



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

Committee Draft 04 / Public Review Draft 02

21 January 2010

## Specification URIs:

### This Version:

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

### Previous Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd03.html>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd03.doc>  
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd03.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/200903200912>  
<http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901>

## Abstract:

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

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

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

**Status:**

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

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

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

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

---

## Notices

Copyright © OASIS® 2007, ~~2009~~20010. 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.....	14
2.4	Component Type and Component.....	15
2.4.1	Interface.c.....	16
2.4.2	Function and CallbackFunction.....	17
2.4.3	Implementation.c.....	18
2.4.4	Implementation Function.....	19
2.5	Implementing a Service with a Program.....	20
3	Basic Client Model.....	22
3.1	Accessing Services from Component Implementations.....	22
3.2	Accessing Services from non-SCA Component Implementations.....	23
3.3	Calling Service Operations.....	23
3.3.1	Proxy Functions.....	23
3.4	Long Running Request-Response Operations.....	24
3.4.1	Asynchronous Invocation.....	25
3.4.2	Polling Invocation.....	26
3.4.3	Synchronous Invocation.....	27
4	Asynchronous Programming.....	29
4.1	Non-blocking Calls.....	29
4.2	Callbacks.....	29
4.2.1	Using Callbacks.....	30
4.2.2	Callback Instance Management.....	31
4.2.3	Implementing Multiple Bidirectional Interfaces.....	31
5	Error Handling.....	32
6	C API.....	33
6.1	SCA Programming Interface.....	33
6.1.1	SCAGetReference.....	36
6.1.2	SCAGetReferences.....	36
6.1.3	SCAInvoke.....	37
6.1.4	SCAProperty<T>.....	37
6.1.5	SCAGetReplyMessage.....	39
6.1.6	SCAGetFaultMessage.....	40

6.1.7	SCASetFaultMessage .....	41
6.1.8	SCASelf .....	41
6.1.9	SCAGetCallback .....	42
6.1.10	SCAReleaseCallback .....	42
6.1.11	SCAInvokeAsync .....	42
6.1.12	SCAInvokePoll .....	43
6.1.13	SCACheckResponse .....	43
6.1.14	SCACancelInvoke .....	44
6.1.15	SCAEntryPoint .....	44
6.2	Program-Based Implementation Support .....	45
6.2.1	SCAService .....	45
6.2.2	SCAOperation .....	45
6.2.3	SCAMessageIn .....	46
6.2.4	SCAMessageOut .....	46
7	C Contributions .....	47
7.1	Executable files .....	47
7.1.1	Executable in contribution .....	47
7.1.2	Executable shared with other contribution(s) (Export) .....	47
7.1.3	Executable outside of contribution (Import) .....	48
7.2	componentType files .....	49
7.3	C Contribution Extensions .....	50
7.3.1	Export.c .....	50
7.3.2	Import.c .....	50
8	C Interfaces .....	51
8.1	Types Supported in Service Interfaces .....	51
8.1.1	Local Service .....	51
8.1.2	Remotable Service .....	51
8.2	Restrictions on C header files .....	52
9	WSDL to C and C to WSDL Mapping .....	53
9.1	Interpretations for WSDL to C Mapping .....	53
9.1.1	Definitions .....	53
9.1.2	PortType .....	53
9.1.3	Operations .....	54
9.1.4	Types .....	55
9.1.5	Fault .....	55
9.1.6	Service and Port .....	55
9.1.7	XML Names .....	55
9.2	Interpretations for C to WSDL Mapping .....	55
9.2.1	Package .....	56
9.2.2	Class .....	56
9.2.3	Interface .....	56
9.2.4	Method .....	56
9.2.5	Method Parameters and Return Type .....	57
9.2.6	Service Specific Exception .....	58
9.2.7	Generics .....	58

9.3 Data Binding .....	58
9.3.1 Simple Content Binding.....	58
9.3.2 Complex Content Binding.....	64
10 Conformance .....	69
10.1 Conformance Targets.....	69
10.2 SCA Implementations.....	69
10.3 SCA Documents .....	70
10.4 C Files.....	70
10.5 WSDL Files.....	70
A C SCA Annotations.....	71
A.1 Application of Annotations to C Program Elements.....	71
A.2 Interface Header Annotations .....	71
A.2.1 @Interface .....	72
A.2.2 @Function.....	72
A.2.3 @Operation.....	73
A.2.4 @Remotable .....	75
A.2.5 @Callback.....	75
A.2.6 @OneWay .....	76
A.3 Implementation Annotations.....	76
A.3.1 @ComponentType.....	76
A.3.2 @Service.....	77
A.3.3 @Reference.....	78
A.3.4 @Property.....	78
A.3.5 @Init.....	80
A.3.6 @Destroy .....	80
A.3.7 @EagerInit.....	81
A.3.8 @AllowsPassByReference .....	82
A.4 Base Annotation Grammar.....	82
B C SCA Policy Annotations.....	84
B.1 General Intent Annotations .....	84
B.2 Specific Intent Annotations.....	86
B.2.1 Security Interaction .....	86
B.2.2 Security Implementation .....	87
B.2.3 Reliable Messaging.....	87
B.2.4 Transactions.....	87
B.2.5 Miscellaneous .....	88
B.3 Policy Set Annotations .....	90
B.4 Policy Annotation Grammar Additions .....	91
B.5 Annotation Constants.....	91
C C WSDL Annotations .....	92
C.1 Interface Header Annotations .....	92
C.1.1 @WebService.....	92
C.1.2 @WebFunction .....	93
C.1.3 @WebOperation.....	95
C.1.4 @OneWay .....	97

C.1.5 @WebParam .....	99
C.1.6 @WebResult.....	101
C.1.7 @SOAPBinding .....	103
C.1.8 @WebFault .....	104
C.1.9 @WebThrows .....	106
D    C WSDL Mapping Extensions .....	107
D.1 <sca-c:bindings> .....	107
D.2 <sca-c:prefix>.....	107
D.3 <sca-c:enableWrapperStyle>.....	109
D.4 <sca-c:function> .....	110
D.5 <sca-c:struct>.....	112
D.6 <sca-c:parameter> .....	113
D.7 JAX-WS WSDL Extensions .....	115
D.8 sca-wsdlext-c-1.1.xsd.....	116
E    XML Schemas .....	117
E.1 sca-interface-c-1.1.xsd.....	117
E.2 sca-implementation-c-1.1.xsd .....	118
E.3 sca-contribution-c-1.1.xsd .....	119
F    Normative Statement Summary .....	121
F.1 Program-Based Normative Statements Summary .....	126
F.2 Annotation Normative Statement Summary.....	126
F.3 WSDL Extension Normative Statement Summary .....	128
F.4 JAX-WS Normative Statements .....	128
F.4.1 Ignored Normative Statments .....	133
G    Migration.....	135
G.1 Implementation.c attributes.....	135
G.2 SCALocate and SCALocateMultiple.....	135
H    Acknowledgements .....	136
I    Revision History.....	137

# 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.

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

Prefix	Namespace	Notes
xs	"http://www.w3.org/2001/XMLSchema"	Defined by XML Schema 1.0 specification
sca	"http://docs.oasis-open.org/ns/opencsa/sca/ <del>200903</del> "200912"	Defined by the SCA specifications
sca-c	"http://docs.oasis-open.org/ns/opencsa/sca-c-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 ~~0305~~, *Service Component Architecture Assembly Model Specification Version 1.1*, ~~March 2009~~ January 2010. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-ed03cd05.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](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/>



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

## 39 1.3 Conventions

### 40 1.3.1 Naming Conventions

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

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

### 52 1.3.2 Typographic Conventions

53 This specification follows ~~some~~-typographic conventions for ~~some~~-specific constructs:

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

---

## 2 Basic Component Implementation Model

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

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

### 2.1 Implementing a Service

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

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

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

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

The following snippets [Snippet 2-1](#) and [Snippet 2-2](#) show the C service interface and the C functions of a C implementation.

~~Service interface:~~

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

~~Implementation:~~

~~[Snippet 2-1: A C Service Interface](#)~~

```
#include "LoanService.h"

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

~~The following snippet [Snippet 2-2: C Service Implementation](#)~~

~~[Snippet 2-3](#) shows the component type for this component implementation.~~

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

```

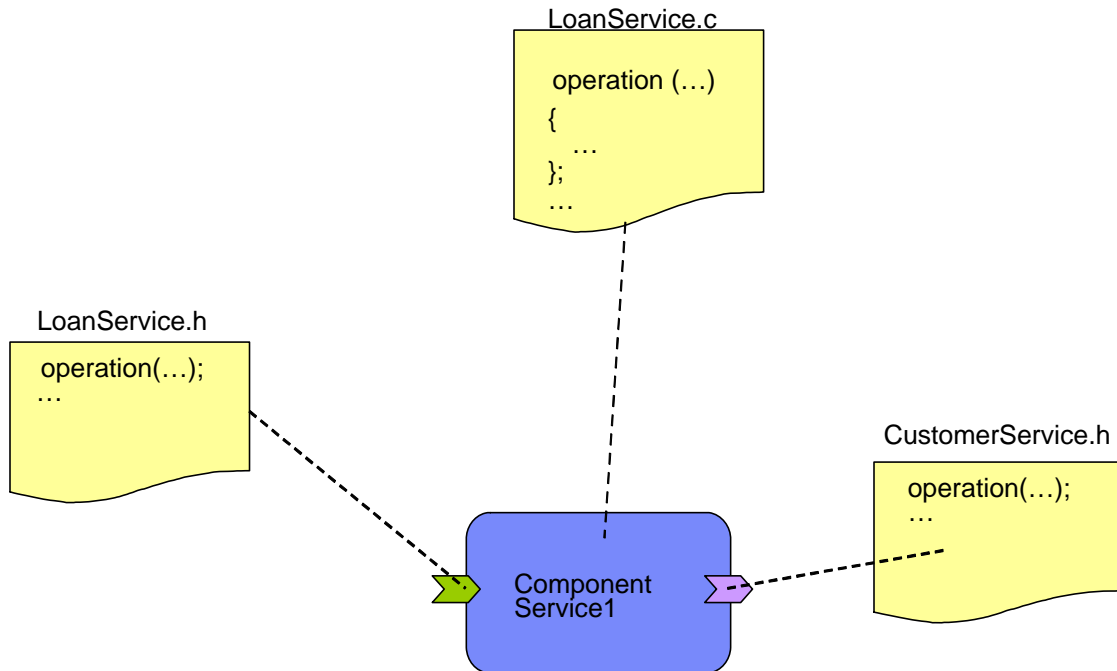
103 | _____ <interface.c header="LoanService.h"/>
104 | _____ </service>
105 | _____ </componentType>

```

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

107 | The following picture

108 | [Figure 2-1](#) shows the relationship between the C header files and implementation files for a component  
 109 | that has a single service and a single reference.



111 | *Figure 2-1: Relationship of C Implementation Artifacts*

## 2.1.1 Implementing a Remotable Service

114 | A `@remotable="true"` attribute on an `interface.c` element indicates that the interface is **remotable** as  
 115 | described in the Assembly Specification [ASSEMBLY]. The following snippet [Snippet 2-4](#) shows the  
 116 | component type for a remotable service:

```

117 |
118 | <?xml version="1.0" encoding="ASCII"?>
119 | <componentType xmlns="http://docs.oasis-
120 | open.org/ns/opencsa/sca/200903" 200912">
121 |   _____ <service name="LoanService">
122 |     _____ <interface.c header="LoanService.h" remotable="true"/>
123 |     _____ </service>
124 |   _____ </componentType>

```

125 | *Snippet 2-4: Component Type for a Remotable Service*

## 2.1.2 AllowsPassByReference

127 | Calls to remotable services have by-value semantics. This means that input parameters passed to the  
 128 | service can be modified by the service without these modifications being visible to the client. Similarly, the  
 129 | return value or exception from the service can be modified by the client without these modifications being  
 130 | visible to the service implementation. For remote calls (either cross-machine or cross-process), these

131 semantics are a consequence of marshalling input parameters, return values and exceptions “on the wire”  
132 and unmarshalling them “off the wire” which results in physical copies being made. For local calls within  
133 the same operating system address space, C calling semantics include by-reference and therefore do not  
134 provide the correct by-value semantics for SCA remotable interfaces. To compensate for this, the SCA  
135 runtime can intervene in these calls to provide by-value semantics by making copies of any by-reference  
136 values passed.

137

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

### 146 2.1.2.1 Marking services and references as “allows pass by reference”

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

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

154

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

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

### 162 2.1.2.2 Using “allows pass by reference” to optimize remotable calls

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

166

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

### 171 2.1.3 Implementing a Local Service

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

## 173 2.2 Component and Implementation **Scopes**Lifecycles

174 Component implementations ~~can either have to~~ manage their own state ~~or allow~~. A library can be loaded  
175 as early as when any component implemented by the SCA runtime to do so. Inlibrary enters the latter  
176 case, SCA defines the concept running state [ASSEMBLY] but no later than the first function invocation of

177 ~~implementation scope, which specifies a service provided by a component implemented by the visibility~~  
178 ~~and lifecycle contract an implementation has with the runtime. Invocations on a service offered by a~~  
179 ~~component will be dispatched by the SCA runtime to an implementation instance according to the~~  
180 ~~semantics of its scope.~~

181

182 ~~Scopes are specified using the @scope attribute of the implementation.c element.~~

183

184 ~~When a scope is library. Component implementations can not specified in an implementation file, the~~  
185 ~~SCA runtime will interpret the implementation scope as **stateless** make any assumptions about when a~~  
186 ~~library might be unloaded. An SCA runtime MUST NOT perform any synchronization of access to~~  
187 ~~component implementations. [C20015]~~

188

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

191

192 ~~The following snippet shows the component type for Component implementations can also specify~~  
193 ~~**lifecycle functions** which are called when a composite-scoped component:~~

194

