @Reference**

2277 Annotation on a service implementation to indicate it depends on another service providing a specified
2278 interface.

2279 **Corresponds to:** *reference* element of a *componentType* element.

2280 **Format:**

```
2281 /* @Reference(name="referenceName", interfaceHeader="headerFile",  
2282             required="true", multiple="true") */
```

2283 *Snippet A-21: @Reference Annotation Format*

2284 where

- 2285 • **name : NCName (1..1)** – specifies the name of the reference.
- 2286 • **interfaceHeader : Name (1..1)** – specifies the C header defining the interface.
- 2287 • **required : boolean (0..1)** – specifies whether a value has to be set for this reference. Default is **true**.
- 2288 • **multiple : boolean (0..1)** – specifies whether this reference can be wired to multiple services. Default
2289 is **false**.

2290 The multiplicity of the reference is determined from the **required** and **multiple** attributes. If the value of
2291 the **multiple** attribute is true, then component type has a reference with a multiplicity of either 0..n or 1..n
2292 depending on the value of the **required** attribute – 1..n applies if **required=true**. Otherwise a multiplicity
2293 of 0..1 or 1..1 is implied.

2294 **Applies to:** Service

2295 Example:

2296 Implementation:

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

2301

2302 ComponentType definition:

```
2303 <componentType name="LoanService">
```

```

2304     <reference name="getRate">
2305         <interface.c header="rates.h">
2306     </reference>
2307     <reference name="publishRate" multiplicity="0..n">
2308         <interface.c header="myRates.h">
2309     </reference>
2310 </componentType>

```

2311 *Snippet A-22: Example of @Reference Annotation*

2312 **A.3.4 @Property**

2313 Annotation on a service implementation to define a property of the service. Should immediately precede
2314 the variable that the property is based on. The variable declaration is only used for determining the type
2315 of the property. The variable will not be populated with the property value at runtime. Programs use the
2316 `SCAProperty<Type>()` functions for accessing property data.

2317 **Corresponds to:** *property* element of a *componentType* element.

2318 **Format:**

```

2319     /* @Property(name="propertyName", type="typeName",
2320                default="defaultValue", required="true")*/

```

2321 *Snippet A-23: @Property Annotation Format*

2322 where

- 2323 • **name : NCName (0..1)** – specifies the name of the property. If name is not specified the property
2324 name is taken from the name of the variable.
- 2325 • **type : QName (0..1)** – specifies the type of the property. If not specified the type of the property is
2326 based on the C mapping of the type of the following global variable to an xsd type as defined in Data
2327 Binding. If the variable is an array, then the property is many-valued.
- 2328 • **required : boolean (0..1)** – specifies whether a value has to be set in the component definition for
2329 this property. Default is **false**.
- 2330 • **default : <type> (0..1)** – specifies a default value and is only needed if **required** is **false**.

2331 **Applies to:** Variable

2332 An SCA implementation **MUST** ensure that all variables in a component implementation with the same
2333 name and annotated with `@Property` have the same type. [\[CA0007\]](#)

2334 Example:

2335 Implementation:

```

2336     /* @Property */
2337     long loanType;

```

2338

2339 ComponentType definition:

```

2340     <componentType name="LoanService">
2341         <property name="loanType" type="xsd:int" />
2342     </componentType>

```

2343 *Snippet A-24: Example of @Property Annotation*

2344 **A.3.5 @Init**

2345 Annotation on a service implementation to indicate a function to be called when the service is
2346 instantiated. If the service is implemented in a program, this annotation indicates the program is to be
2347 called with an initialization flag prior to the first operation.

2348 **Corresponds to:** `@init="true"` attribute of an *implementation.c* element or a *function* child element of an
2349 *implementation.c* element.

2350 **Format:**

```
2351 /* @Init */
```

2352 *Snippet A-25: @Init Annotation Format*

2353 The default is **false** (the function is not to be called on service initialization).

2354 **Applies to:** Function or Service

2355 Example:

2356 Implementation:

```
2357 /* @Init */  
2358 void init();
```

2359

2360 Component definition:

```
2361 <component name="LoanService">  
2362   <implementation.c module="loan" componentType="LoanService">  
2363     <function name="init" init="true" />  
2364   </implementation.c>  
2365 </component>
```

2366 *Snippet A-26: Example of @Init Annotation*

2367 **A.3.6 @Destroy**

2368 Annotation on a service implementation to indicate a function to be called when the service is terminated.
2369 If the service is implemented in a program, this annotation indicates the program is to be called with a
2370 termination flag after to the final operation.

2371 **Corresponds to:** `@destroy="true"` attribute of an *implementation.c* element or a *function* child element of
2372 an *implementation.c* element.

2373 **Format:**

```
2374 /* @Destroy */
```

2375 *Snippet A-27: @Destroy Annotation Format*

2376 The default is **false** (the function is not to be called on service termination).

2377 **Applies to:** Function or Service

2378 Example:

2379 Implementation:

```
2380 /* @Destroy */  
2381 void cleanup();
```

2382

2383 Component definition:

```
2384 <component name="LoanService">  
2385   <implementation.c module="loan" componentType="LoanService">  
2386     <function name="cleanup" destroy="true" />  
2387   </implementation.c>  
2388 </component>
```

2389 *Snippet A-28: Example of @Destroy Annotation*

2390 **A.3.7 @EagerInit**

2391 Annotation on a service implementation to indicate the service is to be instantiated when its containing
2392 component is started.

2393 **Corresponds to:** `@eagerInit="true"` attribute of an *implementation.c* element.

2394 **Format:**

```
2395 /* @EagerInit */
```

2396 *Snippet A-29: @EagerInit Annotation Format*

2397 The default is **false** (the service is initialized lazily).

2398 **Applies to:** Service

2399 **Example:**

2400 **Implementation:**

```
2401 /* @EagerInit */
```

2402

2403 **Component definition:**

```
2404 <component name="LoanService">
2405     <implementation.c module="loan" componentType="LoanService"
2406         eagerInit="true" />
2407 </component>
```

2408 *Snippet A-30: Example of @EagerInit Annotation*

2409 **A.3.8 @AllowsPassByReference**

2410 Annotation on service implementation or operation to indicate that a service or operation allows pass by
2411 reference semantics.

2412 **Corresponds to:** `@allowsPassByReference="true"` attribute of an *implementation.c* element or a *function*
2413 child element of an *implementation.c* element.

2414 **Format:**

```
2415 /* @AllowsPassByReference */
```

2416 *Snippet A-31: @AllowsPassByReference Annotation Format*

2417 The default is **false** (the service does not allow by reference parameters).

2418 **Applies to:** Service or Function

2419 **Example:**

2420 **Implementation:**

```
2421 /* @Service(name="LoanService" )
2422 * @AllowsPassByReference
2423 */
```

2424

2425 **Component definition:**

```
2426 <component name="LoanService">
2427     <implementation.c module="loan" componentType="LoanService"
2428         allowsPassByReference="true" />
2429 </component>
```

2430 *Snippet A-32: Example of @AllowsPassByReference Annotation*

2431 **A.4 Base Annotation Grammar**

2432 While annotations are defined using the `/* ... */` format for comments, if the `// ...` format is supported by a
2433 C compiler, the `// ...` format MAY be supported by an SCA implementation annotation processor.

2434 **[CA0006]**

2435

```
2436 <annotation> ::= /* @<baseAnnotation> */
```

2437


```
2438 <baseAnnotation> ::= <name> [( <params> )]
2439
2440 <params> ::= <paramNameValue>[ , <paramNameValue> ]* |
2441           <paramValue>[ , <paramValue> ]*
2442
2443 <paramNameValue> ::= <name>=" <value> "
2444
2445 <paramValue> ::= " <value> "
2446
2447 <name> ::= NCName
2448
2449 <value> ::= string
```

2450 *Snippet A-33: Base Annotation Grammar*

- 2451 • Adjacent string constants are concatenated
- 2452 • NCName is as defined by XML schema **[XSD]**
- 2453 • Whitespace including newlines between tokens is ignored.
- 2454 • Annotations with parameters can span multiple lines within a comment, and are considered complete
- 2455 when the terminating “)” is reached.

2456

B C SCA Policy Annotations

2457 SCA provides facilities for the attachment of policy-related metadata to SCA assemblies, which influence
2458 how implementations, services and references behave at runtime. The policy facilities are described in
2459 **[POLICY]**. In particular, the facilities include Intents and Policy Sets, where intents express abstract, high-
2460 level policy requirements and policy sets express low-level detailed concrete policies.

2461 Policy metadata can be added to SCA assemblies through the means of declarative statements placed
2462 into Composite documents and into Component Type documents. These annotations are completely
2463 independent of implementation code, allowing policy to be applied during the assembly and deployment
2464 phases of application development.

2465 However, it can be useful and more natural to attach policy metadata directly to the code of
2466 implementations. This is particularly important where the policies concerned are relied on by the code
2467 itself. An example of this from the Security domain is where the implementation code expects to run
2468 under a specific security Role and where any service operations invoked on the implementation have to
2469 be authorized to ensure that the client has the correct rights to use the operations concerned. By
2470 annotating the code with appropriate policy metadata, the developer can rest assured that this metadata
2471 is not lost or forgotten during the assembly and deployment phases.

2472 The SCA C policy annotations provide the capability for the developer to attach policy information to C
2473 implementation code. The annotations provide both general facilities for attaching SCA Intents and Policy
2474 Sets to C code and annotations for specific policy intents. Policy annotation can be used in files for
2475 service interfaces or component implementations.