```
195 <component name="LoanService">  
196 <implementation.c module="loan" componentType="LoanService"  
197 scope="composite"/>  
198 </using the implementation enters the running state or the component>
```

199

200 ~~Certain scoped implementations potentially also specify **lifecycle functions** which are called when an~~  
201 ~~implementation is instantiated or the scope is expired leaves running state. An implementation is either~~  
202 ~~instantiated initialized eagerly when the scope is started component enters the running state (specified by~~  
203 ~~@scope="composite" @eagerInit="true"), or lazily when the first client request is received. Lazy~~  
204 ~~instantiation is the default for all scopes. The C implementation uses the @init="true" attribute of an~~  
205 ~~implementation function element to denote the function to be called upon initialization and the~~  
206 ~~@destroy="true" attribute for the function to be called when the scope ends exiting the running state. A C~~  
207 ~~implementation MUST only designate functions with no arguments and a void return type as lifecycle~~  
208 ~~functions. [C20004]~~

### 209 **1.1.1 Stateless scope**

210 ~~For stateless scope. If an implementation is used by components, there is no implied correlation between~~  
211 ~~implementation instances used to dispatch service requests.~~

212

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

217

218 ~~Lifecycle functions that are not defined for stateless implementations.~~

### 219 **1.1.2 Composite scope**

220 ~~All service requests are dispatched to the same implementation instance for the lifetime of the containing~~  
221 ~~composite, i.e. the binary implementing the component is loaded into memory once and all requests are~~  
222 ~~processed by this single instance. The lifetime of the containing composite is defined as the time in a~~  
223 ~~domain-level composite **[ASSEMBLY]**, it becomes active in the runtime to the time it is deactivated, either~~  
224 ~~normally or abnormally.~~

225  
226 A composite-scoped implementation can also specify eager initialization using the `@eagerInit="true"`  
227 attribute on the `implementation.c` element of a component definition. When marked for eager initialization,  
228 the composite-scoped instance will be created when its containing component is started.

229  
230 The concurrency model for the composite scope is multi-threaded. An SCA runtime MAY run multiple  
231 threads in a single composite-scoped implementation instance object and it MUST NOT perform any  
232 synchronization. [C20015]

233  
234 Composite scope supports both `@init="true"` and `@destroy="true"` functions is possible for a lifecycle  
235 function to be called multiple times.

## 236 2.3 Implementing a Configuration Property

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

241  
242 The following code extract [Snippet 2-5](#) shows how to get the property values.

```
243  
244 #include "SCA.h"  
245  
246 void clientFunction()  
247 {  
248  
249     ___ ...  
250  
251     int32_t loanRating;  
252     int values, compCode, reason;  
253  
254     ...  
255  
256     SCAPropertyInt(L"maxLoanValue", &loanRating, &values, &compCode,  
257 &reason);  
258  
259     ___ ...  
260  
261 }
```

262 [Snippet 2-5: Retrieving a Property Value](#)

263  
264 If the property is many valued, an array of the appropriate type is used as the second parameter, and  
265 The SCA runtime populates the third elements of the array with the configured values, using a stride  
266 based on `<T>` and a `size` parameter would point to an `int` that would receive the number of values. The  
267 typevalue for the property SHOULD NOT allow more values to be defined than strings and binary data  
268 (see `SCAProperty<T>`) or the size of `struct` resulting from the array-default mapping in the  
269 implementation case of complexTypes (see Complex Content Binding). On input, the `num_values`  
270 parameter indicates the number of configured values the client has memory to receive. On output, this  
271 parameter will indicated the actual number of configured values available. If this number exceeds the  
272 input value, only the input value will be returned and `compCode` and `reason` will be set to indicate that  
273 additional values exist.

274  
275 If `<T>` is `Bytes`, `Chars`, `CChars`, `String` or `CString` and the property is many valued, the `size`  
276 parameter is also an array. On input only the first value of the array is relevant – indicating the width of

277 [each member of the value array. On return, for each returned configured value, the value of the size](#)  
278 [array is the number of bytes of characters in the corresponding configured value. If this number exceeds](#)  
279 [the input value, the configured value is truncated and `compCode` and `reason` will be set to indicate the](#)  
280 [data truncation.](#)

## 281 2.4 Component Type and Component

282 For a C component implementation, a component type is specified in a side file. By default, the  
283 `componentType` side file is in the root directory of the composite containing the component or some  
284 subdirectory of the composite root directory with a name specified on the `@componentType` attribute.  
285 The location can be modified as described [below in Implementation.c.](#)

286

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

289

290 [The following snippets Snippet 2-6 and Snippet 2-7](#) show a C service interface and a C implementation of  
291 a service.

292

```
293 /* LoanService interface */  
294 char approveLoan(long customerNumber, long loanAmount);
```

295

296 [Implementation.](#)

297 [Snippet 2-6: A C Service Interface](#)

298

```
299 #include "LoanService.h"  
300  
301 char approveLoan(long customerNumber, long loanAmount)  
302 {  
303     _____  
304     _____  
305 }
```

306

307 [The following snippet Snippet 2-7: C Service Implementation](#)

308

309 [Snippet 2-8](#) shows the component type for this component implementation.

310

```
311 <?xml version="1.0" encoding="ASCII"?>  
312 <componentType xmlns="http://docs.oasis-  
313 open.org/ns/opencsa/sca/200903" >200912">  
314     <service name="LoanService">  
315         <interface.c header="LoanService.h" />  
316     </service>  
317 </componentType>
```

318

319 [The following snippet Snippet 2-8: Component Type for Service Implementation in Snippet 2-7](#)

320

321 [Snippet 2-9](#) shows the component using the implementation.

322

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

```

324 <composite xmlns="http://docs.oasis-
325 open.org/ns/opencsa/sca/200903-200912"
326
327     name="LoanComposite" >
328
329     ...
330
331     <component name="LoanService">
332         <implementation.c module="loan" componentType="LoanService" />
333     </component>
334
335     ...
336
337 </composite>

```

338 [Snippet 2-9: Component Using Implementation in Snippet 2-7](#)

## 339 2.4.1 Interface.c

340 The following snippet [Snippet 2-10](#) shows the [pseudo-schema](#) for the C interface element used to type  
341 services and references of component types.

```

342
343 <?xml version="1.0" encoding="ASCII"?>
344 <!-- interface.c schema snippet -->
345 <interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903-200912"
346     header="string" remotable="boolean"? callbackHeader="string"?
347     requires="listOfQNames"? policySets="listOfQNames"? >
348
349     <function ... />*
350     <callbackFunction ... />*
351     <requires/>*
352     <policySetAttachment/>*
353
354 </interface.c>

```

355 [Snippet 2-10: Pseudo-schema for C Interface Element](#)

356

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

- 358 • **header** : **string (1..1)** – full name of the header file, including either a full path, or its equivalent, or a  
359 relative path from the composite root. This header file describes the interface.
- 360 • **callbackHeader** : **string (0..1)** – full name of the header file that describes the callback interface,  
361 including either a full path, or its equivalent, or a relative path from the composite root.
- 362 • **remotable** : **boolean (0..1)** – indicates whether the service is remotable or local. The default is local.  
363 See Implementing a Remotable Service
- 364
- 365 • **requires** : **listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification  
366 [\[POLICY\]](#) for a description of this attribute. If intents are specified at both the interface and function  
367 level, the effective intents for the function is determined by merging the combined intents from the  
368 function with the combined intents for the interface according to the Policy Framework rules for  
369 merging intents within a structural hierarchy, with the function at the lower level and the interface at  
370 the higher level.
- 371 • **policySets** : **listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification  
372 [\[POLICY\]](#) for a description of this attribute.

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

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



- 375 • **callbackFunction** : **CFunction (0..n)** – see Function and CallbackFunction
- 376 • **requires** : **requires (0..n)** - See the Policy Framework specification **[POLICY]** for a description of this
- 377 **element.**
- 378 • **policySetAttachment** : **policySetAttachment (0..n)** - See the Policy Framework specification
- 379 **[POLICY]** for a description of this element.

## 380 2.4.2 Function and CallbackFunction

381 ~~Some functions~~ A **function** of an interface **might** have behavioral characteristics, ~~which will be described~~  
 382 ~~later,~~ that need to be identified. -This is done using a *function* or *callbackFunction* child element of  
 383 *interface.c*. -These child elements are also used when not all functions in a header file are part of the  
 384 interface or when the interface is implemented by a program.

- 385 • If the header file identified by the `@header` attribute of an `<interface.c/>` element contains function **or**  
 386 **struct** declarations that are not operations of the interface, then the functions **or structs** that **define** **are**  
 387 **not** operations of the interface **MUST** be **identified** **excluded** using `<function/>` child elements of the  
 388 `<interface.c/>` element: **with `@exclude="true"`.** [C20006]
- 389 • If the header file identified by the `@callbackHeader` attribute of an `<interface.c/>` element contains  
 390 function **or struct** declarations that are not operations of the callback interface, then the functions **or**  
 391 **structs** that **define** **are not** operations of the callback interface **MUST** be **identified** **excluded** using  
 392 `<callbackFunction/>` child elements of the `<interface.c/>` element: **with `@exclude="true"`.** [C20007]
- 393 • ~~If the header file identified by the `@header` or `@callbackHeader` attribute of an `<interface.c/>` element~~  
 394 ~~defines the operations of the interface (callback interface) using message formats, then all functions~~  
 395 ~~of the interface (callback interface) **MUST** be identified using `<function/>` (`<callbackFunction/>`) child~~  
 396 ~~elements of the `<interface.c/>` element. Snippet 2-11 [C20008]~~

397

398 The following snippet shows the *interface.c* **pseudo**-schema with the **pseudo**-schema for the *function* and  
 399 *callbackFunction* child elements:

```
400
401 <?xml version="1.0" encoding="ASCII"?>
402 <!-- interface.c schema snippet -->
403 <interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"... >
404
405     <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
406         oneWay="Boolean"? exclude="Boolean"?
407         input="NCName"? output="NCNAME"? -/>*NCName" ? >
408         <requires/>*
409         <policySetAttachment/>*
410     </function> *
411
412     <callbackFunction name="NCName" requires="listOfQNames"?
413         policySets="listOfQNames"? oneWay="Boolean"? exclude="Boolean"?
414         input="NCName"? output="NCName"? -/>
415         <requires/>*
416         <policySetAttachment/>*
417     </callbackFunction> *
418
419 </interface.c>
```

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

421

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

- 423 • **name** : **NCName (1..1)** – name of the **function** **operation** being decorated ~~or included in~~. **If the**  
 424 **interface** **operation** **is implemented as a function, this is the function name.** The `@name` attribute of a

425 `<function/>` child element of a `<interface.c/>` MUST be unique amongst the `<function/>` elements of  
426 that `<interface.c/>`. [C20009]

- 427 • `@name` attribute of a `<callbackFunction/>` child element of a `<interface.c/>` MUST be unique  
428 amongst the `<callbackFunction/>` elements of that `<interface.c/>`. [C20010]
- 429
- 430 • `requires : listOfQNames (0..1)` – a list of `policy` intents. See the Policy Framework specification  
431 [POLICY] needed by this function for a description of this attribute.
- 432 • `policySets : listOfQNames (0..1)` – a list of policy sets. See the Policy Framework specification  
433 [POLICY] for a description of this attribute.
- 434 • `oneWay : boolean (0..1)` – see Non-blocking Calls
- 435 • `exclude : boolean (0..1)` – if true, the function or message struct is excluded from the interface. The  
436 default is false.
- 437 • `input : NCNAME (0..1)` – name of the request message struct if it not the same as the operation  
438 name. (See If the header file identified by the `@header` or `@callbackHeader` attribute of an  
439 `<interface.c/>` element defines the operations of the interface (callback interface) using message  
440 formats, then the `struct` defining the input message format MUST be identified using an `@input`  
441 attribute. Implementing a Service with a Program [C20011] (See-)
- 442 • `output : NCNAME (0..1)` – name of the response message struct if it not the same as the operation  
443 name "Response" appended.
- 444 The `function` and `callbackFunction` elements have the *child elements*:
- 445 • `requires : requires (0..n)` - See the Policy Framework specification Implementing a Service with a  
446 Program [POLICY]
- 447 • `output : NCNAME (0..1)` – for a description of this element.
- 448 • `policySetAttachment : policySetAttachment (0..n)` - See the Policy Framework specification If the  
449 header file identified by the `@header` or `@callbackHeader` attribute of an `<interface.c/>` element  
450 defines the operations of the interface (callback interface) using message formats, then the `struct`  
451 defining the output message format MUST be identified using an `@input` attribute. [POLICY] [C20012]
- 452 • for a description of this element.

## 453 2.4.3 Implementation.c

454 The following snippet Snippet 2-12 shows the `pseudo`-schema for the C implementation element used to  
455 define the implementation of a component.

```
456  
457 <?xml version="1.0" encoding="ASCII"?>  
458 <!-- implementation.c schema snippet -->  
459 <implementation.c xmlns="http://docs.oasis-  
460 open.org/ns/opencsa/sca/200903200912"  
461 module="NCName" library="boolean"? path="string"?  
462 scope="scope"? componentType="string" allowsPassByReference="Boolean"?  
463 eagerInit="Boolean"? init="Boolean"? destroy="Boolean"?  
464 requires="listOfQNames"? policySets="listOfQNames"? >  
465  
466 <function ... />*  
467 <requires/>*  
468 <policySetAttachment/>*  
469  
470 </implementation.c>
```

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

472

473 The `implementation.c` element has the following attributes:

- 474 • **module** : *NCName (1..1)* – name of the binary executable for the service component. This is the root  
475 name of the module.
- 476 • **library** : *boolean (0..1)* – indicates whether the service is implemented as a library or a program. The  
477 default is library. See [Implementing a Service with a Program](#)
- 478 • **path** : *string (0..1)* – path to the module which is either relative to the root of the contribution  
479 containing the composite or is prefixed with a contribution import name and is relative to the root of  
480 the import. See [C Contributions](#).
- 481 • ~~**scope** : *ImplementationScope (0..1)* – indicates the scope of the component implementation. The  
482 default is stateless. Component and Implementation Scopes~~
- 483 • **componentType** : *string (1..1)* – name of the componentType file. A “.componentType” extension  
484 will be appended. A path to the componentType file which is relative to the root of the contribution  
485 containing the composite or is prefixed with a contribution import name and is relative to the root of  
486 the import (see [C Contributions](#)) can be included.
- 487 • **allowsPassByReference** : *boolean (0..1)* – indicates the implementation allows pass by reference  
488 data exchange semantics on calls to it or from it. These semantics apply to all services provided by  
489 and references used by an implementation. See [AllowsPassByReference](#)
- 490 • **eagerInit** : *boolean (0..1)* – indicates a composite scoped implementation is to be initialized when it  
491 is loaded. See [Composite-scoped Component and Implementation Lifecycles](#)
- 492 • **init** : *boolean (0..1)* – indicates program is to be called with an initialize flag to initialize the  
493 implementation. See [Component and Implementation Scopes Lifecycles](#)
- 494 • **destroy** : *boolean (0..1)* – indicates is to be called with a destroy flag to to cleanup the  
495 implementation. See [Component and Implementation Scopes Lifecycles](#)
- 496
- 497 • ~~**requires** : *listOfQNames (0..1)* – a list of policy intents. See the Policy Framework specification  
498 [\[POLICY\]](#) for a description of this attribute. If intents are specified at both the implementation and  
499 function level, the effective intents for the function is determined by merging the combined intents  
500 from the function with the combined intents for the implementation according to the Policy Framework  
501 rules for merging intents within a structural hierarchy, with the function at the lower level and the  
502 implementation at the higher level.~~
- 503 • ~~**policySets** : *listOfQNames (0..1)* – a list of policy sets. See the Policy Framework specification  
504 [\[POLICY\]](#) for a description of this attribute.~~
- 505 The **interface.c** element has the following child elements:
- 506 • **function** : *ImplementationFunction (0..n)* – see [Implementation Function](#)
- 507 • ~~**requires** : *requires (0..n)* - See the Policy Framework specification [\[POLICY\]](#) for a description of this  
508 element.~~
- 509 • ~~**policySetAttachment** : *policySetAttachment (0..n)* - See the Policy Framework specification  
510 [\[POLICY\]](#) for a description of this element.~~

## 511 2.4.4 Implementation Function

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

514

515 The following snippet [Snippet 2-13](#) shows the *implementation.c* pseudo-schema with the pseudo-schema  
516 for a *function* child element:

517

```
518 <?xml version="1.0" encoding="ASCII"?>
519 <!-- ImplementationFunction schema snippet -->
520 <implementation.c xmlns="http://docs.oasis-
521 open.org/ns/opencsa/sca/200903200912"... >
```

```

522 <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
523
524     allowsPassByReference="Boolean"?
525     _____ init="Boolean"?
526     _____ destroy="Boolean"? →
527     <requires/*
528     <policySetAttachment/*
529     </function> *
530
531 </implementation.c>

```

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

533

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

- 535 • **name : NCName (1..1)** – name of the function being decorated. The @name attribute of a  
536 <function/> child element of a <implementation.c/> MUST be unique amongst the <function/>  
537 elements of that <implementation.c/>. [C20013]
- 538 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification  
539 [POLICY] ~~needed by~~ for a description of this ~~function~~ attribute.
- 540 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification  
541 [POLICY] for a description of this attribute.
- 542 • **allowsPassByReference : boolean" (0..1)** – indicates the function allows pass by reference data  
543 exchange semantics. See AllowsPassByReference
- 544 • **init : boolean (0..1)** – indicates this function is to be called to initialize the implementation. See  
545 Component and Implementation SeopesLifecycles
- 546 • **destroy : boolean (0..1)** – indicates this function is to be called to cleanup the implementation. See  
547 Component and Implementation SeopesLifecycles

548 The function element has the child elements:

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

## 553 2.5 Implementing a Service with a Program

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

559

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

565

566 Since a program can implement multiple services and often will implement multiple operations, the  
567 program has to query the runtime to determine which service and operation caused the program to be  
568 invoked. -This is done using SCAService() and SCAOperation(). -Once the specific service and  
569 operation is known, the proper input message can be retrieved using SCAMessageIn(). -Once the logic

570 of the operation is finished SCAMessageOut ( ) is used to provide the return data to the runtime to be  
571 marshalled.

572

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

578

579 [The following shows Snippet 2-14](#) a program implementing a service using these support functions.

580

```
581 #include "SCA.h"
582 #include "myInterface.h"
583 main () {
584     wchar_t myService [255];
585     wchar_t myOperation [255];
586     int compCode, reason;
587     struct FirstInputMsg myFirstIn;
588     struct FirstOutputMsg myFirstOut;
589
590
591     SCAService(myService, &compCode, &reason);
592
593     SCAOperation(myOperation, &compCode, &reason);
594
595     if (wcsncmp(myOperation,L"myFirstOperation")==0){
596         SCAMessageIn(myService, myOperation,
597                     sizeof(struct FirstInputMsg), (void *)&myFirstIn,
598                     &compCode, &reason);
599         ...
600         SCAMessageOut(myService, myOperation,
601                       sizeof(struct FirstOutputMsg),(void *)&myFirstOut,
602                       &compCode, &reason);
603     }
604     else
605     {
606         ...
607     }
608 }
```

609 [Snippet 2-14: C Service Implementation in a Program](#)

---

## 610 3 Basic Client Model

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

### 613 3.1 Accessing Services from Component Implementations

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

615

616 *The following shows [Snippet 3-1](#) the SCAGetReference() function used for this.*

617

```
618 void SCAGetReference(wchar_t *referenceName, SCAREF *referenceToken,  
619                    int *compCode, int *reason);  
620 void SCAInvoke(SCAREF referenceToken, wchar_t *operationName,  
621              int inputMsgLen, void *inputMsg,  
622              int outputMsgLen, void *outputMsg, int *compCode, int *reason);
```

623

624 *The following [Snippet 3-1: Partial SCA API Definition](#)*

625

626 *[Snippet 3-2](#) shows a sample of how a service is called in a C component implementation.*

627

```
628 #include "SCA.h"  
629  
630 void clientFunction()  
631 {  
632  
633     SCAREF serviceToken;  
634     int compCode, reason;  
635     long custNum = 1234;  
636     short rating;  
637  
638     ...  
639     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);  
640     SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),  
641             (void *)&custNum, sizeof(rating), (void *)&rating,  
642             &compCode, &reason);  
643  
644 }
```

645 *[Snippet 3-2: Using SCAGetReference](#)*

646

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

649

```
650 SCAREF *tokens;  
651 int num_targets;  
652 ...  
653 myFunction(...) {  
654     int compCode, reason;  
655     ...  
656     SCAGetReferences(L"myReference", &tokens, &num_targets, &compCode,  
657                    &reason);  
658     for (i = 0; i < num_targets; i++)
```



```

659     {
660         SCAInvoke(tokens[i], L"myOperation", sizeof(inputMsg),
661                 (void *)&inputMsg, 0, NULL, &compCode, &reason);
662     };
663 };

```

664 [Snippet 3-3: Using SCAGetReferences](#)

665

## 666 **3.2 Accessing Services from non-SCA component** 667 **implementationsComponent Implementations**

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

670

671 [The following Snippet 3-4](#) shows a sample of how a service is called in non-SCA C code.

672

```

673 #include "SCA.h"
674
675 void externalFunction()
676 {
677     SCAREF serviceToken;
678     int compCode, reason;
679     long custNum = 1234;
680     short rating;
681
682     SCAEntryPoint(L"customerService", L"http://example.com/mydomain",
683                 &serviceToken, &compCode, &reason);
684     SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),
685             (void *)&custNum, sizeof(rating), (void *)&rating,
686             &compCode, &reason);
687 }

```

688 [Snippet 3-4: Using SCAEntryPoint](#)

689

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

692

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

- 694 • componentName/serviceName/bindingName

## 695 **3.3 Calling Service Operations**

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

698

699 If you have access to a service whose interface is marked as remotable, then on calls to operations of  
700 that service you will experience remote semantics. Arguments and return values are passed by-value and  
701 it is possible to get a SCA\_SERVICE\_UNAVAILABLE reason code which is a Runtime error.

### 702 **3.3.1 Proxy Functions**

703 It is more natural to use specific function calls than the generic SCAInvoke() API for invoking operations.  
704 An SCA runtime typically needs to be involved when a client invokes an operation, particularly if the  
705 service is remote. Proxy functions provide a mechanism for using specific function calls and still allow the

706 | necessary SCA runtime processing. -However, proxies require generated code and managing additional  
707 | source files, so use of proxies is not always desirable.

708 |

709 | For SCA, proxy functions have the form:

```
710 | <functionReturn> SCA_<functionName>( SCAREF referenceToken,  
711 | <functionParameters> )
```

712 | where:

- 713 | • <functionName> is the name of interface function
- 714 | • <functionParameters> are the parameters of the interface function
- 715 | • <functionReturn> is the return type of the interface function

716 |

### 717 | Snippet : Proxy Function Format

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

- 719 | • `ENOENT` if a remote service is unavailable
- 720 | • `EFAULT` if a fault is returned by the operation

721 |

722 | The following Snippet 3-5 shows a sample of using a proxy function.

723 |

```
724 | #include "SCA.h"  
725 |  
726 | void clientFunction()  
727 | {  
728 |  
729 |     SCAREF serviceToken;  
730 |     int compCode, reason;  
731 |     long custNum = 1234;  
732 |     short rating;  
733 |  
734 |     ...  
735 |     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);  
736 |     errno = 0;  
737 |     rating = SCA_getCreditRating(serviceToken, custNum);  
738 |     if (errno) {  
739 |         /* handle error or fault */  
740 |     }  
741 |     else {  
742 |         ...  
743 |     }  
744 | }  
745 |
```

746 | Snippet 3-5: Using a Proxy Function

747 |

748 | An SCA implementation MAY support proxy functions. [C30001]

## 749 | 3.4 Long Running Request-Response Operations

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



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

### 759 3.4.1 Asynchronous Invocation

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

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

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

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

769  
770 For program-based component implementations, the handler parameter is set to an empty string and  
771 when the SCA runtime starts the program to process the response, a call to `SCAService()` returns the  
772 name of the reference and a call to `SCAOperation()` returns the name of the reference operation.

773

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

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

776 the asynchronous invocation style includes a proxy function

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

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

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

780

781 **The following** [Snippet 3-8](#) shows a sample of how the asynchronous invocation style is used in a C  
782 component implementation.

783

```
784 #include "SCA.h"  
785 #include "TravelService.h"  
786  
787 SCAREF serviceToken;  
788 int compCode, reason;  
789  
790 void makeReservationsHandler(short rspType)  
791 {  
792     struct confirmationData cd;  
793     wchar_t *fault, *faultDetails;  
794  
795     if (rspType == SCA_REPLY_MESSAGE) {  
796         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);  
797         ...  
798     }  
799     else {  
800         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,  
801                             &faultDetails, &compCode, &reason);  
802         if (wstricmpwcsncmp(*fault, L"noFlight") {  
803             ...  
804         }  
805     }  
806     else {
```

```

806     ...
807     }
808 }
809
810     return;
811 }
812
813 void clientFunction()
814 {
815     struct itineraryData id;
816
817     ...
818
819     void (*ah)(short) = &makeReservationsHandler;
820
821     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
822
823     SCAInvokeAsync(serviceToken, L"makeReservations", sizeof(itineraryData),
824                   (void *)&id, ah, &compCode, &reason);
825
826
827     return;
828 }

```

829 [Snippet 3-8: Using Asynchronous Invocation](#)

### 830 3.4.2 Polling Invocation

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

835  
836 If proxy functions are supported, for a service operation with signature

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

838 the polling invocation style includes a proxy function

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

840 [Snippet 3-9: Asynchronous Pooling Proxy Function Format](#)

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

842  
843 [The following Snippet 3-10](#) shows a sample of how the polling invocation style is used in a C component  
844 implementation.