2476 B.1 General Intent Annotations

2477 SCA provides the annotation **@Requires** for the attachment of any intent to a C function, to a C function
2478 declaration or to sets of functions implementing a service or sets of function declarations defining a
2479 service interface.

2480 The @Requires annotation can attach one or multiple intents in a single statement. Each intent is
2481 expressed as a string. Intents are XML QNames, which consist of a Namespace URI followed by the
2482 name of the Intent. The precise form used is:

```
2483     "{ " + Namespace URI + " }" + intentname
```

2484 *Snippet B-1: Intent Format*

2485

2486 Intents can be qualified, in which case the string consists of the base intent name, followed by a ".",
2487 followed by the name of the qualifier. There can also be multiple levels of qualification.

2488 This representation is quite verbose, so we expect that reusable constants will be defined for the
2489 namespace part of this string, as well as for each intent that is used by C code. SCA defines constants for
2490 intents such as the following:

2491

```
2492     /* @Define SCA_PREFIX "{http://docs.oasis-pen.org/ns/opencsa/sca/200912}"  
2493     */  
2494     /* @Define CONFIDENTIALITY SCA_PREFIX ## "confidentiality" */  
2495     /* @Define CONFIDENTIALITY_MESSAGE CONFIDENTIALITY ## ".message" */
```

2496 *Snippet B-2: Example Intent Constants*

2497

2498 Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,
2499 separated by an underscore. These intent constants are defined in the file that defines an annotation for
2500 the intent (annotations for intents, and the formal definition of these constants, are covered in a following
2501 section).

2502 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.

2503 **Corresponds to:** @requires attribute of an *interface.c*, *implementation.c*, *function* or *callbackFunction*
2504 element.

2505 **Format:**

```
2506 /* @Requires("qualifiedIntent" | {"qualifiedIntent" [, "qualifiedIntent"]}) */
```

2507 where

```
2508 qualifiedIntent ::= QName | QName.qualifier | QName.qualifier1.qualifier2
```

2509 *Snippet B-3: @Requires Annotation Format*

2510 **Applies to:** Interface, Service, Function, Function Prototype

2511 **Examples:**

2512 Attaching the intents "confidentiality.message" and "integrity.message".

```
2513 /* @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE}) */
```

2514 *Snippet B-4: Example @Requires Annotation*

2515 A reference requiring support for confidentiality:

```
2516 /* @Requires(CONFIDENTIALITY)
2517 * @Reference(interfaceHeader="SetBar.h") */
2518 void setBar(struct barType *bar);
```

2519 *Snippet B-5: @Requires Annotation applied with an @Reference Annotation*

2520

2521 Users can also choose to only use constants for the namespace part of the QName, so that they can add
2522 new intents without having to define new constants. In that case, this definition would instead look like
2523 this:

2524

```
2525 /* @Requires(SCA_PREFIX "confidentiality")
2526 * @Reference(interfaceHeader="SetBar.h") */
2527 void setBar(struct barType *bar);
```

2528 *Snippet B-6: @Requires Annotation Using Mixed Constants and Literals*

2529 B.2 Specific Intent Annotations

2530 In addition to the general intent annotation supplied by the @Requires annotation described above, there
2531 are C annotations that correspond to specific policy intents.

2532 The general form of these specific intent annotations is an annotation with a name derived from the name
2533 of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation
2534 in the form of a string or an array of strings.

2535 For example, the SCA confidentiality intent described in General Intent Annotations using the
2536 @Requires(CONFIDENTIALITY) intent can also be specified with the specific @Confidentiality intent
2537 annotation. The specific intent annotation for the "integrity" security intent is:

2538

```
2539 /* @Integrity */
```

2540 *Snippet B-7: Example Specific Intent Annotation*

2541

2542 **Corresponds to:** @requires="<Intent>" attribute of an *interface.c*, *implementation.c*, *function* or
2543 *callbackFunction* element.

2544 **Format:**

```
2545 /* @<Intent>[(qualifiers)] */
```

2546 where Intent is an NCName that denotes a particular type of intent.

```
2547 Intent ::= NCName
2548 qualifiers ::= "qualifier" | {"qualifier" [, "qualifier"]}
2549 qualifier ::= NCName | NCName/qualifier
```

2550 *Snippet B-8: @<Intent> Annotation Format*

2551 **Applies to:** Interface, Service, Function, Function Prototype – but see specific intents for restrictions

2552 **Example:**

```
2553 /* @ClientAuthentication( {"message", "transport"} ) */
```

2554 *Snippet B-9 Example @<Intent> Annotation*

2555

2556 This annotation attaches the pair of qualified intents: *authentication.message* and *authentication.transport*
2557 (the sca: namespace is assumed in both of these cases – "http:// docs.oasis-
2558 open.org/ns/opencsa/sca/200912").

2559 The Policy Framework **[POLICY]** defines a number of intents and qualifiers. Security Interaction –
2560 Miscellaneous define the annotations for those intents.

2561 B.2.1 Security Interaction

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

2562 *Table B-1: Security Interaction Intent Annotations*

2563

2564 These three intents can be qualified with

- 2565 • transport
- 2566 • message

2567 B.2.2 Security Implementation

Intent	Annotation	Qualifiers
authorization	@Authorization	fine_grain

2568 *Table B-2: Security Implementation Intent Annotations*

2569 B.2.3 Reliable Messaging

Intent	Annotation
atLeastOnce	@AtLeastOnce
atMostOnce	@AtMostOnce
ordered	@Ordered

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

2570 Table B-3: Reliable Messagng Intent Annotations

2571 B.2.4 Transactions

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

2572 Table B-4: Transaction Intent Annotations

2573 B.2.5 Miscellaneous

Intent	Annotation	Qualifiers
SOAP	@SOAP	v1_1 v1_2

2574 Table B-5: Miscellaneous Intent Annotations

2575 B.3 Policy Set Annotations

2576 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies (for example,
2577 a concrete policy is the specific encryption algorithm to use when encrypting messages when using a
2578 specific communication protocol to link a reference to a service).

2579 Policy Sets can be applied directly to C implementations using the **@PolicySets** annotation. The
2580 PolicySets annotation either takes the QName of a single policy set as a string or the name of two or
2581 more policy sets as an array of strings.

2582 **Corresponds to:** @policySets attribute of an *interface.c*, *implementation.c*, *function* or *callbackFunction*
2583 element.

2584 **Format:**

```
2585 /* @PolicySets( "<policy set QName>" |
2586 { "<policy set QName>" [, "<policy set QName>"] }) */
```

2587 *Snippet B-10: @PolicySets Annotation Format*

2588 As for intents, PolicySet names are QNames – in the form of “{Namespace-URI}localPart”.

2589 **Applies to:** Interface, Service, Function, Function Prototype

2590 **Example:**

```
2591 /* @Reference(name="helloService", interfaceHeader="helloService.h",
2592 *           required=true)
2593 * @PolicySets({ MY_NS "WS_Encryption_Policy",
2594 *             MY_NS "WS_Authentication_Policy" }) */
2595 HelloService* helloService;
2596 ...
```

2597 }

2598 *Snippet B-11: Example @PolicySets Annotation*

2599

2600 In this case, the Policy Sets `WS_Encryption_Policy` and `WS_Authentication_Policy` are applied, both
2601 using the namespace defined for the constant `MY_NS`.

2602 PolicySets satisfy intents expressed for the implementation when both are present, according to the rules
2603 defined in **[POLICY]**.

2604 **B.4 Policy Annotation Grammar Additions**

```
2605    <annotation> ::= /* @<baseAnnotation> | @<requiresAnnotation> |  
2606                    @<intentAnnotation> | @<policySetAnnotation> */  
2607  
2608    <requiresAnnotation> ::= Requires(<intents>)  
2609  
2610    <intents> ::= "<qualifiedIntent>" |  
2611                {"<qualifiedIntent>"[, "<qualifiedIntent>"]*})  
2612  
2613    <qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |  
2614                        <intentName>.<qualifier>.<qualifier>  
2615  
2616    <intentName> ::= {anyURI}NCName  
2617  
2618    <intentAnnotation> ::= <intent>[(<qualifiers>)]  
2619  
2620    <intent> ::= NCName[(param)]  
2621  
2622    <qualifiers> ::= "<qualifier>" | {"<qualifier>"[, "<qualifier>"]*}  
2623  
2624    <qualifier> ::= NCName | NCName/<qualifier>  
2625  
2626    <policySetAnnotation> ::= policySets(<policysets>)  
2627  
2628    <policysets> ::= "<policySetName>" | {"<policySetName>"[, "<policySetName>"]*}  
2629  
2630    <policySetName> ::= {anyURI}NCName
```

2631 *Snippet B-12: Annotation Grammar Additions for Policy Annotations*

- 2632 • anyURI is as defined by XML schema **[XSD]**

2633 **B.5 Annotation Constants**

```
2634    <annotationConstant> ::= /* @Define <identifier> <token string> */  
2635  
2636    <identifier> ::= token  
2637  
2638    <token string> ::= "string" | "string"[ ## <token string>]
```

2639 *Snippet B-13: Annotation Constants Grammar*

- 2640 • Constants are immediately expanded

2641

C C WSDL Annotations

2642 To allow developers to control the mapping of C to WSDL, a set of annotations is defined. If WSDL
2643 mapping annotations are supported by an implementation, the annotations defined here MUST be
2644 supported and MUST be mapped to WSDL as described. [CC0005]