```

845
846 #include "SCA.h"
847 #include "TravelService.h"
848
849 void pollingClientFunction()
850 {
851     SCAREF serviceToken;
852     int compCode, reason;
853     short rspType;
854
855     struct itineraryData id;
856     struct confirmationData cd;
857     wchar_t *fault, *faultDetails;
858
859     ...

```

```

860
861     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
862
863     SCAInvokePoll(serviceToken, L"makeReservations", sizeof(itineraryData),
864         (void *) &id), &compCode, &reason);
865
866     SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
867     while (!rspType) {
868         // do something, then wait for some time...
869         SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
870     }
871     if (rspType == SCA_REPLY_MESSAGE) {
872         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
873         ...
874     }
875     else {
876         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
877             &faultDetails, &compCode, &reason);
878         if (wcsncmp(*fault, L"noFlight") {
879             ...
880         }
881         else {
882             ...
883         }
884     }
885
886     return;
887 }

```

888 [Snippet 3-10: Using Asynchronous Polling Invocation](#)

### 889 3.4.3 Synchronous Invocation

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

895

896 [The following Snippet 3-11](#) shows a sample of how the synchronous invocation style is used in a C  
897 component implementation.

898

```

899     #include "SCA.h"
900     #include "TravelService.h"
901
902     void synchronousClientFunction()
903     {
904         SCAREF serviceToken;
905         int compCode, reason;
906
907         struct itineraryData id;
908         struct confirmationData *cd;
909         wchar_t *fault, *faultDetails;
910
911         ...
912
913         SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
914
915         SCAInvoke(serviceToken, L"makeReservations", sizeof(itineraryData),
916             (void *)&id, sizeof(confirmationData), (void *)&cd,
917             &compCode, &reason);
918         if (compCode == SCA_FAULT) {

```

```
919     ...
920   }
921   else {
922     SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
923                       &faultDetails, &compCode, &reason);
924   if (wetwcscmp(*fault, L"noFlight") {
925     ...
926   }
927   else {
928     ...
929   }
930 }
931
932 return;
933 }
```

934 | [Snippet 3-11: Using Synchronous Invocation for an Asynchronous Operation](#)

---

## 935 4 Asynchronous Programming

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

### 943 4.1 Non-blocking Calls

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

947  
948 Any function that returns `void` and has only by-value parameters can be marked with the  
949 `@oneWay="true"` attribute in the interface definition of the service. An operation marked as `oneWay` is  
950 considered non-blocking and the SCA runtime MAY use a binding that buffers the requests to the function  
951 and sends them at some time after they are made. [C40001]

952  
953 The following snippet [Snippet 4-1](#) shows the component type for a service with the `reportEvent()`  
954 function declared as a one-way operation:

```
955  
956 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912">  
957   <service name="LoanService">  
958     <interface.c header="LoanService.h">  
959       <function name="reportEvent" oneWay="true" />  
960     </interface.c>  
961   </service>  
962 </componentType>
```

963 [Snippet 4-1: ComponentType with oneWay Function](#)

964  
965 SCA does not currently define a mechanism for making non-blocking calls to functions that return values.  
966 It is considered to be a best practice that service designers define one-way operations as often as  
967 possible, in order to give the greatest degree of binding flexibility to deployers.

### 968 4.2 Callbacks

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

971  
972 A callback interface is declared by the `@callbackHeader` and `@callbackFunctions` attributes in the  
973 interface definition of the service. The following snippet [Snippet 4-2](#) shows the component type for a  
974 service *MyService* with the interface defined in *MyService.h* and the interface for callbacks defined in  
975 *MyServiceCallback.h*,

```
976  
977 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912">  
978 >  
979   <service name="MyService">  
980     <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h" />
```

```
981     </service>
982 </componentType>
```

983 [Snippet 4-2: ComponentType with a Callback Interface](#)

## 984 4.2.1 Using Callbacks

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

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

```
994
995     double requestQuotation(char *productCode,int quantity);
996
997     char *getState();
998     char *getZipCode();
999     char *getCreditRating();
```

1000 [Snippet 4-3: C Interface with a Callback Interface](#)

1001  
1002 In ~~this example~~ [Snippet 4-3](#), the `requestQuotation` operation requests a quotation to supply a given  
1003 quantity of a specified product. -The `QuotationCallback` interface provides a number of operations that the  
1004 supplier can use to obtain additional information about the client making the request. -For example, some  
1005 suppliers might quote different prices based on the state or the zip code to which the order will be  
1006 shipped, and some suppliers might quote a lower price if the ordering company has a good credit rating.  
1007 Other suppliers might quote a standard price without requesting any additional information from the client.

1008  
1009 ~~The following code snippet~~ [Snippet 4-4](#) illustrates a possible implementation of the example service.

```
1010
1011     #include "QuotationCallback.h"
1012     #include "SCA.h"
1013
1014     double requestQuotation(char *productCode,int quantity) {
1015         double price, discount = 0;
1016         char state[3], creditRating[4];
1017         SCAREF callbackRef;
1018         int compCode, reason;
1019
1020         price = getPrice(productQuote, quantity);
1021
1022         ___SCAGetCallback(L"", &callbackRef, &compCode, &reason);
1023         SCAInvoke(callbackRef, L"getState", 0, NULL, sizeof(state), state,
1024                 &compCode, &reason);
1025         if (quantity > 1000 && strcmp(state,"FL") == 0)
1026             discount = 0.05;
1027         SCAInvoke(callbackRef, L"getCreditRating", 0, NULL, sizeof(creditRating),
1028                 creditRating, &compCode, &reason);
1029         if (quantity > 10000 && creditRating[0] == 'A')
1030             discount += 0.05;
1031         ___SCAReleaseCallback(callbackRef, &compCode, &reason);
1032         return price * (1-discount);
1033     }
```

1034

1035 *The code snippet below Snippet 4-4: Implementation of Forward Service with Interface in Snippet 4-3*

1036

1037 *Snippet 4-5* is taken from the client of this example service. -The client's service implementation **class**  
1038 implements the functions of the QuotationCallback interface as well as those of its own service interface  
1039 ClientService.

1040

```
1041 #include "QuotationCallback.h"
1042 #include "SCA.h"
1043
1044 char state[3] = "TX", zipCode[6] = "78746", creditRating[3] = "AA";
1045
1046 aClientFunction() {
1047     SCAREF serviceToken;
1048     int compCode, reason;
1049
1050     SCAGetReference(L"quotationService", &serviceToken, &compCode, &reason);
1051
1052     SCA_requestQuotation(serviceToken, "AB123", 2000);
1053 }
1054
1055 char *getState() {
1056     return state;
1057 }
1058 char *getZipCode() {
1059     return zipCode;
1060 }
1061 char *getCreditRating() {
1062     return creditRating;
1063 }
```

1064 *Snippet 4-5: Implementation of Callback Interface in Snippet 4-3*

1065

1066 In this example the callback is **stateless**, i.e., the callback requests do not need any information relating  
1067 to the original service request. -For a callback that needs information relating to the original service  
1068 request (a **stateful** callback), this information can be passed to the client by the service provider as  
1069 parameters on the callback request.

## 1070 4.2.2 Callback Instance Management

1071 ~~Instance management for callback requests received by the client of the bidirectional service is~~  
1072 ~~handled as described in the same way as instance management for regular service requests. If the client~~  
1073 ~~implementation has STATELESS scope, the callback is dispatched using a newly initialized instance. If~~  
1074 ~~the client implementation has COMPOSITE scope, the callback is dispatched using the same shared~~  
1075 ~~instance that is used to dispatch regular service requests.~~

1076

1077 ~~As described~~ Using Callbacks, a stateful callback can obtain information relating to the original service  
1078 request from parameters on the callback request. -Alternatively, a ~~composite-scoped~~ client could store  
1079 information relating to the original request as ~~instance~~ data and retrieve it when the callback request is  
1080 received. -These approaches could be combined by using a key passed on the callback request (e.g., an  
1081 order ID) to retrieve information that was stored ~~in a composite-scoped instance~~ by the client code that  
1082 made the original request.

## 1083 4.2.3 Implementing Multiple Bidirectional Interfaces

1084 Since it is possible for a single component to implement multiple services, it is also possible for callbacks  
1085 to be defined for each of the services that it implements. The service name parameter of  
1086 SCAGetCallback() identifies the service for which the callback is to be obtained.

## 1087 5 Error Handling

1088 Clients calling service operations will experience business logic errors, and SCA runtime errors.

1089

1090 Business logic errors are generated by the implementation of the called service operation. They are  
1091 handled by client the invoking the operation of the service.

1092

1093 SCA runtime errors are generated by the SCA runtime and signal problems in the management of the  
1094 execution of components, and in the interaction with remote services. The SCA C API includes two return  
1095 codes on every function, a completion code and a reason code. -The reason code is used to provide  
1096 more detailed information if a function does not complete successfully. -Currently the following SCA codes  
1097 are defined:

1098

```
1099 /* Completion Codes */
1100 #define SCACC_OK 0
1101 #define SCACC_WARNING 1
1102 #define SCACC_FAULT 2
1103 #define SCACC_ERROR 3
1104
1105 /* Reason Codes */
1106 #define SCA_SERVICE_UNAVAILABLE 1
1107 #define SCA_MULTIPLE_SERVICES 2
1108 #define SCA_DATA_TRUNCATED 3
1109 #define SCA_PRAMETER_ERROR 4
1110 #define SCA_BUSY 45
1111 #define SCA_RUNTIME_ERROR 6
1112 #define SCA_ADDITIONAL_VALUES 7
1113
1114 /* Response Types */
1115 #define SCA_NO_RESPONSE 0
1116 #define SCA_REPLY_MESSAGE 1
1117 #define SCA_FAULT_MESSAGE 2
```

1118 *Snippet 5-1: SCA Constant Defintions*

1119

1120 Reason codes between 0 and 100 are reserved for use by this specification. -Vendor defined reason  
1121 codes SHOULD start at 101. [C50001]



1122

## 6 C API

1123

### 6.1 SCA Programming Interface

1124

The following shows the C interface declarations for synchronous programming.

1125

The SCA API definition is:

1126

1127

```
typedef void *SCAREF;  
  
void SCAGetReference(wchar_t *referenceName,  
                    SCAREF *referenceToken,  
                    int *compCode,  
                    int *reason);  
  
void SCAGetReferences(wchar_t *referenceName,  
                     SCAREF **referenceTokens,  
                     int *num_targets,  
                     int *CompCodeCompCode,  
                     int *Reason);  
  
void SCAInvoke(SCAREF token,  
              wchar_t *operationName,  
              int inputMsgLen,  
              void *inputMsg,  
              int *outputMsgLen,  
              void *outputMsg,  
              int *compCode,  
              int *reason);  
  
void SCAPropertyBoolean(wchar_t *propertyName,  
                       char *value,  
                       int *num_values,  
                       int *compCode,  
                       int *reason);  
  
void SCAPropertyByte(wchar_t *propertyName,  
                    int8_t *value,  
                    int *num_values,  
                    int *compCode,  
                    int *reason);  
  
void SCAPropertyBytes(wchar_t *propertyName,  
                     int8_t **value,  
                     int *size,  
                     int *num_values,  
                     int *compCode,  
                     int *reason);  
  
void SCAPropertyChar(wchar_t *propertyName,  
                    wchar_t *value,  
                    int *num_values,  
                    int *compCode,  
                    int *reason);  
  
void SCAPropertyChars(wchar_t *propertyName,  
                      wchar_t **value,  
                      int *size,  
                      int *num_values,
```

1177

```

1178         int *compCode,
1179         int *reason);
1180
1181 void SCAPPropertyCChar(wchar_t *propertyName,
1182                       char *value,
1183                       int *num_values,
1184                       int *compCode,
1185                       int *reason);
1186
1187 void SCAPPropertyCChars(wchar_t *propertyName,
1188                        char **value,
1189                        int *size,
1190                        int *num_values,
1191                        int *compCode,
1192                        int *reason);
1193
1194 void SCAPPropertyShort(wchar_t *propertyName,
1195                       int16_t *value,
1196                       int *num_values,
1197                       int *compCode,
1198                       int *reason);
1199
1200 void SCAPPropertyInt(wchar_t *propertyName,
1201                    int32_t *value,
1202                    int *num_values,
1203                    int *compCode,
1204                    int *reason);
1205
1206 void SCAPPropertyLong(wchar_t *propertyName,
1207                      int64_t *value,
1208                      int *num_values,
1209                      int *compCode,
1210                      int *reason);
1211
1212 void SCAPPropertyFloat(wchar_t *propertyName,
1213                       float *value,
1214                       int *num_values,
1215                       int *compCode,
1216                       int *reason);
1217
1218 void SCAPPropertyDouble(wchar_t *propertyName,
1219                        double *value,
1220                        int *num_values,
1221                        int *compCode,
1222                        int *reason);
1223
1224 void SCAPPropertyString(wchar_t *propertyName,
1225                        wchar_t **value,
1226                        int *size,
1227                        int *num_values,
1228                        int *compCode,
1229                        int *reason);
1230
1231 void SCAPPropertyCString(wchar_t *propertyName,
1232                          char **value,
1233                          int *size,
1234                          int *num_values,
1235                          int *compCode,
1236                          int *reason);
1237
1238 void SCAPPropertyStruct(wchar_t *propertyName,
1239                        void **value,
1240                        int *num_values,
1241                        int *compCode,

```

```

1242         int *reason);
1243
1244 void SCAGetReplyMessage(SCAREF token,
1245     int *bufferLen,
1246     charvoid *buffer,
1247     int *compCode,
1248     int *reason);
1249
1250 void SCAGetFaultMessage(SCAREF token,
1251     int *bufferLen,
1252     wchar_t **faultName,
1253     charvoid *buffer,
1254     int *compCode,
1255     int *reason);
1256
1257 void SCASetFaultMessage(wchar_t *serviceName,
1258     wchar_t *operationName,
1259     wchar_t *faultName,
1260     int bufferLen,
1261     charvoid *buffer,
1262     int *compCode,
1263     int *reason);
1264
1265 void SCASelf(wchar_t *serviceName,
1266     SCAREF *serviceToken,
1267     int *compCode,
1268     int *reason);
1269
1270 void SCAGetCallback(wchar_t *serviceName,
1271     SCAREF *serviceToken,
1272     int *compCode,
1273     int *reason);
1274
1275 void SCAReleaseCallback(SCAREF serviceToken,
1276     int *compCode,
1277     int *reason);
1278
1279 void SCAINvokeAsync(SCAREF token,
1280     wchar_t *operationName,
1281     int inputMsgLen,
1282     void *inputMsg,
1283     void (*handler)(short ++),
1284     int *compCode,
1285     int *reason);
1286
1287 void SCAINvokePoll(SCAREF token,
1288     wchar_t *operationName,
1289     int inputMsgLen,
1290     void *inputMsg,
1291     int *compCode,
1292     int *reason);
1293
1294 void SCACheckResponse(SCAREF token,
1295     short *responseType,
1296     int *compCode,
1297     int *reason);
1298
1299 void SCACancelInvoke(SCAREF token,
1300     int *compCode,
1301     int *reason);
1302
1303 void SCAEntryPoint(wchar_t *serviceURI,
1304     wchar_t *domainURI,
1305     SCAREF *serviceToken,

```

1306  
1307

```
int *compCode,  
int *reason);
```

1308

1309 *The C synchronous programming interface has the following functions:*

1310 *[Snippet 6-1: SCA API Definition](#)*

### 1311 6.1.1 SCAGetReference

1312 A C component implementation uses SCAGetReference( ) to initialize a Reference before invoking any  
1313 operations of the Reference.

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

1314 *[Table 6-1: SCAGetReference Details](#)*

### 1315 6.1.2 SCAGetReferences

1316 A C component implementation uses SCAGetReferences( ) to initialize a Reference that might be  
1317 bound to multiple Services before invoking any operations of the Reference.

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

1318 [Table 6-2: SCAGetReferencse Details](#)

### 1319 6.1.3 SCAInvoke

1320 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	<del>token</del> Token	Token returned by prior SCAGetReference() or SCAGetReferences(), SCASelf() or SCAGetCallback() call.
	operationName	Name of the operation to invoke
	inputMsgLen	Length of the request message buffer
	inputMsg	Request message
In/Out Parameter	outputMsgLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	outputMsg	Response message
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the response data was truncated. The buffer size needs to be increased and SCAGetReplyMessage() called with the larger buffer. SCACC_FAULT, if the operation returned a business fault. SCAGetFaultMessage() needs to be called to get the fault details. SCACC_ERROR, otherwise – see reason for details
	Reason	SCA_DATA_TRUNCATED if the response data was truncated SCA_PARAMETER_ERROR if the operationName is not defined for the interface SCA_SERVICE_UNAVAILABLE if the provider for the interface is no longer operational
Post Condition	Unless a SCA_SERVICE_UNAVAILABLE reason is returned, the token remains valid for subsequent calls.	

1321 [Table 6-3: SCAInvoke Details](#)

### 1322 6.1.4 SCAProperty<T>

1323 A C component implementation uses SCAProperty<T>() to get the configured value for a Property.

1324  
 1325 This API is available for Boolean, Byte, Bytes, Char, Chars, CChar, CChars, Short, Int, Long, Float,  
 1326 Double, String, CString and Struct. -The Char, Chars, and String variants return wchar\_t based data while  
 1327 the CChar, CChars, and CString variants return char based data. The Bytes, Chars, and CChars variants  
 1328 return a buffer of data. -The String and CString variants return a null terminated string.  
 1329

1330 An SCA runtime MAY additionally provide a DataObject variant of this API for handling properties with  
 1331 complex XML types. -The type of the value parameter in this variant is DATAOBJECT. [C60002]

1332

1333 If &lt;T&gt; is one of: Boolean, Byte, Char, CChar, Short, Int, Long, Float, Double or Struct

Precondition	C component instance is running	
Input Parameter	propertyName	Name of the Property value to obtain
Output Parameters	value	Configured value of the property
	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
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
	reasonCompCode	SCA_PARAMETER_ERRORCC_OK, if the propertyName call is not defined for the component or its type is incompatible with <T> 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.	

1334 *Table 6-4: SCAProperty<T> Details for fixed length types*

1335

1336 If &lt;T&gt; is one of: Bytes, Chars, CChars, String or CString

Precondition	C component instance is running	
Input Parameter	propertyName	Name of the Property value to obtain
In/Out Parameter	size	Input: Maximum number of bytes or characters that can be returned Output: Actual number of bytes or characters returned or size needed to hold entire value
Output Parameters	value	Configured value of the property
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer.

		<u>SCACC_ERROR, otherwise – see reason for details</u>
<u>In/Out Parameters</u>	<u>size</u>	<p><u>Input: Maximum number of bytes or characters that can be returned for each configured value</u></p> <p><u>Output: Actual number of bytes or characters returned or size needed to hold a configured value.</u></p> <p><u>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.</u></p>
	<u>num_values</u>	<p><u>Input: Maximum number of configured values that can be returned.</u></p> <p><u>Output: Actual number of configured values</u></p>
<u>Output Parameters</u>	<u>value</u>	<u>Configured value(s) of the property</u>
	<del>reason</del> <u>compCode</u>	<p><u>SCACC_OK, if the call is successful</u></p> <p><u>SCACC_WARNING, if the data was truncated</u></p> <p><del><u>SCA_PARAMETER_ERROR if the propertyName is not defined f or the component or its type is incompatible with &lt;T&gt; number of configured values exceeds the input value of num_values</u></del></p> <p><u>SCACC_ERROR, otherwise – see reason for details</u></p>
	<u>reason</u>	<p><u>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.</u></p> <p><u>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.</u></p> <p><del><u>SCA_PARAMETER_ERROR if the propertyName is not defined for the component or its type is incompatible with &lt;T&gt;</u></del></p>
<u>Post Condition</u>	<u>The configured value of the Property is loaded into the appropriate variable.</u>	

1337 *Table 6-5: SCAProperty<T> Details for variable length types*

1338 **6.1.5 SCAGetReplyMessage**

1339 A C component implementation uses SCAGetReplyMessage ( ) to retrieve the reply message of an  
 1340 operation invocation if the length of the message exceeded the buffer size provided on SCAInvoke ( ).

Precondition	C component instance is running, has a valid token and an <code>SCAInvoke()</code> returned a <code>SCACC_WARNING compCode</code> or has a valid <code>serviceToken</code> and an <code>SCACallback()</code> returned a <code>SCACC_WARNING 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>buffer</code>	Response 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.
Post Condition	The <code>referenceToken</code> remains valid for subsequent calls.	

1341 [Table 6-6: SCAGetReplyMessage Details](#)

## 1342 6.1.6 SCAGetFaultMessage

1343 A C component implementation uses `SCAGetFaultMessage()` to retrieve the details of a business fault  
1344 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
	<del>buffer</del> <u>buffer</u>	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
	<del>Reason</del> <u>reason</u>	<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 <u>not</u> return a business fault



Post Condition	The <code>referenceToken</code> remains valid for subsequent calls.
----------------	---

1345 [Table 6-7: SCAGetFaultMessage Details](#)

### 1346 6.1.7 SCASetFaultMessage

1347 A C component implementation uses `SCASetFaultMessage()` to return a business fault in response to  
 1348 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>	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	<code>reason</code>	SCA_PARAMETER_ERROR 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	

1349 [Table 6-8: SCASetFaultMessage Details](#)

### 1350 6.1.8 SCASelf

1351 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
	<del>reason</del> <u>Reason</u>	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.	

1352 [Table 6-9: SCASelf Details](#)

1353 **6.1.9 SCAGetCallback**

1354 A C component implementation uses `SCAGetCallback()` to initialize a Service before invoking any  
 1355 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
	<del>reason</del> <u>Reason</u>	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.	

1356 [Table 6-10: SCAGetCallback Details](#)

1357 **6.1.10 SCAReleaseCallback**

1358 A C component implementation uses `SCAReleaseCallback()` to tell the SCA runtime it has completed  
 1359 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
	<code>reason</code>	SCA_PARAMETER_ERROR if the <code>serviceToken</code> is not valid
Post Condition	The token becomes invalid for subsequent calls.	

1360 [Table 6-11: SCAReleaseCallbacke Details](#)

1361 **6.1.11 SCAInvokeAsync**

1362 A C component implementation uses `SCAInvokeAsync()` to invoke a long running operation of an  
 1363 interface using the asynchronous style.

Precondition	C component instance is running and has a valid token	
Input Parameters	<code>token</code>	Token returned by prior <code>SCAGetReference()</code> , <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
	<code>operationName</code>	Name of the operation to invoke
	<code>inputMsgLen</code>	Length of the request message buffer
	<code>inputMsg</code>	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.	

1364 [Table 6-12: SCAInvokeAsynch Details](#)

### 1365 6.1.12 SCAInvokePoll

1366 A C component implementation uses SCAInvokePoll() to invoke a long running operation of a  
1367 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
	<del>reason</del> 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.	

1368 [Table 6-13: SCAInvokePoll Details](#)

### 1369 6.1.13 SCACheckResponse

1370 A C component implementation uses SCACheckResponse() to determine if a response to a long  
1371 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 <code>SCALocate()</code> , <code>SCALocateMultiple()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> 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	

1372 [Table 6-14: SCACheckResponse Details](#)

### 1373 6.1.14 SCACancelInvoke

1374 A C component implementation uses `SCACancelInvoke()` to cancel a long running operation request.

Precondition	C component instance is running, has a valid token and has made a <code>SCAInvokeAsync()</code> or <code>SCAInvokePoll()</code> but has not received a response.	
Input Parameter	token	Token returned by prior <code>SCALocate()</code> , <code>SCALocateMultiple()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
Output Parameters	compCode	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.	

1375 [Table 6-15: SCACancelInvokee Details](#)

### 1376 6.1.15 SCAEntryPoint

1377 Non-SCA C code uses `SCAEntryPoint()` to access a Service before invoking any operations of the  
1378 Service.

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

1379 [Table 6-16: SCAEntryPoint Details](#)

## 1380 6.2 Program-Based Implementation Support

1381 ~~A SCA runtime MAY provide the functions SCAService(), SCAOperation(), SCAMessageIn() and~~  
1382 ~~SCAMessageOut() to support C implementations in programs. [C60003]~~

1383 Support for components implemented via C programs is provided by the functions SCAService(),  
1384 SCAOperation(), SCAMessageIn() and SCAMessageOut().

1385