2645 C.1 Interface Header Annotations

2646 C.1.1 @WebService

2647 Annotation on a C header file indicating that it represents a web service. A second or subsequent
2648 instance of this annotation in a file, or a first instance after any function declarations indicates the start of
2649 a new service and has to contain a name value. An SCA implementation MUST treat any instance of a
2650 @Remotable annotation and without an explicit @WebService annotation as if a @WebService
2651 annotation with a name value equal to the name value of the @Interface annotation, if specified, and no
2652 other parameters was specified. [CC0001]

2653 **Corresponds to:** javax.jws.WebService annotation in the JAX-WS specification (7.11.1)

2654 **Format:**

```
2655 /* @WebService(name="portTypeName", targetNamespace="namespaceURI",  
2656 *             serviceName="WSDLServiceName", portName="WSDLPortName") */
```

2657 *Snippet C-1: @WebService Annotation Format*

2658 where

- 2659 • **name : NCName (0..1)** – specifies the name of the web service portType. The default is the root
2660 name of the header file containing the annotation. The name of the associated binding is also
2661 determined by the portType. The binding name is the name of the portType suffixed with “Binding”.
- 2662 • **targetNamespace : anyURI (0..1)** – specifies the target namespace for the web service. The default
2663 namespace is determined by the implementation.
- 2664 • **serviceName : NCName (0..1)** – specifies the name for the associated WSDL service. The default
2665 service name is the name of the header file containing the annotation suffixed with “Service”.
- 2666 • **portName : NCName (0..1)** – specifies the name for the associated WSDL port for the service. If
2667 portName is not specified, the name of the WSDL port is the name of the portType suffixed with
2668 “Port”. See [CF0032]

2669 **Applies to:** Header file

2670 **Example:**

2671 Input C header file (stockQuote.h):

```
2672 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
2673 *             serviceName="StockQuoteService") */  
2674 ...
```

2676

2677 **Generated WSDL file:**

```
2678 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
2679             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
2680             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
2681             xmlns:tns="http://www.example.org/"  
2682             targetNamespace="http://www.example.org/">  
2683  
2684     <portType name="StockQuote">  
2685     <sca-c:bindings>
```

```

2686     <sca-c:prefix name="stockQuote" />
2687     </sca-c:bindings>
2688 </portType>
2689
2690 <binding name="StockQuoteServiceSoapBinding">
2691     <soap:binding style="document"
2692         transport="http://schemas.xmlsoap.org/soap/http" />
2693 </binding>
2694
2695 <service name="StockQuoteService">
2696     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2697         <soap:address location="REPLACE_WITH_ACTUAL_URL" />
2698     </port>
2699 </service>
2700 </definitions>

```

2701 *Snippet C-2: Example @WebService Annotation*

2702 C.1.2 @WebFunction

2703 Annotation on a C function indicating that it represents a web service operation. An SCA implementation
2704 MUST treat a function annotated with an @Function annotation and without an explicit @WebFunction
2705 annotation as if a @WebFunction annotation with with an operationName value equal to the name value
2706 of the @Function annotation, an exclude value equal to the exclude value of the @Function annotation
2707 and no other parameters was specified. [CC0002]

2708 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2709 **Format:**

```

2710 /* @WebFunction(operationName="operation", action="SOAPAction",
2711 *               exclude="false") */

```

2712 *Snippet C-3: @WebFunction Annotation Format*

2713 where:

- 2714 • **operationName : NCName (0..1)** – specifies the name of the WSDL operation to associate with this
2715 function. The default is the name of the C function the annotation is applied to omitting any preceding
2716 namespace prefix and portType name.
- 2717 • **action : string (0..1)** – specifies the value associated with the soap:operation/@soapAction attribute
2718 in the resulting code. The default value is an empty string.
- 2719 • **exclude : boolean (0..1)** – specifies whether this function is included in the web service interface.
2720 The default value is "false".

2721 **Applies to:** Function.

2722 **Example:**

2723 Input C header file:

```

2724 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2725 *             serviceName="StockQuoteService") */
2726
2727 /* @WebFunction(operationName="GetLastTradePrice",
2728 *             action="urn:GetLastTradePrice") */
2729 float getLastTradePrice(const char *tickerSymbol);
2730
2731 /* @WebFunction(exclude="true") */
2732 void setLastTradePrice(const char *tickerSymbol, float value);

```

2733

2734 Generated WSDL file:

```

2735 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2736             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2737             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"

```



```

2738     xmlns:tns="http://www.example.org/"
2739     targetNamespace="http://www.example.org/"
2740
2741 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2742           xmlns:tns="http://www.example.org/"
2743           attributeFormDefault="unqualified"
2744           elementFormDefault="unqualified"
2745           targetNamespace="http://www.example.org/">
2746   <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2747   <xs:element name="GetLastTradePriceResponse"
2748             type="tns:GetLastTradePriceResponse"/>
2749   <xs:complexType name="GetLastTradePrice">
2750     <xs:sequence>
2751       <xs:element name="tickerSymbol" type="xs:string"/>
2752     </xs:sequence>
2753   </xs:complexType>
2754   <xs:complexType name="GetLastTradePriceResponse">
2755     <xs:sequence>
2756       <xs:element name="return" type="xs:float"/>
2757     </xs:sequence>
2758   </xs:complexType>
2759 </xs:schema>
2760
2761 <message name="GetLastTradePrice">
2762   <part name="parameters" element="tns:GetLastTradePrice">
2763     </part>
2764 </message>
2765
2766 <message name="GetLastTradePriceResponse">
2767   <part name="parameters" element="tns:GetLastTradePriceResponse">
2768     </part>
2769 </message>
2770
2771 <portType name="StockQuote">
2772   <sca-c:bindings>
2773     <sca-c:prefix name="stockQuote"/>
2774   </sca-c:bindings>
2775   <operation name="GetLastTradePrice">
2776     <sca-c:bindings>
2777       <sca-c:function name="getLastTradePrice"/>
2778     </sca-c:bindings>
2779     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2780       </input>
2781     <output name="GetLastTradePriceResponse"
2782            message="tns:GetLastTradePriceResponse">
2783       </output>
2784   </operation>
2785 </portType>
2786
2787 <binding name="StockQuoteServiceSoapBinding">
2788   <soap:binding style="document"
2789                transport="http://schemas.xmlsoap.org/soap/http"/>
2790   <operation name="GetLastTradePrice">
2791     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2792     <input name="GetLastTradePrice">
2793       <soap:body use="literal"/>
2794     </input>
2795     <output name="GetLastTradePriceResponse">
2796       <soap:body use="literal"/>
2797     </output>
2798   </operation>
2799 </binding>
2800
2801 <service name="StockQuoteService">

```

```

2802     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2803         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2804     </port>
2805 </service>
2806 </definitions>

```

2807 *Snippet C-4: Example @WebFunction Annotation*

2808 C.1.3 @WebOperation

2809 Annotation on a C request message struct indicating that it represents a web service operation. An SCA
2810 implementation MUST treat a struct annotated with an @Operation annotation without an explicit
2811 @WebOperation annotation as if a @WebOperation annotation with with an operationName value equal
2812 to the name value of the @Operation annotation, a response value equal to the response value of the
2813 @Operation annotation, an exclude value equal to the exclude value of the @Operation annotation and
2814 no other parameters was specified. [CC0003]

2815 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2816 **Format:**

```

2817 /* @WebOperation(operationName="operation", response="responseStruct",
2818 *                action="SOAPAction", exclude="false") */

```

2819 *Snippet C-5: @WebOperation Annotation Format*

2820 where:

- 2821 • **operationName : NCName (0..1)** – specifies the name of the WSDL operation to associate with this
2822 request message struct. The default is the name of the C struct the annotation is applied to omitting
2823 any preceding namespace prefix and portType name.
- 2824 • **response : NMTOKEN (0..1)** – specifies the name of the struct that defines the format of the
2825 response message.
- 2826 • **action string : (0..1)** – specifies the value associated with the soap:operation/@soapAction attribute
2827 in the resulting code. The default value is an empty string.
- 2828 • **exclude binary : (0..1)** – specifies whether this struct is included in the web service interface. The
2829 default value is “false”.

2830 **Applies to:** Struct.

2831 Example:

2832 Input C header file:

```

2833 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2834 *             serviceName="StockQuoteService") */
2835
2836 /* @WebOperation(operationName="GetLastTradePrice",
2837 *               response="getLastTradePriseResponseMsg"
2838 *               action="urn:GetLastTradePrice") */
2839 struct getLastTradePriceMsg {
2840     char tickerSymbol[10];
2841 } getLastTradePrice;
2842
2843 struct getLastTradePriceResponseMsg {
2844     float return;
2845 } getLastTradePriceResponse;

```

2846

2847 Generated WSDL file:

```

2848 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2849             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2850             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
2851             xmlns:tns="http://www.example.org/"
2852             targetNamespace="http://www.example.org/">

```

```

2853
2854 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2855         xmlns:tns="http://www.example.org/"
2856         attributeFormDefault="unqualified"
2857         elementFormDefault="unqualified"
2858         targetNamespace="http://www.example.org/">
2859   <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
2860   <xs:element name="GetLastTradePriceResponse"
2861             type="tns:GetLastTradePriceResponse"/>
2862   <xs:simpleType name="TickerSymbolType">
2863     <xs:restriction base="xs:string">
2864       <xsd:maxLength value="9"/>
2865     </xs:restriction>
2866   </xs:simpleType>
2867   <xs:complexType name="GetLastTradePrice">
2868     <xs:sequence>
2869       <xs:element name="tickerSymbol" type="TickerSymbolType"/>
2870     </xs:sequence>
2871   </xs:complexType>
2872   <xs:complexType name="GetLastTradePriceResponse">
2873     <xs:sequence>
2874       <xs:element name="return" type="xs:float"/>
2875     </xs:sequence>
2876   </xs:complexType>
2877 </xs:schema>
2878
2879 <message name="GetLastTradePrice">
2880   <sca-c:bindings>
2881     <sca-c:struct name="getLastTradePrice"/>
2882   </sca-c:bindings>
2883   <part name="parameters" element="tns:GetLastTradePrice">
2884     </part>
2885 </message>
2886
2887 <message name="GetLastTradePriceResponse">
2888   <sca-c:bindings>
2889     <sca-c:struct name="getLastTradePriceResponse"/>
2890   </sca-c:bindings>
2891   <part name="parameters" element="tns:GetLastTradePriceResponse">
2892     </part>
2893 </message>
2894
2895 <portType name="StockQuote">
2896   <sca-c:bindings>
2897     <sca-c:prefix name="stockQuote"/>
2898   </sca-c:bindings>
2899   <operation name="GetLastTradePrice">
2900     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
2901       </input>
2902     <output name="GetLastTradePriceResponse"
2903            message="tns:GetLastTradePriceResponse">
2904       </output>
2905   </operation>
2906 </portType>
2907
2908 <binding name="StockQuoteServiceSoapBinding">
2909   <soap:binding style="document"
2910               transport="http://schemas.xmlsoap.org/soap/http"/>
2911   <operation name="GetLastTradePrice">
2912     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
2913     <input name="GetLastTradePrice">
2914       <soap:body use="literal"/>
2915     </input>
2916     <output name="GetLastTradePriceResponse">

```

```

2917         <soap:body use="literal"/>
2918     </output>
2919 </operation>
2920 </binding>
2921
2922 <service name="StockQuoteService">
2923     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2924         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2925     </port>
2926 </service>
2927 </definitions>

```

2928 *Snippet C-6: Example @WebOperation Annotation*

2929 C.1.4 @OneWay

2930 Annotation on a C function indicating that it represents a one-way request. The @OneWay annotation
 2931 also affects the service interface. See @OneWay.

2932 **Corresponds to:** javax.jws.OneWay annotation in the JAX-WS specification (7.11.3)

2933 **Format:**

```

2934 /* @OneWay */

```

2935 *Snippet C-7: @OneWay Annotation Format*

2936 **Applies to:** Function.

2937 **Example:**

2938 **Input C header file:**

```

2939 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2940 *             serviceName="StockQuoteService") */
2941
2942 /* @WebFunction(operationName="SetTradePrice",
2943 *             action="urn:SetTradePrice")
2944 * @OneWay */
2945 void setTradePrice(const char *tickerSymbol, float price);

```

2946

2947 **Generated WSDL file:**

```

2948 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2949             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2950             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
2951             xmlns:tns="http://www.example.org/"
2952             targetNamespace="http://www.example.org/">
2953
2954     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2955               xmlns:tns="http://www.example.org/"
2956               attributeFormDefault="unqualified"
2957               elementFormDefault="unqualified"
2958               targetNamespace="http://www.example.org/">
2959         <xs:element name="SetTradePrice" type="tns:SetTradePrice"/>
2960         <xs:complexType name="SetTradePrice">
2961             <xs:sequence>
2962                 <xs:element name="tickerSymbol" type="xs:string"/>
2963                 <xs:element name="price" type="xs:float"/>
2964             </xs:sequence>
2965         </xs:complexType>
2966     </xs:schema>
2967
2968     <message name="SetTradePrice">
2969         <part name="parameters" element="tns:SetTradePrice">
2970             </part>
2971     </message>

```

```

2972
2973 <portType name="StockQuote">
2974   <sca-c:bindings>
2975     <sca-c:prefix name="stockQuote"/>
2976   </sca-c:bindings>
2977   <operation name="SettTradePrice">
2978     <sca-c:bindings>
2979       <sca-c:function name="setTradePrice"/>
2980     </sca-c:bindings>
2981     <input name="SetTradePrice" message="tns:SetTradePrice">
2982       </input>
2983   </operation>
2984 </portType>
2985
2986 <binding name="StockQuoteServiceSoapBinding">
2987   <soap:binding style="document"
2988     transport="http://schemas.xmlsoap.org/soap/http"/>
2989   <operation name="SetTradePrice">
2990     <soap:operation soapAction="urn:SetTradePrice" style="document"/>
2991     <input name="SetTradePrice">
2992       <soap:body use="literal"/>
2993     </input>
2994   </operation>
2995 </binding>
2996
2997 <service name="StockQuoteService">
2998   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2999     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3000   </port>
3001 </service>
3002 </definitions>

```

3003 *Snippet C-8: Example @OneWay Annotation*

3004 C.1.5 @WebParam

3005 Annotation on a C function indicating the mapping of a parameter to the associated input and output
3006 WSDL messages. Or on a C struct indicating the mapping of a member to the associated WSDL
3007 message.

3008 **Corresponds to:** javax.jws.WebParam annotation in the JAX-WS specification (7.11.4)

3009 **Format:**

```

3010 /* @WebParam(paramName="parameter", name="WSDLElement",
3011 *           targetNamespace="namespaceURI", mode="IN"|"OUT"|"INOUT",
3012 *           header="false", partName="WSDLPart", type="xsdType") */

```

3013 *Snippet C-9: @WebParam Annotation Format*

3014 where:

- 3015 • **paramName : NCName (1..1)** – specifies the name of the parameter that this annotation applies to.
3016 The value of the paramName of a @WebParam annotation MUST be the name of a parameter of the
3017 function the annotation is applied to. [CC0009]
- 3018 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default
3019 value is the name of the parameter. If an @WebParam annotation is not present, and the parameter
3020 is unnamed, then a name of "argN", where N is an incrementing value from 1 indicating the position of
3021 he parameter in the argument list, will be used.
- 3022 • **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default
3023 namespace is is the namespace of the associated @WebService. The targetNamespace attribute is
3024 ignored unless the binding style is document, and the binding parameterStyle is bare. See
3025 @SOAPBinding.

- 3026 • **mode : token (0..1)** – specifies whether the parameter is associated with the input message, output
3027 message, or both. The default value is determined by the passing mechanism for the parameter. See
3028 Method Parameters and Return Type.
- 3029 • **header : boolean (0..1)** – specifies whether this parameter is associated with a SOAP header
3030 element. The default value is "false".
- 3031 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The
3032 default value is the value of name.
- 3033 • **type : QName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with
3034 this parameter. The value of the type property of a @WebParam annotation MUST be either one of
3035 the simpleTypes defined in namespace
3036 http://www.w3.org/2001/XMLSchema or, if the type of the
3037 parameter is a struct, the QName of a XSD complex type following the mapping specified in Complex
3038 Content Binding. [CC0006] The default type is determined by the mapping defined in Data Binding.

3039 **Applies to:** Function parameter or struct member.

3040 **Example:**

3041 **Input C header file:**

```
3042 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3043 *             serviceName="StockQuoteService") */
3044
3045 /* @WebFunction(operationName="GetLastTradePrice",
3046 *             action="urn:GetLastTradePrice")
3047 * @WebParam(paramName="tickerSymbol", name="symbol", mode="IN") */
3048 float getLastTradePrice(char *tickerSymbol);
```

3049

3050 **Generated WSDL file:**

```
3051 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3052             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3053             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3054             xmlns:tns="http://www.example.org/"
3055             targetNamespace="http://www.example.org/">
3056
3057     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3058               xmlns:tns="http://www.example.org/"
3059               attributeFormDefault="unqualified"
3060               elementFormDefault="unqualified"
3061               targetNamespace="http://www.example.org/">
3062         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3063         <xs:element name="GetLastTradePriceResponse"
3064                   type="tns:GetLastTradePriceResponse"/>
3065         <xs:complexType name="GetLastTradePrice">
3066             <xs:sequence>
3067                 <xs:element name="symbol" type="xs:string"/>
3068             </xs:sequence>
3069         </xs:complexType>
3070         <xs:complexType name="GetLastTradePriceResponse">
3071             <xs:sequence>
3072                 <xs:element name="return" type="xs:float"/>
3073             </xs:sequence>
3074         </xs:complexType>
3075     </xs:schema>
3076
3077     <message name="GetLastTradePrice">
3078         <part name="parameters" element="tns:GetLastTradePrice">
3079         </part>
3080     </message>
3081
3082     <message name="GetLastTradePriceResponse">
3083         <part name="parameters" element="tns:GetLastTradePriceResponse">
```

```

3084     </part>
3085 </message>
3086
3087 <portType name="StockQuote">
3088   <sca-c:bindings>
3089     <sca-c:prefix name="stockQuote" />
3090   </sca-c:bindings>
3091   <operation name="GetLastTradePrice">
3092     <sca-c:bindings>
3093       <sca-c:function name="getLastTradePrice" />
3094       <sca-c:parameter name="tickerSymbol"
3095         part="tns:GetLastTradePrice/parameter"
3096         childElementName="symbol" />
3097     </sca-c:bindings>
3098     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3099     </input>
3100     <output name="GetLastTradePriceResponse"
3101       message="tns:GetLastTradePriceResponse">
3102     </output>
3103   </operation>
3104 </portType>
3105
3106 <binding name="StockQuoteServiceSoapBinding">
3107   <soap:binding style="document"
3108     transport="http://schemas.xmlsoap.org/soap/http" />
3109   <operation name="GetLastTradePrice">
3110     <soap:operation soapAction="urn:GetLastTradePrice" style="document" />
3111     <input name="GetLastTradePrice">
3112       <soap:body use="literal" />
3113     </input>
3114     <output name="GetLastTradePriceResponse">
3115       <soap:body use="literal" />
3116     </output>
3117   </operation>
3118 </binding>
3119
3120 <service name="StockQuoteService">
3121   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3122     <soap:address location="REPLACE_WITH_ACTUAL_URL" />
3123   </port>
3124 </service>
3125 </definitions>

```

3126 *Snippet C-10: Example @WebParam Annotation*

3127 **C.1.6 @WebResult**

3128 Annotation on a C function indicating the mapping of the function's return type to the associated output
3129 WSDL message.