```
1386 void SCAService(wchar_t *serviceName, int *compCode, int *reason);  
1387  
1388 void SCAOperation(wchar_t *operationName, int *compCode, int *reason);  
1389  
1390 void SCAMessageIn(wchar_t *serviceName,  
1391                  wchar_t *operationName,  
1392                  int *bufferLen,  
1393                  void *buffer,  
1394                  int *compCode,  
1395                  int *reason);  
1396  
1397 void SCAMessageOut(wchar_t *serviceName,  
1398                   wchar_t *operationName,  
1399                   int bufferLen,  
1400                   void *buffer,  
1401                   int *CompCode,  
1402                   int *Reason);
```

1403

1404 ~~The C program-based implementation support has the following functions:~~

1405 [Snippet 6-2: SCA API for Program Implementations Definition](#)

### 1406 6.2.1 SCAService

1407 A program-based C component implementation uses SCAService() to determine which service was  
1408 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	

1409 [Table 6-17: SCAService Details](#)

### 1410 6.2.2 SCAOperation

1411 A program-based C component implementation uses SCAOperation() to determine which operation of  
1412 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.	

1413 [Table 6-18: SCAOperation Details](#)

### 1414 6.2.3 SCAMessageIn

1415 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.	

1416 [Table 6-19: SCAaMessgeln Details](#)

### 1417 6.2.4 SCAMessageOut

1418 A program-based C component implementation uses `SCAMessageOut()` to return a reply message.

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

1419 [Table 6-20: SCAMessgOut Details](#)

## 1420 7 C Contributions

1421 Contributions are defined in the Assembly specification [ASSEMBLY] C contributions are typically, but  
1422 not necessarily contained in .zip files. -In addition to SCDL and potentially WSDL artifacts, C contributions  
1423 include binary executable files, componentType files and potentially C interface headers. -No additional  
1424 discussion is needed for header files, but there are ~~some~~ additional considerations for executable and  
1425 componentType files discussed ~~in the following sections~~.

### 1426 7.1 Executable files

1427 Executable files containing the C implementations for a contribution can be contained in the contribution,  
1428 contained in another contribution or external to any contribution. -In some cases, it could be desirable to  
1429 have contributions share an executable. -In other cases, an implementation deployment policy might  
1430 dictate that executables are placed in specific directories in a file system.

#### 1431 7.1.1 Executable in contribution

1432 When the executable file containing a C implementation is in the same contribution, the *@path* attribute of  
1433 the *implementation.c* element is used to specify the location of the executable. -The specific location of an  
1434 executable within a contribution is not defined by this specification.

1435  
1436 ~~The following~~ [Snippet 7-1](#) shows a contribution containing a DLL.

```
1437  
1438 META-INF/  
1439   sca-contribution.xml  
1440 bin/  
1441   _____ autoinsurance.dll  
1442 AutoInsurance/  
1443   _____ AutoInsurance.composite  
1444   _____ AutoInsuranceService/  
1445     _____ AutoInsurance.h  
1446     _____ AutoInsurance.componentType  
1447   _____ include/  
1448     _____ Customers.h  
1449     _____ Underwriting.h  
1450     _____ RateUtils.h
```

1451 [Snippet 7-1: Contribution Containing a DLL](#)

1452  
1453 The SCDL for the AutoInsuranceService component [of Snippet 7-1](#) is:

```
1454  
1455 <component name="AutoInsuranceService">  
1456   <implementation.c module="autoinsurance" path="bin/"  
1457   <_____ componentType="AutoInsurance" />  
1458 </component>
```

1459 [Snippet 7-2: Component Definition Using Implementation in a Common DLL](#)

#### 1460 7.1.2 Executable shared with other contribution(s) (Export)

1461 If a contribution contains an executable that also implements C components found in other contributions,  
1462 the contribution has to export the executable. -An executable in a contribution is made visible to other  
1463 contributions by adding an **export.c** element to the contribution definition as shown in ~~the following~~  
1464 ~~snippet~~ [Snippet 7-3](#).

1465

```
1466 <contribution>
1467 | — <deployable composite="myNS:RateUtilities"
1468 |   <export.c name="contribNS:rates" >
1469 | </contribution>
```

1470 [Snippet 7-3: Exporting a Contribution](#)

1471

1472 It is also possible to export only a subtree of a contribution. ~~If~~For a contribution ~~contains the following~~:

1473

```
1474 META-INF/
1475 |   sca-contribution.xml
1476 | bin/
1477 | — rates.dll
1478 | RateUtilities/
1479 | — RateUtilities.composite
1480 | — RateUtilitiesService/
1481 | ————— RateUtils.h
1482 | ————— RateUtils.componentType
```

1483 [Snippet 7-4: Contribution with a Subdirectory to be Shared](#)

1484

1485 An export of the form:

1486

```
1487 <contribution>
1488 | — <deployable composite="myNS:RateUtilities"
1489 |   <export.c name="contribNS:ratesbin" path="bin/" >
1490 | </contribution>
```

1491 [Snippet 7-5: Exporting a Subdirectory of a Contribution](#)

1492

1493 only makes the contents of the bin directory visible to other contributions. ~~By~~ placing all of the executable  
1494 files of a contribution in a single directory and exporting only that directory, the amount of information  
1495 available to a contribution that uses the exported executable files is limited. ~~This is considered a best~~  
1496 practice.

### 1497 7.1.3 Executable outside of contribution (Import)

1498 When the executable that implements a C component is located outside of a contribution, the contribution  
1499 MUST ~~has to~~ import the executable. ~~If~~ the executable is located in another contribution, the **import.c**  
1500 element of the contribution definition uses a *@location* attribute that identifies the name of the export as  
1501 defined in the contribution that defined the export as shown in ~~the following snippet~~ [Snippet 7-6](#).

1502

```
1503 <contribution>
1504 | — <deployable composite="myNS:Underwriting"
1505 |   <import.c name="rates" location="contribNS:rates">
1506 | </contribution>
```

1507 [Snippet 7-6: Contribution with an Import](#)

1508

1509 The SCDL for the UnderwritingService component [of Snippet 7-6](#) is:

1510

```
1511 <component name="UnderwritingService">
1512 |   <implementation.c module="rates" path="rates:bin/"
1513 |     componentType="Underwriting" />
1514 | </component>
```

1515 [Snippet 7-7: Component Definition Using Implementation in an External DLL](#)



1516

1517 If the executable is located in the file system, the `@location` attribute identifies the location in the files  
1518 system used as the root of the import as shown in [this snippet Snippet 7-8](#).

1519

```
1520 <contribution>  
1521   <deployable composite="myNS:CustomerUtilities"  
1522     <import.c name="usr-bin" location="/usr/bin/" >  
1523   </contribution>
```

1524 [Snippet 7-8: Component Definition Using Implementation in a File System](#)

## 1525 7.2 componentType files

1526 As stated in [section 2.5 Component Type and Component](#), each component implemented in C has a  
1527 corresponding componentType file. -This componentType file is, by default, located in the root directory of  
1528 the composite containing the component or a subdirectory of the composite root with a name specified on  
1529 the `@componentType` attribute as shown in the [following example Snippet 7-9](#).

1530

```
1531 META-INF/  
1532   sca-contribution.xml  
1533 bin/  
1534   autoinsurance.dll  
1535 AutoInsurance/  
1536   AutoInsurance.composite  
1537   AutoInsuranceService/  
1538     AutoInsurance.h  
1539     AutoInsurance.componentType
```

1540 [Snippet 7-9: Contribution with ComponentType](#)

1541

1542 The SCDL for the AutoInsuranceService component [of Snippet 7-9](#) is:

1543

```
1544 <component name="AutoInsuranceService">  
1545   <implementation.c module="autoinsurance" path="bin/"  
1546     componentType="AutoInsurance" />  
1547 </component>
```

1548 [Snippet 7-10: Component Definition with Local ComponentType](#)

1549

1550 Since there is a one-to-one correspondence between implementations and componentTypes, when an  
1551 implementation is shared between contributions, it is desirable to also share the componentType file.  
1552 ComponentType files can be exported and imported in the same manner as executable files. -The  
1553 location of a `.componentType` file can be specified using the `@componentType` attribute of the  
1554 `implementation.c` element.

1555

```
1556 <component name="UnderwritingService">  
1557   <implementation.c library="rates" path="rates:bin/"  
1558     componentType="rates:types/Underwriting" />  
1559 </component>
```

1560 [Snippet 7-11: Component Definition with Imported ComponentType](#)

## 1561 7.3 C Contribution Extensions

### 1562 7.3.1 Export.c

1563 | The following snippet [Snippet 7-12](#) shows the **pseudo**-schema for the C export element used to make an  
1564 executable or componentType file visible outside of a contribution.

1565

```
1566 <?xml version="1.0" encoding="ASCII"?>  
1567 <!-- export.c schema snippet -->  
1568 <export.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"  
1569 name="QName" path="string"? >
```

1570 [Snippet 7-12: Pseudo-schema for C Export Element](#)

1571

1572 The **export.c** element has the following **attributes**:

- 1573 • **name : QName (1..1)** – name of the export. The **@name** attribute of a **<export.c/>** element MUST be  
1574 unique amongst the **<export.c/>** elements in a domain. [C70001]
- 1575 • **path : string (0..1)** – path of the exported executable relative to the root of the contribution. -If not  
1576 present, the entire contribution is exported.

### 1577 7.3.2 Import.c

1578 | The following snippet [Snippet 7-13](#) shows the **pseudo**-schema for the C import element used to reference  
1579 an executable or componentType file that is outside of a contribution.

1580

```
1581 <?xml version="1.0" encoding="ASCII"?>  
1582 <!-- import.c schema snippet -->  
1583 <import.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"  
1584 name="QName" location="string" >
```

1585 [Snippet 7-13: Pseudo-schema for C Import Element](#)

1586

1587 The **import.c** element has the following **attributes**:

- 1588 • **name : QName (1..1)** – name of the import. The **@name** attribute of a **<import.c/>** child element of a  
1589 **<contribution/>** MUST be unique amongst the **<import.c/>** elements in of that contribution. [C70002]
- 1590 • **location : string (1..1)** – either the QName of an export or a file system location. -If the value does not  
1591 match an export name it is taken as an absolute file system path.

## 8 C Interfaces

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

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