3130 **Corresponds to:** javax.jws.WebResult annotation in the JAX-WS specification (7.11.5)

3131 **Format:**

```

3132 /* @WebResult(name="WSDLElement", targetNamespace="namespaceURI",
3133 *           header="false", partName="WSDLPart", type="xsdType") */

```

3134 *Snippet C-11: @WebResult Annotation Format*

3135 where:

- 3136 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default
3137 value is "return".
- 3138 • **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default
3139 namespace is the namespace of the associated @WebService. The targetNamespace attribute is

- 3140 ignored unless the binding style is document, and the binding parameterStyle is bare. (See
3141 @SOAPBinding).
- 3142 • **header : boolean (0..1)** – specifies whether the result is associated with a SOAP header element.
3143 The default value is "false".
 - 3144 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The
3145 default value is the value of name.
 - 3146 • **type : NCName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with
3147 this parameter. The value of the type property of a @WebResult annotation MUST be one of the
3148 simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema. [CC0007] The default type
3149 is determined by the mapping defined in 11.3.1.

3150 **Applies to:** Function.

3151 **Example:**

3152 Input C header file:

```
3153 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3154 *             serviceName="StockQuoteService") */
3155
3156 /* @WebFunction(operationName="GetLastTradePrice",
3157 *             action="urn:GetLastTradePrice")
3158 * @WebResult(name="price") */
3159 float getLastTradePrice(const char *tickerSymbol);
```

3160

3161 **Generated WSDL file:**

```
3162 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3163             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3164             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3165             xmlns:tns="http://www.example.org/"
3166             targetNamespace="http://www.example.org/">
3167
3168     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3169               xmlns:tns="http://www.example.org/"
3170               attributeFormDefault="unqualified"
3171               elementFormDefault="unqualified"
3172               targetNamespace="http://www.example.org/">
3173       <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3174       <xs:element name="GetLastTradePriceResponse"
3175                 type="tns:GetLastTradePriceResponse"/>
3176       <xs:complexType name="GetLastTradePrice">
3177         <xs:sequence>
3178           <xs:element name="tickerSymbol" type="xs:string"/>
3179         </xs:sequence>
3180       </xs:complexType>
3181       <xs:complexType name="GetLastTradePriceResponse">
3182         <xs:sequence>
3183           <xs:element name="price" type="xs:float"/>
3184         </xs:sequence>
3185       </xs:complexType>
3186     </xs:schema>
3187
3188     <message name="GetLastTradePrice">
3189       <part name="parameters" element="tns:GetLastTradePrice">
3190       </part>
3191     </message>
3192
3193     <message name="GetLastTradePriceResponse">
3194       <part name="parameters" element="tns:GetLastTradePriceResponse">
3195       </part>
3196     </message>
3197
```



```

3198     <portType name="StockQuote">
3199         <sca-c:bindings>
3200             <sca-c:prefix name="stockQuote"/>
3201         </sca-c:bindings>
3202         <operation name="GetLastTradePrice">
3203             <sca-c:bindings>
3204                 <sca-c:function name="getLastTradePrice"/>
3205             </sca-c:bindings>
3206             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3207                 </input>
3208             <output name="GetLastTradePriceResponse"
3209                 message="tns:GetLastTradePriceResponse">
3210                 </output>
3211             </operation>
3212         </portType>
3213
3214         <binding name="StockQuoteServiceSoapBinding">
3215             <soap:binding style="document"
3216                 transport="http://schemas.xmlsoap.org/soap/http"/>
3217             <operation name="GetLastTradePrice">
3218                 <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3219                 <input name="GetLastTradePrice">
3220                     <soap:body use="literal"/>
3221                 </input>
3222                 <output name="GetLastTradePriceResponse">
3223                     <soap:body use="literal"/>
3224                 </output>
3225             </operation>
3226         </binding>
3227
3228         <service name="StockQuoteService">
3229             <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3230                 <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3231             </port>
3232         </service>
3233     </definitions>

```

3234 *Snippet C-12: Example @WebResult Annotation*

3235 **C.1.7 @SOAPBinding**

3236 Annotation on a C WebService or function specifying the mapping of the web service onto the SOAP
3237 message protocol.

3238 **Corresponds to:** javax.jws.SOAPBinding annotation in the JAX-WS specification (7.11.6)

3239 **Format:**

```

3240 /* @SOAPBinding(style="DOCUMENT" | "RPC", use="LITERAL" | "ENCODED",
3241 *               parameterStyle="BARE" | "WRAPPED") */

```

3242 *Snippet C-13: @SOAPBinding Annotation Format*

3243 where:

- 3244 • **style : token (0..1)** – specifies the WSDL binding style. The default value is “DOCUMENT”.
- 3245 • **use : token (0..1)** – specifies the WSDL binding use. The default value is “LITERAL”.
- 3246 • **parameterStyle : token (0..1)** – specifies the WSDL parameter style. The default value is
3247 “WRAPPED”.

3248 **Applies to:** WebService, Function.

3249 **Example:**

3250 Input C header file:

```

3251 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",

```

```

3252 *             serviceName="StockQuoteService") */
3253 * @SOAPBinding(style="RPC") */
3254
3255 ...

```

3256

3257 **Generated WSDL file:**

```

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

```

3281 *Snippet C-14: Example @SOAPBinding Annotation*

3282 **C.1.8 @WebFault**

3283 Annotation on a C struct indicating that it format of a fault message.

3284 **Corresponds to:** javax.xml.ws.WebFault annotation in the JAX-WS specification (7.2)

3285 **Format:**

```

3286 /* @WebFault(name="WSDL_Element", targetNamespace="namespaceURI") */

```

3287 *Snippet C-15: @WebFault Annotation Format*

3288 where:

- 3289 • **name : NCName (1..1)** – specifies the local name of the global element mapped to this fault.
- 3290 • **targetNamespace : string (0..1)** – specifies the namespace of the global element mapped to this
- 3291 fault. The default namespace is determined by the implementation.

3292 **Applies to:** struct.

3293 **Example:**

3294 **Input C header file:**

```

3295 /* @WebFault(name="UnknownSymbolFault",
3296 *             targetNamespace="http://www.example.org/")
3297 struct UnkSymMsg {
3298     char faultInfo[10];
3299 } unkSymInfo;
3300
3301 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3302 *              serviceName="StockQuoteService") */
3303
3304 /* @WebFunction(operationName="GetLastTradePrice",

```

```
3305 *           action="urn:GetLastTradePrice")
3306 * @WebThrows(faults="unkSymMsg") */
3307 float getLastTradePrice(const char *tickerSymbol);
```

3308

3309 **Generated WSDL file:**

```
3310 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3311             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3312             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3313             xmlns:tns="http://www.example.org/"
3314             targetNamespace="http://www.example.org/">
3315
3316   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3317             xmlns:tns="http://www.example.org/"
3318             attributeFormDefault="unqualified"
3319             elementFormDefault="unqualified"
3320             targetNamespace="http://www.example.org/">
3321     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3322     <xs:element name="GetLastTradePriceResponse"
3323               type="tns:GetLastTradePriceResponse"/>
3324     <xs:complexType name="GetLastTradePrice">
3325       <xs:sequence>
3326         <xs:element name="tickerSymbol" type="xs:string"/>
3327       </xs:sequence>
3328     </xs:complexType>
3329     <xs:complexType name="GetLastTradePriceResponse">
3330       <xs:sequence>
3331         <xs:element name="return" type="xs:float"/>
3332       </xs:sequence>
3333     </xs:complexType>
3334     <xs:simpleType name="UnknownSymbolFaultType">
3335       <xs:restriction base="xs:string">
3336         <xsd:maxLength value="9"/>
3337       </xs:restriction>
3338     </xs:simpleType>
3339     <xs:element name="UnknownSymbolFault" type="UnknownSymbolFaultType"/>
3340   </xs:schema>
3341
3342   <message name="GetLastTradePrice">
3343     <part name="parameters" element="tns:GetLastTradePrice">
3344       </part>
3345   </message>
3346
3347   <message name="GetLastTradePriceResponse">
3348     <part name="parameters" element="tns:GetLastTradePriceResponse">
3349       </part>
3350   </message>
3351
3352   <message name="UnknownSymbol">
3353     <sca-c:bindings>
3354       <sca-c:struct name="unkSymMsg"/>
3355     </sca-c:bindings>
3356     <part name="parameters" element="tns:UnknownSymbolFault">
3357       </part>
3358   </message>
3359
3360   <portType name="StockQuote">
3361     <sca-c:bindings>
3362       <sca-c:prefix name="stockQuote"/>
3363     </sca-c:bindings>
3364     <operation name="GetLastTradePrice">
3365       <sca-c:bindings>
3366         <sca-c:function name="getLastTradePrice"/>
3367       </sca-c:bindings>
```

```

3368     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3369     </input>
3370     <output name="GetLastTradePriceResponse"
3371           message="tns:GetLastTradePriceResponse">
3372     </output>
3373     <fault name="UnknownSymbol" message="tns:UnknownSymbol">
3374     </fault>
3375   </operation>
3376 </portType>
3377
3378 <binding name="StockQuoteServiceSoapBinding">
3379   <soap:binding style="document"
3380     transport="http://schemas.xmlsoap.org/soap/http"/>
3381   <operation name="GetLastTradePrice">
3382     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3383     <input name="GetLastTradePrice">
3384       <soap:body use="literal"/>
3385     </input>
3386     <output name="GetLastTradePriceResponse">
3387       <soap:body use="literal"/>
3388     </output>
3389     <fault>
3390       <soap:fault name="UnknownSymbol" use="literal"/>
3391     </fault>
3392   </operation>
3393 </binding>
3394
3395 <service name="StockQuoteService">
3396   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3397     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3398   </port>
3399 </service>
3400 </definitions>

```

3401 *Snippet C-16: Example @WebFault Annotation*

3402 **C.1.9 @WebThrows**

3403 Annotation on a C function or operation indicating which faults might be thrown by this function or
3404 operation.

3405 **Corresponds to:** No equivalent in JAX-WS.

3406 **Format:**

```
3407 /* @WebThrows(faults="faultMsg1" [, "faultMsgn"]*) */
```

3408 *Snippet C-17: @WebThrows Annotation Format*

3409 where:

- 3410 • **faults : NMTOKEN (1..n)** – specifies the names of all faults that might be thrown by this function or
3411 operation. The name of the fault is the name of its associated C struct name. A C struct that is listed
3412 in a @WebThrows annotation MUST itself have a @WebFault annotation. [CC0004]

3413 **Applies to:** Function or Operation

3414 **Example:**

3415 See @WebFault.

3416 D C WSDL Mapping Extensions

3417 The following WSDL extensions are used to augment the conversion process from WSDL to C. All of
3418 these extensions are defined in the namespace `http://docs.oasis-open.org/ns/opencsa/sca-c-
3419 cpp/c/200901`. For brevity, all definitions of these extensions will be fully qualified, and all references to
3420 the "sca-c" prefix are associated with the namespace above. If WSDL extensions are supported by an
3421 implementation, all the extensions defined here MUST be supported and MUST be mapped to C as
3422 described. [CD0001]

3423 D.1 <sca-c:bindings>

3424 <sca-c:bindings> is a container type which can be used as a WSDL extension. All other SCA wsdl
3425 extensions will be specified as children of a <sca-c:bindings> element. An <sca-c:bindings> element can
3426 be used as an extension to any WSDL type that accepts extensions.

3427 D.2 <sca-c:prefix>

3428 <sca-c:prefix> provides a mechanism for defining an alternate prefix for the functions or structs
3429 implementing the operations of a portType.

3430 Format:

```
3431 <sca-c:prefix name="portTypePrefix"/>
```

3432 *Snippet D-1: <sca-c:prefix> Element Format*

3433 where:

- 3434 • **prefix/@name : string (1..1)** – specifies the string to prepend to an operation name when generating
3435 a C function or structure name.

3436 Applicable WSDL element(s):

- 3437 • wsdl:portType

3438 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:prefix/> child element. [CD0003]

3439 Example:

3440 Input WSDL file:

```
3441 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3442   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3443   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
3444   xmlns:tns="http://www.example.org/"  
3445   targetNamespace="http://www.example.org/">  
3446  
3447   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
3448     xmlns:tns="http://www.example.org/"  
3449     attributeFormDefault="unqualified"  
3450     elementFormDefault="unqualified"  
3451     targetNamespace="http://www.example.org/">  
3452     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
3453     <xs:element name="GetLastTradePriceResponse"  
3454       type="tns:GetLastTradePriceResponse"/>  
3455     <xs:complexType name="GetLastTradePrice">  
3456       <xs:sequence>  
3457         <xs:element name="tickerSymbol" type="xs:string"/>  
3458       </xs:sequence>  
3459     </xs:complexType>  
3460     <xs:complexType name="GetLastTradePriceResponse">  
3461       <xs:sequence>  
3462         <xs:element name="return" type="xs:float"/>
```

```

3463     </xs:sequence>
3464   </xs:complexType>
3465 </xs:schema>
3466
3467   <message name="GetLastTradePrice">
3468     <part name="parameters" element="tns:GetLastTradePrice">
3469       </part>
3470   </message>
3471
3472   <message name="GetLastTradePriceResponse">
3473     <part name="parameters" element="tns:GetLastTradePriceResponse">
3474       </part>
3475   </message>
3476
3477   <portType name="StockQuote">
3478     <sca-c:bindings>
3479       <sca-c:prefix name="stockQuote"/>
3480     </sca-c:bindings>
3481     <operation name="GetLastTradePrice">
3482       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3483         </input>
3484       <output name="GetLastTradePriceResponse"
3485         message="tns:GetLastTradePriceResponse">
3486         </output>
3487     </operation>
3488   </portType>
3489
3490   <binding name="StockQuoteServiceSoapBinding">
3491     <soap:binding style="document"
3492       transport="http://schemas.xmlsoap.org/soap/http"/>
3493     <operation name="GetLastTradePrice">
3494       <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3495       <input name="GetLastTradePrice">
3496         <soap:body use="literal"/>
3497       </input>
3498       <output name="GetLastTradePriceResponse">
3499         <soap:body use="literal"/>
3500       </output>
3501     </operation>
3502   </binding>
3503
3504   <service name="StockQuoteService">
3505     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3506       <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3507     </port>
3508   </service>
3509 </definitions>

```

3510

Generated C header file:

```

3512 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3513 *           serviceName="StockQuoteService") */
3514
3515 /* @WebFunction(operationName="GetLastTradePrice",
3516 *           action="urn:GetLastTradePrice") */
3517 float stockQuoteGetLastTradePrice(const char *tickerSymbol);

```

3518 *Snippet D-2: Example <sca-c:prefix> Element*

3519 **D.3 <sca-c:enableWrapperStyle>**

3520 <sca-c:enableWrapperStyle> indicates whether or not the wrapper style for messages is applied, when
3521 otherwise applicable. If false, the wrapper style will never be applied.

3522 **Format:**

```
3523 <sca-c:enableWrapperStyle>value</sca-c:enableWrapperStyle>
```

3524 *Snippet D-3: <sca-c:enableWrapperStyle> Element Format*

3525 where:

- 3526 • **enableWrapperStyle/text() : boolean (1..1)** – specifies whether wrapper style is enabled or disabled
3527 for this element and any of its children. The default value is “true”.

3528 **Applicable WSDL element(s):**

- 3529 • wsdl:definitions
- 3530 • wsdl:portType – overrides a binding applied to wsdl:definitions
- 3531 • wsdl:portType/wsdl:operation – overrides a binding applied to wsdl:definitions or the enclosing
3532 wsdl:portType

3533 A <sca-c:bindings/> element **MUST NOT** have more than one <sca-c:enableWrapperStyle/> child
3534 element. [CD0004]

3535 Example:

3536 Input WSDL file:

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



```

3579     <sca-c:bindings>
3580         <sca-c:function name="getLastTradePrice" />
3581     </sca-c:bindings>
3582     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3583     </input>
3584     <output name="GetLastTradePriceResponse"
3585         message="tns:GetLastTradePriceResponse">
3586     </output>
3587     </operation>
3588 </portType>
3589 </definitions>

```

3590

3591 Generated C header file:

```

3592 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3593 *      serviceName="StockQuoteService") */
3594
3595 /* @WebFunction(operationName="GetLastTradePrice",
3596 *      action="urn:GetLastTradePrice") */
3597 DATAOBJECT getLastTradePrice(DATAOBJECT parameters);

```

3598 *Snippet D-4: Example <sca-c:enableWrapperStyle> Element*

3599 **D.4 <sca-c:function>**

3600 <sca-c:function> specifies the name of the C function that the associated WSDL operation is associated
3601 with. If <sca-c:function> is used, the portType prefix, either default or a specified with <sca-c:prefix> is not
3602 prepended to the function name.

3603 **Format:**

```

3604 <sca-c:function name="myFunction" />

```

3605 *Snippet D-5: <sca-c:function> Element Format*

3606 where:

- 3607 • **function/@name : NCName (1..1)** – specifies the name of the C function associated with this WSDL
3608 operation.

3609 **Applicable WSDL element(s):**

- 3610 • wsdl:portType/wsdl:operation

3611 A <sca-c:bindings/> element **MUST NOT** have more than one <sca-c:function/> child element. [CD0005]

3612 Example:

3613 Input WSDL file:

```

3614 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3615     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3616     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3617     xmlns:tns="http://www.example.org/"
3618     targetNamespace="http://www.example.org/">
3619
3620     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3621         xmlns:tns="http://www.example.org/"
3622         attributeFormDefault="unqualified"
3623         elementFormDefault="unqualified"
3624         targetNamespace="http://www.example.org/">
3625         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice" />
3626         <xs:element name="GetLastTradePriceResponse"
3627             type="tns:GetLastTradePriceResponse" />
3628         <xs:complexType name="GetLastTradePrice">
3629             <xs:sequence>
3630                 <xs:element name="tickerSymbol" type="xs:string" />
3631             </xs:sequence>

```



```

3632     </xs:complexType>
3633     <xs:complexType name="GetLastTradePriceResponse">
3634         <xs:sequence>
3635             <xs:element name="return" type="xs:float"/>
3636         </xs:sequence>
3637     </xs:complexType>
3638 </xs:schema>
3639
3640 <message name="GetLastTradePrice">
3641     <part name="parameters" element="tns:GetLastTradePrice">
3642     </part>
3643 </message>
3644
3645 <message name="GetLastTradePriceResponse">
3646     <part name="parameters" element="tns:GetLastTradePriceResponse">
3647     </part>
3648 </message>
3649
3650 <portType name="StockQuote">
3651     <sca-c:bindings>
3652         <sca-c:prefix name="stockQuote"/>
3653     </sca-c:bindings>
3654     <operation name="GetLastTradePrice">
3655         <sca-c:bindings>
3656             <sca-c:function name="getTradePrice"/>
3657         </sca-c:bindings>
3658         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3659         </input>
3660         <output name="GetLastTradePriceResponse"
3661             message="tns:GetLastTradePriceResponse">
3662         </output>
3663     </operation>
3664 </portType>
3665 </definitions>

```

3666
3667 **Generated C header file:**

```

3668 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3669 *      serviceName="StockQuoteService") */
3670
3671 /* @WebFunction(operationName="GetLastTradePrice",
3672 *      action="urn:GetLastTradePrice") */
3673 float getTradePrice(const wchar_t *tickerSymbol);

```

3674 *Snippet D-6: Example <sca-c:function> Element*

3675 **D.5 <sca-c:struct>**

3676 <sca-c:struct> specifies the name of the C struct that the associated WSDL message is associated with. If
3677 <sca-c:struct> is used for an operation request or response message, the portType prefix, either default
3678 or a specified with <sca-c:prefix> is not prepended to the struct name.

3679 **Format:**

```

3680 <sca-c:struct name="myStruct"/>

```

3681 *Snippet D-7: <sca-c:struct> Element Format*

3682 where:

- 3683 • **struct/@name : NCName (1..1)** – specifies the name of the C struct associated with this WSDL
3684 message.

3685 **Applicable WSDL element(s):**

- 3686 • wsdl:message

3687 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:struct/> child element. [CD0006]

3688 Example:

3689 Input WSDL file:

```
3690 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3691             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3692             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3693             xmlns:tns="http://www.example.org/"
3694             targetNamespace="http://www.example.org/">
3695
3696     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3697               xmlns:tns="http://www.example.org/"
3698               attributeFormDefault="unqualified"
3699               elementFormDefault="unqualified"
3700               targetNamespace="http://www.example.org/">
3701       <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3702       <xs:element name="GetLastTradePriceResponse"
3703                 type="tns:GetLastTradePriceResponse"/>
3704       <xs:complexType name="GetLastTradePrice">
3705         <xs:sequence>
3706           <xs:element name="tickerSymbol" type="xs:string"/>
3707         </xs:sequence>
3708       </xs:complexType>
3709       <xs:complexType name="GetLastTradePriceResponse">
3710         <xs:sequence>
3711           <xs:element name="return" type="xs:float"/>
3712         </xs:sequence>
3713       </xs:complexType>
3714     </xs:schema>
3715
3716     <message name="GetLastTradePrice">
3717       <sca-c:bindings>
3718         <sca-c:struct name="getTradePrice"/>
3719       </sca-c:bindings>
3720       <part name="parameters" element="tns:GetLastTradePrice">
3721         </part>
3722     </message>
3723
3724     <message name="GetLastTradePriceResponse">
3725       <sca-c:bindings>
3726         <sca-c:struct name="getTradePriceResponse"/>
3727       </sca-c:bindings>
3728       <part name="parameters" element="tns:GetLastTradePriceResponse">
3729         </part>
3730     </message>
3731
3732     <portType name="StockQuote">
3733       <sca-c:bindings>
3734         <sca-c:prefix name="stockQuote"/>
3735       </sca-c:bindings>
3736       <operation name="GetLastTradePrice">
3737         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3738           </input>
3739         <output name="GetLastTradePriceResponse"
3740                message="tns:GetLastTradePriceResponse">
3741           </output>
3742       </operation>
3743     </portType>
3744 </definitions>
```

3745

3746 Generated C header file:

```
3747 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
```

```

3748 *           serviceName="StockQuoteService") */
3749
3750 /* @WebOperation(operationName="GetLastTradePrice",
3751 *           response="getLastTradePriceResponse"
3752 *           action="urn:GetLastTradePrice") */
3753 struct getLastTradePrice {
3754     wchar_t *tickerSymbol; /* Since the length of the element is not
3755 *                           * restricted, a pointer is returned with the
3756 *                           * actual value held by the SCA runtime. */
3757 };
3758
3759 struct getLastTradePriceResponse {
3760     float return;
3761 };

```

3762 *Snippet D-8: Example <sca-c:struct> Element*

3763 D.6 <sca-c:parameter>

3764 <sca-c:parameter> specifies the name of the C function parameter or struct member associated with a
3765 specific WSDL message part or wrapper child element.

3766 **Format:**

```

3767 <sca-c:parameter name="CParameter" part="WSDLPart"
3768 childElementName="WSDLElement" type="CType"/>

```

3769 *Snippet D-9: <sca-c:parameter> Element Format*

3770 where:

- 3771 • **parameter/@name : NCName (1..1)** – specifies the name of the C function parameter or struct
3772 member associated with this WSDL operation part or wrapper child element. “return” is used to
3773 denote the return value.
- 3774 • **parameter/@part : string (1..1)** - an XPath expression identifying the wsdl:part of a wsdl:message.
- 3775 • **parameter/@childElementName : QName (1..1)** – specifies the qualified name of a child element of
3776 the global element identified by parameter/@part.
- 3777 • **parameter/@type : string (0..1)** – specifies the type of the parameter or struct member or return
3778 type. The @type attribute of a <parameter/> element MUST be either a C type specified in Simple
3779 Content Binding or, if the message part has complex content, a struct following the mapping specified
3780 in Complex Content Binding. [CD0002] The default type is determined by the mapping defined in
3781 Data Binding.

3782 **Applicable WSDL element(s):**

- 3783 • wsdl:portType/wsdl:operation

3784 **Example:**

3785 **Input WSDL file:**

```

3786 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3787             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3788             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3789             xmlns:tns="http://www.example.org/"
3790             targetNamespace="http://www.example.org/">
3791
3792     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3793             xmlns:tns="http://www.example.org/"
3794             attributeFormDefault="unqualified"
3795             elementFormDefault="unqualified"
3796             targetNamespace="http://www.example.org/">
3797         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3798         <xs:element name="GetLastTradePriceResponse"
3799             type="tns:GetLastTradePriceResponse"/>

```

```

3800     <xs:complexType name="GetLastTradePrice">
3801         <xs:sequence>
3802             <xs:element name="symbol" type="xs:string"/>
3803         </xs:sequence>
3804     </xs:complexType>
3805     <xs:complexType name="GetLastTradePriceResponse">
3806         <xs:sequence>
3807             <xs:element name="return" type="xs:float"/>
3808         </xs:sequence>
3809     </xs:complexType>
3810 </xs:schema>
3811
3812 <message name="GetLastTradePrice">
3813     <part name="parameters" element="tns:GetLastTradePrice">
3814     </part>
3815 </message>
3816
3817 <message name="GetLastTradePriceResponse">
3818     <part name="parameters" element="tns:GetLastTradePriceResponse">
3819     </part>
3820 </message>
3821
3822 <portType name="StockQuote">
3823     <sca-c:bindings>
3824         <sca-c:prefix name="stockQuote"/>
3825     </sca-c:bindings>
3826     <operation name="GetLastTradePrice">
3827         <sca-c:bindings>
3828             <sca-c:function name="getLastTradePrice"/>
3829             <sca-c:parameter name="tickerSymbol"
3830                 part="tns:GetLastTradePrice/parameters"
3831                 childElementName="symbol"/>
3832         </sca-c:bindings>
3833         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3834         </input>
3835         <output name="GetLastTradePriceResponse"
3836             message="tns:GetLastTradePriceResponse">
3837         </output>
3838     </operation>
3839 </portType>
3840
3841 <binding name="StockQuoteServiceSoapBinding">
3842     <soap:binding style="document"
3843         transport="http://schemas.xmlsoap.org/soap/http"/>
3844     <operation name="GetLastTradePrice">
3845         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3846         <input name="GetLastTradePrice">
3847             <soap:body use="literal"/>
3848         </input>
3849         <output name="GetLastTradePriceResponse">
3850             <soap:body use="literal"/>
3851         </output>
3852     </operation>
3853 </binding>
3854
3855 <service name="StockQuoteService">
3856     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3857         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3858     </port>
3859 </service>
3860 </definitions>

```

3861
3862 Generated C header file:

```

3863  /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3864  *          serviceName="StockQuoteService") */
3865
3866  /* @WebFunction(operationName="GetLastTradePrice",
3867  *          action="urn:GetLastTradePrice")
3868  * @WebParam(paramName="tickerSymbol", name="symbol") */
3869  float getLastTradePrice(const wchar_t *tickerSymbol);

```

3870 *Snippet D-10: Example <sca-c:parameter> Element*

3871 **D.7 JAX-WS WSDL Extensions**

3872 An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL
3873 extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.
3874 Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted, and their corresponding SCA
3875 WSDL extension. [CD0007]

3876

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

3877 *Table D-1: Allowed JAX-WS Extensions*

3878 **D.8 sca-wsdlext-c-1.1.xsd**

```

3879 <?xml version="1.0" encoding="UTF-8"?>
3880 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3881   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3882   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3883   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3884   elementFormDefault="qualified">
3885
3886   <element name="bindings" type="sca-c:BindingsType" />
3887   <complexType name="BindingsType">
3888     <choice minOccurs="0" maxOccurs="unbounded">
3889       <element ref="sca-c:prefix" />
3890       <element ref="sca-c:enableWrapperStyle" />
3891       <element ref="sca-c:function" />
3892       <element ref="sca-c:struct" />
3893       <element ref="sca-c:parameter" />
3894     </choice>
3895   </complexType>
3896
3897   <element name="prefix" type="sca-c:PrefixType" />
3898   <complexType name="PrefixType">
3899     <attribute name="name" type="xsd:string" use="required" />
3900   </complexType>
3901
3902   <element name="function" type="sca-c:FunctionType" />
3903   <complexType name="FunctionType">
3904     <attribute name="name" type="xsd:NCName" use="required" />
3905   </complexType>
3906
3907   <element name="struct" type="sca-c:StructType" />

```

```
3908 <complexType name="StructType">
3909   <attribute name="name" type="xsd:NCName" use="required" />
3910 </complexType>
3911
3912 <element name="parameter" type="sca-c:ParameterType" />
3913 <complexType name="ParameterType">
3914   <attribute name="part" type="xsd:string" use="required" />
3915   <attribute name="childElementName" type="xsd:QName" use="required" />
3916   <attribute name="name" type="xsd:NCName" use="required" />
3917   <attribute name="type" type="xsd:string" use="optional" />
3918 </complexType>
3919
3920 <element name="enableWrapperStyle" type="xsd:boolean" />
3921
3922 </schema>
```

3923 *Snippet D-11: SCA C WSDL Extension Schema*

3924

E XML Schemas

3925

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

```
3926 <?xml version="1.0" encoding="UTF-8"?>
3927 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3928         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3929         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3930         elementFormDefault="qualified">
3931
3932     <include schemaLocation="sca-core.xsd" />
3933
3934     <element name="interface.c" type="sca:CInterface"
3935             substitutionGroup="sca:interface" />
3936
3937     <complexType name="CInterface">
3938         <complexContent>
3939             <extension base="sca:Interface">
3940                 <sequence>
3941                     <element name="function" type="sca:CFunction"
3942                             minOccurs="0" maxOccurs="unbounded" />
3943                     <element name="callbackFunction" type="sca:CFunction"
3944                             minOccurs="0" maxOccurs="unbounded" />
3945                     <any namespace="##other" processContents="lax"
3946                             minOccurs="0" maxOccurs="unbounded" />
3947                 </sequence>
3948                 <attribute name="header" type="string" use="required" />
3949                 <attribute name="callbackHeader" type="string" use="optional" />
3950             </extension>
3951         </complexContent>
3952     </complexType>
3953
3954     <complexType name="CFunction">
3955         <sequence>
3956             <choice minOccurs="0" maxOccurs="unbounded">
3957                 <element ref="sca:requires" />
3958                 <element ref="sca:policySetAttachment" />
3959             </choice>
3960             <any namespace="##other" processContents="lax" minOccurs="0"
3961                 maxOccurs="unbounded" />
3962         </sequence>
3963         <attribute name="name" type="NCName" use="required" />
3964         <attribute name="requires" type="sca:listOfQNames" use="optional" />
3965         <attribute name="policySets" type="sca:listOfQNames" use="optional" />
3966         <attribute name="oneWay" type="boolean" use="optional" />
3967         <attribute name="exclude" type="boolean" use="optional" />
3968         <attribute name="input" type="NCName" use="optional" />
3969         <attribute name="output" type="NCName" use="optional" />
3970         <anyAttribute namespace="##other" processContents="lax" />
3971     </complexType>
3972
3973 </schema>
```

3974

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

3975

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

```
3976 <?xml version="1.0" encoding="UTF-8"?>
3977 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3978         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">
```

```

3979     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3980     elementFormDefault="qualified">
3981
3982     <include schemaLocation="sca-core.xsd"/>
3983
3984     <element name="implementation.c" type="sca:CImplementation"
3985         substitutionGroup="sca:implementation" />
3986
3987     <complexType name="CImplementation">
3988         <complexContent>
3989             <extension base="sca:Implementation">
3990                 <sequence>
3991                     <element name="operation" type="sca:CImplementationFunction"
3992                         minOccurs="0" maxOccurs="unbounded" />
3993                     <any namespace="##other" processContents="lax"
3994                         minOccurs="0" maxOccurs="unbounded" />
3995                 </sequence>
3996                 <attribute name="module" type="NCName" use="required"/>
3997                 <attribute name="path" type="string" use="optional"/>
3998                 <attribute name="library" type="boolean" use="optional"/>
3999                 <attribute name="componentType" type="string" use="required"/>
4000                 <attribute name="eagerInit" type="boolean" use="optional"/>
4001                 <attribute name="init" type="boolean" use="optional"/>
4002                 <attribute name="destroy" type="boolean" use="optional"/>
4003                 <attribute name="allowsPassByReference" type="boolean"
4004                     use="optional"/>
4005             </extension>
4006         </complexContent>
4007     </complexType>
4008
4009     <complexType name="CImplementationFunction">
4010         <sequence>
4011             <choice minOccurs="0" maxOccurs="unbounded">
4012                 <element ref="sca:requires"/>
4013                 <element ref="sca:policySetAttachment"/>
4014             </choice>
4015             <any namespace="##other" processContents="lax" minOccurs="0"
4016                 maxOccurs="unbounded" />
4017         </sequence>
4018         <attribute name="name" type="NCName" use="required"/>
4019         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4020         <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4021         <attribute name="allowsPassByReference" type="boolean"
4022             use="optional"/>
4023         <attribute name="init" type="boolean" use="optional"/>
4024         <attribute name="destroy" type="boolean" use="optional"/>
4025         <anyAttribute namespace="##other" processContents="lax"/>
4026     </complexType>
4027
4028 </schema>

```

4029 *Snippet E-2: SCA <implementation.c> Schema*

4030 **E.3 sca-contribution-c-1.1.xsd**

```

4031 <?xml version="1.0" encoding="UTF-8"?>
4032 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4033     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4034     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4035     elementFormDefault="qualified">
4036
4037     <include schemaLocation="sca-contributions.xsd"/>
4038

```



```
4039 <element name="export.c" type="sca:CExport "  
4040 substitutionGroup="sca:Export" />  
4041  
4042 <complexType name="CExport">  
4043 <complexContent>  
4044 <attribute name="name" type="QName" use="required" />  
4045 <attribute name="path" type="string" use="optional" />  
4046 </complexContent>  
4047 </complexType>  
4048  
4049 <element name="import.c" type="sca:CImport "  
4050 substitutionGroup="sca:Import" />  
4051  
4052 <complexType name="CImport">  
4053 <complexContent>  
4054 <attribute name="name" type="QName" use="required" />  
4055 <attribute name="location" type="string" use="required" />  
4056 </complexContent>  
4057 </complexType>  
4058  
4059 </schema>
```

4060 *Snippet E-3: SCA <export.c> and <import.c> Schema*

4061

F Normative Statement Summary

4062

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 <code>_Bool</code> 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 <code>@WebService</code> 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-wsdlex-c-1.1.xsd .

4063 Table F-1: SCA C Core Normative Statements

4064 **F.1 Program-Based Normative Statements Summary**

4065 This section contains a list of normative statements related to program-based component
 4066 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.

4067 Table F-2: SCA C Program-Based Normative Statements

4068 **F.2 Annotation Normative Statement Summary**

4069 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 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 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.

4070 *Table F-3: SCA C Annotation Normative Statements*

4071 **F.3 WSDL Extension Normative Statement Summary**

4072 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.

4073 Table F-4: SCA C WSDL Extension Normative Statements

4074 F.4 JAX-WS Normative Statements

4075 The JAX-WS 2.1 specification [JAXWS21] defines normative statements for various requirements defined
4076 by that specification. Table F-5 outlines those normative statements which apply to the WSDL mapping
4077 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]

4078 [A] All references to Java in the conformance point are treated as references to C.

4079 [B] Annotation generation is only necessary if annotations are supported by an SCA implementation.

4080 *Table F-5: JAX-WS Normative Statements that are Applicable to SCA C*

4081 **F.4.1 Ignored Normative Statments**

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

4082 *Table F-6: JAX-WS Normative Statements that Are Not Applicable to SCA C*

4083 **G Migration**

4084 To aid migration of an implementation or clients using an implementation based the version of the Service
4085 Component Architecture for C defined in [SCA C Client and Implementation V1.00](#), this appendix identifies
4086 the relevant changes to APIs, annotations, or behavior defined in V1.00.

4087 **G.1 Implementation.c attributes**

4088 *@location* has been replaced with *@path*.

4089 **G.2 SCALocate and SCALocateMultiple**

4090 `SCALocate()` and `SCALocateMultiple()` have been renamed to `SCAGetReference()`
4091 `SCAGetReferences()` respectively.

4092

H Acknowledgements

4093

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

4094

4095

Participants:

4096

Participant Name	Affiliation
Bryan Aupperle	IBM
Andrew Borley	IBM
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
David Haney	Individual
Mark Little	Red Hat
Jeff Mischkin	Oracle Corporation
Peter Robbins	IBM

4097

I Revision History

4098

[optional; should not be included in OASIS Standards]

4099

Revision

Date

Editor

Changes Made

•

4100