```
#if LIB_BUILD
# define DECLSPEC_FUNC(ReturnType) __declspec(dllimport) ReturnType
#else
# define DECLSPEC_FUNC(ReturnType) __declspec(dllexport) ReturnType
#endif
DECLSPEC_FUNC(int) fooFunc(void) {}
```

Snippet 8-1: Example Macro Impacting Function Signature

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

### 7.48.1 Types Supported in Service Interfaces

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

#### 1.2 Local service

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

- Any of the C primitive types (for example, int, short, char). In this case the data will be passed by value as is normal for C.
- Pointers to any of the C primitive types (for example, int \*, short \*, char \*).
- DATAOBJECT. An SDO handle. [C80004]

#### 1.3 Remotable service

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

##### 8.1.1 Local Service

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

##### 8.1.2 Remotable Service

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

- Any of the C primitive types (for example, int, short, char). This will be copied.
- DATAOBJECT. An SDO handle. The SDO will be copied and passed to the destination. Any of the C types specified in Simple Content Binding and Complex Content Binding. These types may be

1632 passed by-value or by-pointer. Unless the function and client indicate that they allow by-reference  
1633 semantics (see AllowsPassByReference), a copy will be explicitly created by the runtime for any  
1634 parameters passed by-pointer.

- 1635 • An SDO DATAOBJECT. This type may be passed by-value or by-pointer. Unless the function and  
1636 client indicate that they allow by-reference semantics (see AllowsPassByReference), a deep-copy of  
1637 the DATAOBJECT will be created by the runtime for any parameters passed by-value or by-pointer.  
1638 When by-reference semantics are allowed, the DATAOBJECT handle will be passed. [C80002]

1639 ~~Unless the interface is marked as allowing pass-by-reference semantics, the behavior of the following are~~  
1640 ~~not defined:~~

- 1641 ~~• Pointers.~~

## 1642 **7.58.2 Restrictions on C header files**

1643 ~~A C header file that is used to describe an interface has some restrictions.~~

1644 A C header file used to define an interface MUST:

- 1645 • ~~Declare declare at least one function or message format struct [C90001]~~

1646

1647 ~~A C header file used to define an interface MUST NOT use the following constructs:~~

- 1648 • ~~Macros [C90002]~~

1649 [C80003]

1650 Function definitions in a header file are not considered part of a service definition.

1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
  
1679  
1680  
1681  
1682  
1683  
  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692

## **89** WSDL to C and C to WSDL Mapping

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

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

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

- References to Java are considered references to C.
- References to Java classes are considered references to a collection of C functions or programs that implement an interface.
- References to Java methods are considered references to C functions or message format struct declarations.
- References to Java interfaces are considered references to a collection of C function or message format struct declarations used to define an interface.
- ~~• For the purposes of the C-to-WSDL mapping algorithm, a C header file with containing function declarations and no annotations is treated as if it had a @WebService annotation. All default values are assumed for the @WebService annotation.~~

### **8.19.1** Interpretations for WSDL to C Mapping

External binding files are not supported.

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

#### **8.1.19.1.1** Definitions

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

MIME binding is not supported.

#### **8.1.29.1.2** PortType

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

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

1693 | If the implementation is contained in a program, the declarations are all struct declarations. -See the next  
1694 | section for details.

1695 |  
1696 | ~~In the absence of customizations, an SCA implementation SHOULD map each portType to separate~~  
1697 | ~~header file. An SCA implementation MAY use any sca-c:prefix binding declarations to control this~~  
1698 | ~~mapping. An SCA implementation MUST map a WSDL portType to a remotable C interface definition.~~  
1699 | ~~[C100023]~~

1700 | In the absence of customizations, an SCA implementation SHOULD map each portType to separate  
1701 | header file. An SCA implementation MAY use any sca-c:prefix binding declarations to control this  
1702 | mapping. [C100001] For example, all portTypes in a WSDL document with a common sca-c:prefix binding  
1703 | declaration could be mapped to a single header file.-

1704 |  
1705 | Header file naming is implementation dependent.

### 1706 | **8.1.39.1.3 Operations**

1707 | Asynchronous mapping is not supported.

#### 1708 | **8.1.3.19.1.3.1 Operation Names**

1709 | WSDL operation names are only guaranteed to be unique with a portType. -C requires function and struct  
1710 | names loaded into an address space to be distinct. -The mapping of operation names to function or struct  
1711 | names have to take this into account.

1712 |  
1713 | For components implemented in libraries, in the absence of customizations, an SCA implementation  
1714 | MUST ~~concatenate~~map an operation name, with the first character converted to lower case, to a function  
1715 | name. If necessary, to avoid name collisions, an SCA implementation MAY prepend the portType name,  
1716 | with the first character converted to lower case, and the operation name, with the first character converted  
1717 | to upper case, to form the function-:name. [C100002]

1718 |  
1719 | An application can customize this mapping using the sca-c:prefix and/or sca-c:function binding  
1720 | declarations.

1721 |  
1722 | For program-based service implementations:

- 1723 | • If the number of **In** parameters plus the number of **In/Out** parameters is greater than one there will be  
1724 | a request struct.
- 1725 | • If the number of **Out** parameters plus the number of **In/Out** parameters is greater than one there will  
1726 | be a response struct.

1727 |  
1728 | For components implemented in a program, in the absence of customizations, an SCA implementation  
1729 | MUST map an operation name, with the first character converted to lowercase to a request struct name. If  
1730 | necessary, to avoid name collisions, an SCA implementation MAY concatenate the portType name, with  
1731 | the first character converted to lower case, and the operation name, with the first character converted to  
1732 | upper case, to form the request struct name. Additionally an SCA implementation MUST append  
1733 | "Response" to the request struct name to form the response struct name. [C100005]

1734 |  
1735 | An application can customize this mapping using the sca-c:prefix and/or sca-c:struct binding declarations.

1736 **8.1.3.29.1.3.2 Message and Part**

1737 In the absence of any customizations for a WSDL operation that does not meet the requirements for the  
1738 wrapped style, the name of a mapped function parameter or struct member MUST be the value of the  
1739 name attribute of the wsdl:part element with the first character converted to lower case. [C100003]

1740

1741 In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped  
1742 style, the name of a mapped function parameter or struct member MUST be the value of the local name  
1743 of the wrapper child with the first character converted to lower case. [C100004]

1744

1745 An application can customize this mapping using the sca-c:parameter binding declaration.

1746

1747 For library-based service implementations, an SCA implementation MUST map **In** parameters as pass  
1748 by-value or const and **In/Out** and **Out** parameters as pass via pointers. [C100019]

1749

1750 For program-based service implementations, an SCA implementation MUST map all values in the input  
1751 message as pass by-value and the updated values for **In/Out** parameters and all **Out** parameters in the  
1752 response message as pass by-value. [C100020]

1753 **8.1.49.1.4 Types**

1754 As per section Data Binding (based on SDO type mapping).

1755

1756 MTOM/XOP content processing is left to the application.

1757 **8.1.59.1.5 Fault**

1758 C has no exceptions so an API is provided for getting and setting fault messages (see  
1759 SCAGetFaultMessage and SCASetFaultMessage). Fault messages are mapped in same manner as  
1760 input and output messages.

1761

1762 In the absence of customizations, an SCA implementation MUST map the name of the message element  
1763 referred to by a fault element to the name of the struct describing the fault message content. If necessary,  
1764 to avoid name collisions, an implementation MAY append "*Fault*" to the name of the message element  
1765 when mapping to the struct name. [C100006]

1766

1767 An application can customize this mapping using the sca-c:struct binding declaration.

1768 **8.1.69.1.6 Service and Port**

1769 This mapping does not define generation of client side code.

1770 **8.1.79.1.7 XML Names**

1771 See comments in Operations

1772 **8.29.2 Interpretations for C to WSDL Mapping**

1773 Where annotations are discusses as a means for an application to control the mapping to WSDL, an  
1774 implementation-specific means of controlling the mapping can be used instead.

1775 **8.2.19.2.1 Package**

1776 Not relevant.

1777

1778 An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be  
1779 configurable. [C100007]

1780 **8.2.29.2.2 Class**

1781 Not relevant since mapping is only based on declarations.

1782 **8.2.39.2.3 Interface**

1783 The declarations in a header file are used to define an interface. A header file can be used to define an  
1784 interface if it satisfies either (for components implemented in libraries):

- 1785
- Contains one or more function declarations
  - 1786 • Any of these functions declarations might carry a @WebFunction annotation
  - 1787 • The parameters and return types of these function declarations are compatible with the C to XML  
1788 Schema mapping in Data Binding

1789 or (for components implemented in programs):

- 1790 • Contains one request message struct declarations
- 1791 • Any of the request message struct declarations might carry a @WebOperation annotation
- 1792 • Any of the request message struct declarations can have a corresponding response message struct,  
1793 identified by either having a name with "Response" appended to the request message struct name or  
1794 identified in a @WebOperation annotation
- 1795 • Members of these struct declarations are compatible with the C to XML Schema mapping in Data  
1796 Binding

1797

1798 ~~In the absence of customizations, an SCA implementation MUST map the header file name to the~~  
1799 ~~portType name. An implementation MAY append "PortType" to the header file name in the mapping to~~  
1800 ~~the portType name. An SCA implementation MUST map a C interface definition to WSDL as if it has a~~  
1801 ~~@WebService annotation with all default values. [C100008]~~

1802

1803 [C100024]

1804 ~~In the absence of customizations, an SCA implementation MUST map the header file name to the~~  
1805 ~~portType name. An implementation MAY append "PortType" to the header file name in the mapping to the~~  
1806 ~~portType name. [C100008]~~

1807 An application can customize this mapping using the @WebService annotation.

1808 **8.2.49.2.4 Method**

1809 For components implemented in libraries, functions map to operations.

1810

1811 In the absence of customizations, an SCA implementation MUST map ~~the~~ function name to ~~the an~~  
1812 operation name, stripping the portType name, if present, and any namespace prefix from the ~~function~~  
1813 ~~name from the~~ front of function name before mapping it to the operation name. [C100009]

1814

1815 An application can customize function to operation mapping or exclude a function from an interface using  
1816 the @WebFunction annotation.

1817



1818 For components implemented in programs, operations are mapped from request structs.

1819  
1820 In the absence of customizations, a struct with a name that does not end in “*Response*” or “*Fault*” is  
1821 considered to be a request message struct and an SCA implementation MUST map the struct name to  
1822 the operation name, stripping the portType name, if present, and any namespace prefix from the front of  
1823 the struct name before mapping it to the operation name. [C100010]

1824  
1825 An application can customize struct to operation mapping or exclude a struct from an interface using the  
1826 @WebOperation annotation.

## 1827 **8.2.59.2.5 Method Parameters and Return Type**

1828 For components implemented in libraries, function parameters and return type map to either message or  
1829 global element components.

1830  
1831 In the absence of customizations, an SCA implementation MUST map the *the\_a* parameter name, if present,  
1832 to the *the\_a* part or global element component name. If the parameter does not have a name the SCA  
1833 implementation MUST use argN as the part or global element child name. [C100011]

1834  
1835 An application can customize parameter to message or global element component mapping using the  
1836 @WebParam annotation.

1837  
1838 In the absence of customizations, an SCA implementation MUST map the return type to a part or global  
1839 element child named “*return*”. [C100012]

1840  
1841 An application can customize return type to message or global element component mapping using the  
1842 @WebReturn annotation.

1843  
1844 An SCA implementation MUST map:

- 1845 • a function’s return value as an **out** parameter.
- 1846 • by-value and const parameters as **in** parameters.
- 1847 • in the absence of customizations, pointer parameters as **in/out** parameters. [C100017]

1848  
1849 An application can customize parameter classification using the @WebParam annotation.

1850  
1851 Program based implementation SHOULD use the Document-Literal style and encoding. [C100013]

1852  
1853 In the absence of customizations, an SCA implementation MUST map the struct member name to the  
1854 part or global element child name. [C100014]

1855  
1856 An application can customize struct member to message or global element component mapping using the  
1857 @WebParam annotation.

- 1858
- 1859 • Members of the request struct that are not members of the response struct are **in** parameters
  - 1860 • Members of the response struct that are not members of the request struct are **out** parameters

- Members of both the request and response structs are **in/out** parameters. Matching is done by member name. An SCA implementation **MUST** ensure that **in/out** parameters have the same type in the request and response structs. [C100015]

### 8.2.69.2.6 Service Specific Exception

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

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

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

### 8.2.79.2.7 Generics

Not relevant.

### ~~1.3.1 Service and Ports~~

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

### 8.39.3 Data Binding

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

### 8.3.19.3.1 Simple Content Binding

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

<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

<i>XSD Schema Type →</i>	<i>C Type</i>	<i>→ XSD Schema Type</i>
ENTITIES	wchar_t *	string
ENTITY	wchar_t *	string
float	float	float
gDay	wchar_t *	string
gMonth	wchar_t *	string
gMonthDay	wchar_t *	string
gYear	wchar_t *	string
gYearMonth	wchar_t *	string
hexBinary	char *	string
ID	wchar_t *	string
IDREF	wchar_t *	string
IDREFS	wchar_t *	string
int	int32_t	int
integer	wchar_t *	string
language	wchar_t *	string
long	int64_t	long
Name	wchar_t *	string
NCName	wchar_t *	string
negativeInteger	wchar_t *	string
NMTOKEN	wchar_t *	string
NMTOKENS	wchar_t *	string
nonNegativeInteger	wchar_t *	string
nonPositiveInteger	wchar_t *	string
normalizedString	wchar_t *	string
NOTATION	wchar_t *	string
positiveInteger	wchar_t *	string
QName	wchar_t *	string
short	int16_t	short
string	wchar_t *	string
time	wchar_t *	string
token	wchar_t *	string
unsignedByte	uint8_t	unsignedByte

<b>XSD Schema Type →</b>	<b>C Type</b>	<b>→ XSD Schema Type</b>
unsignedInt	uint32_t	unsignedInt
unsignedLong	uint64_t	unsignedLong
unsignedShort	uint16_t	unsignedShort

1886 *Table 9-1: XSD simple type to C type mapping*

1887

1888 [Table 9-2 defines the mapping of C++ types to XSD schema types that are not covered in Table 9-1.](#)

1889

<b>C Type →</b>	<b>XSD Schema Type</b>
_Bool	boolean
wchar_t	string
signed char	byte
unsigned char	unsignedByte
short	short
unsigned short	unsignedShort
int	int
unsigned int	unsignedInt
long	long
unsigned long	unsignedLong
long long	long
unsigned long long	unsignedLong
<b>wchar_t*</b>	<b>string</b>
long double	decimal
time_t	time
struct tm	dateTime

1890 *Table 9-2: C type to XSD type mapping*

1891

1892 The C standard does not define value ranges for integer types so it is possible that on a platform  
 1893 parameters or return values could have values that are out of range for the default XSD schema type. In  
 1894 these circumstances, the mapping would need to be customized, using @WebParam or @WebResult if  
 1895 supported, or some other implementation-specific mechanism.

1896

1897 **An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.**

1898 **[C100021]**

1899

1900 **An SCA implementation MAY map boolean to \_Bool by default. [C100022]**

### 1901 **8.3.1-19.3.1.1 WSDL to C Mapping Details**

1902 In general, when `xsd:string` and types derived from `xsd:string` map to a struct member, the  
1903 mapping is to a combination of a `wchar_t *` and a separately allocated data array. -If either the `length`  
1904 or `maxLength` facet is used, then a `wchar_t[]` is used. -If the `pattern` facet is used, this might allow  
1905 the use of `char` and/or also constrain the length.

1906 Example:

```
1907 <xsd:element name="myString" type="xsd:string"/>
```

1908 maps to:

```
1909 wchar_t *myString;  
1910 /* this points to a dynamically allocated buffer with the data */
```

#### 1911 *Snippet 9-1: Unbounded String Mapping*

1912

```
1913 <xsd:simpleType name="boundedString25">  
1914 <xsd:restriction base="xsd:string">  
1915 <xsd:length value="25"/>  
1916 </xsd:restriction>  
1917 </xsd:simpleType>  
1918 ...  
1919 <xsd:element name="myString" type="boundedString25"/>
```

1920 maps to:

```
1921 wchar_t myString[26];
```

#### 1922 *Snippet 9-2: Bounded String Mapping*

1923

- 1924 • When unbounded binary data maps to a struct member, the mapping is to a `char *` that points to  
1925 the location where the actual data is located. -Like strings, if the binary data is bounded in length, a  
1926 `char[]` is used.

1927

1928 Examples:

```
1929 <xsd:element name="myData" type="xsd:hexBinary"/>
```

1930 maps to:

```
1931 char *myData;  
1932 /* this points to a dynamically allocated buffer with the data */
```

#### 1933 *Snippet 9-3: Unbounded Binary Data Mapping*

1934

```
1935 <xsd:simpleType name="boundedData25">  
1936 <xsd:restriction base="xsd:hexBinary">  
1937 <xsd:length value="25"/>  
1938 </xsd:restriction>  
1939 </xsd:simpleType>  
1940 ...  
1941 <xsd:element name="myData" type="boundedData25"/>
```

1942 maps to:

```
1943 char myData[26];
```

#### 1944 *Snippet 9-4: Bounded Binary Data Mapping*

1945

- 1946 • Since C does not have a way of representing unset values, when elements with `minOccurs !=`  
1947 `maxOccurs` and lists with `minLength != maxLength`, which have a variable, but bounded, number

1948 of instances, map to a struct, the mapping is to a count of the number of occurrences and an array. -If  
1949 the count is 0, then the contents of the array is undefined.

1950

1951 Examples:

```
1952 <xsd:element name="counts" type="xsd:int" maxOccurs="5"/>
```

1953 maps to:

```
1954 size_t counts_num;  
1955 int counts[5];
```

1956 *Snippet 9-5: minOccurs != maxOccurs Mapping*

1957

```
1958 <xsd:simpleType name="lineNumList">  
1959   <xsd:list itemType="xsd:int"/>  
1960 </xsd:simpleType>  
1961 <xsd:simpleType name="lineNumList6">  
1962   <xsd:restriction base="lineNumList ">  
1963     <xsd:minLength value="1"/>  
1964     <xsd:maxLength value="6"/>  
1965   </xsd:restriction>  
1966 </xsd:simpleType>  
1967 ...  
1968 <xsd:element name="lineNums" type="lineNumList6"/>
```

1969 maps to:

```
1970 size_t lineNums_num;  
1971 long lineNums[6];
```

1972 *Snippet 9-6: minLength != maxLength Mapping*

1973

- 1974 • Since C does not allow for unbounded arrays, when elements with maxOccurs = unbounded and  
1975 lists without a defined length or maxLength, map to a struct, the mapping is to a count of the  
1976 number of occurrences and a pointer to the location where the actual data is located as an array

1977

1978 Examples:

```
1979 <xsd:element name="counts" type="xsd:int" maxOccurs="unbounded"/>
```

1980 maps to:

```
1981 size_t counts_num;  
1982 int *counts;  
1983 /* this points to a dynamically allocated array of struct tm's */
```

1984

```
1985 <xsd:simpleType name="lineNumList">  
1986   <xsd:list itemType="xsd:int"/>  
1987 </xsd:simpleType>  
1988 ...  
1989 <xsd:element name="lineNums" type="lineNumList"/>
```

1990 maps to:

```
1991 size_t lineNums_num;  
1992 long *lineNums;  
1993 /* this points to a dynamically allocated array of longs */
```

1994 *Snippet 9-7: Unbounded Array Mapping*

1995

- 1996 • Union Types are not supported.

1997 **8.3.1.29.3.1.2 C to WSDL Mapping Details**

- 1998
- wchar\_t[] and char[] map to xsd:string with a maxLength facet.
- 1999
- C arrays map as normal elements but with multiplicity allowed via the minOccurs and maxOccurs facets.
- 2000

2001

2002 Example:

```
2003 long myFunction(char* name, int idList[], double value);  
2004 int idList[];
```

2005 maps to:

```
2006 <xsd:element name="idList" type="xsd:int"  
2007 minOccurs="0" maxOccurs="unbounded" />
```

2008 *Snippet 9-8: Array Mapping*

- 2009
- Multi-dimensional arrays map into nested elements.
- 2010

2011 Example:

```
2012 int multiIdArray[4][2];
```

2013 maps to:

```
2014 <xsd:element name="myFunction">multiIdArray"  
2015 minOccurs="0" maxOccurs="4" />  
2016 <xsd:complexType>  
2017 <xsd:sequence>  
2018 <xsd:element name="multiIdArray" type="xsd:string"/>int "  
2019 <xsd:element name="idList" type="xsd:short"  
2020 minOccurs="2" maxOccurs="unbounded"2" />  
2021 <xsd:element name="value" type="xsd:double"/>  
2022 </xsd:sequence>  
2023 </xsd:complexType>  
2024 </xsd:element>
```

- 2026
- *Snippet 9-9: Multi-dimensional arrays map into nested elements. Dimensional Array Mapping*

2027

2028 Example:

```
2029 long myFunction(int multiIdArray[][4][2]);
```

2030 maps to:

```
2031 <xsd:element name="myFunction">  
2032 <xsd:complexType>  
2033 <xsd:sequence>  
2034 <xsd:element name="multiIdArray"  
2035 minOccurs="0" maxOccurs="unbounded" />  
2036 <xsd:complexType>  
2037 <xsd:sequence>  
2038 <xsd:element name="multiIdArray"  
2039 minOccurs="4" maxOccurs="4" />  
2040 <xsd:complexType>  
2041 <xsd:sequence>  
2042 <xsd:element name="multiIdArray" type="xsd:short"  
2043 minOccurs="2" maxOccurs="2" />  
2044 </xsd:sequence>  
2045 </xsd:complexType>  
2046 </xsd:element>
```

```
2047 </xsd:sequence>
2048 </xsd:complexType>
2049 </xsd:element>
2050 </xsd:sequence>
2051 </xsd:complexType>
2052 </xsd:element>
2053
```

- 2054 • Except as detailed in the table above, pointers do not affect the type mapping, only the classification
2055 as in, out, or in/out.

### 2056 **8.3.29.3.2 Complex Content Binding**

2057 When mapping between XSD complex content types and C, either instances of SDO DataObjects or
2058 structs are used. An SCA implementation MUST support mapping message parts or global elements with
2059 complex types and parameters, return types, and struct members with a type defined by a struct. The
2060 mapping from WSDL MAY be to DataObjects and/or structs. The mapping to and from structs MUST
2061 follow the rules defined in WSDL to C Mapping Details. [C100016]

### 2062 **8.3.2-19.3.2.1 WSDL to C Mapping Details**

- 2063 • Complex types and groups mapped to static DataObjects follow the rules defined in [SDO21].
- 2064 • Complex types and groups mapped to structs have the attributes and elements of the type mapped
2065 to members of the struct.
  - 2066 – The name of the struct is the name of the type or group.
  - 2067 – Attributes appear in the struct before elements.
  - 2068 – Simple types are mapped to members as described above.
  - 2069 – The same rules for variable number of instances of a simple type element apply to complex type
2070 elements.
  - 2071 – A sequence group is mapped as either a simple type or a complex type as appropriate.

2073 Example:

```
2074 <xsd:complexType name="myType">
2075 <xsd:sequence>
2076 <xsd:element name="name">
2077 <xsd:simpleType>
2078 <xsd:restriction base="xsd:string">
2079 <xsd:length value="25"/>
2080 </xsd:restriction>
2081 </xsd:simpleType>
2082 </xsd:element>
2083 <xsd:element name="idList" type="xsd:int"
2084 minOccurs="0" maxOccurs="unbounded"/>
2085 <xsd:element name="value" type="xsd:double"/>
2086 </xsd:sequence>
2087 </xsd:complexType>
```

2088 maps to:

```
2089 struct myType {
2090     wchar_t name[26];
2091     size_t idList_num;
2092     long *idList;
2093     /* this points to a dynamically allocated array of longs */
2094     double value;
2095 };
```

2096 *Snippet 9-10: Sequence Group Mapping*



2097

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

2101

2102 Example:

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

2113 maps to:

```
struct myType {
  wchar_t *name;
  /* this points to a dynamically allocated string */
  size_t idList_num;
  long *idList;
  /* this points to a dynamically allocated array of longs */
  double *value;
  /* this points to a dynamically allocated long */
} *pmyVariable, myVariable;
```

2123 [Snippet 9-11: All Group Mapping](#)

2124

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

2129

2130 Example:

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

2132 maps to:

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

2134 [Snippet 9-12: Nillable Mapping](#)

2135

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

### 2137 **8.3.2-29.3.2.2 C to WSDL Mapping Details**

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

2139

2140 Example:

```
char *myFunction(struct DataStruct data, int id);
with the DataStruct type defined as a struct holding mappable types:
```

```
struct DataStruct {
  char *name;
```

2144

```
2145     double value;
2146     };
```

2147 maps to:

```
2148 <xsd:element name="myFunction">
2149 <xsd:complexType>
2150 <xsd:sequence>
2151 <xsd:element name="data" type="DataStruct" />
2152 <xsd:element name="id" type="xsd:int" />
2153 </xsd:sequence>
2154 </xsd:complexType>
2155 </xsd:element>
2156
2157 <xsd:complexType name="DataStruct">
2158   <xsd:sequence>
2159     <xsd:element name="name" type="xsd:string" />
2160     <xsd:element name="value" type="xsd:double" />
2161   </xsd:sequence>
2162 </xsd:complexType>
```

2163 *Snippet 9-13: Struct Mapping*

2164

- 2165 • char and wchar\_t arrays inside of structs are mapped to a restricted subtype of xsd:string that  
2166 limits the length the space allowed in the array.

2167

2168 Example:

```
2169 struct DataStruct {
2170     char name[256];
2171     double value;
2172 };
```

2173 maps to:

```
2174 <xsd:element name="myFunction">
2175 <xsd:complexType>
2176 <xsd:sequence>
2177 <xsd:element name="data" type="DataStruct" />
2178 <xsd:element name="id" type="xsd:int" />
2179 </xsd:sequence>
2180 </xsd:complexType>
2181 </xsd:element>
2182
2183 <xsd:complexType name="DataStruct">
2184   <xsd:sequence>
2185     <xsd:element name="name">
2186       <xsd:simpleType>
2187         <xsd:restriction base="xsd:string">
2188           <xsd:maxLength value="255" />
2189         </xsd:restriction>
2190       </xsd:simpleType>
2191     </xsd:element>
2192     <xsd:element name="value" type="xsd:double" />
2193   </xsd:sequence>
2194 </xsd:complexType>
```

2195 *Snippet 9-14: Character Array in Struct Mapping*

2196

- 2197 • C enums define a list of named symbols that map to values. If a function uses an enum type, this is  
2198 mapped to a restricted element in the WSDL schema.

2199

2200 Example:

2201 ~~char \*getValueFromType(enum ParameterType type);~~  
2202 with the ParameterType type defined as an enum:

```
2203 enum ParameterType {
2204     UNSET = 1,
2205     TYPEA,
2206     TYPEB,
2207     TYPEC
2208 };
```

2209 maps to:

```
2210 <del><xsd:element name="getValueFromType">
2211 <xsd:complexType>
2212 <xsd:sequence>
2213 <xsd:element name="type" type="ParameterType"/>
2214 </xsd:sequence>
2215 </xsd:complexType>
2216 </del></xsd:element>
2217
2218 <xsd:simpleType name="ParameterType">
2219 <xsd:restriction base="xsd:int">
2220 <xs:minInclusive value="1"/>
2221 <xs:maxInclusive value="4"/>
2222 </xsd:restriction>
2223 </xsd:simpleType>
```

2224 [Snippet 9-15: Enum Mapping](#)

2225

2226 The restriction used will have to be appropriate to the values of the enum elements.

2227

2228 Example:

```
2229 enum ParameterType {
2230     UNSET = 'u',
2231     TYPEA = 'A',
2232     TYPEB = 'B',
2233     TYPEC = 'C'
2234 };
```

2235 maps to:

```
2236 <xsd:simpleType name="ParameterType">
2237 <xsd:restriction base="xsd:int">
2238 <xsd:enumeration value="86"/> <!-- Character 'u' -->
2239 <xsd:enumeration value="65"/> <!-- Character 'A' -->
2240 <xsd:enumeration value="66"/> <!-- Character 'B' -->
2241 <xsd:enumeration value="67"/> <!-- Character 'C' -->
2242 </xsd:restriction>
2243 </xsd:simpleType>
```

2244 [Snippet 9-16: Non-contiguous Value Enum Mapping](#)

2245

- 2246 • If a struct or enum contains other structs or enums, the mapping rules are applied recursively.

2247

2248 Example:

2249 ~~char \*myFunction(struct DataStruct data);~~

2250 with types defined as follows:

```
2251 struct DataStruct {
2252     char name[30];
2253     double values[20];
2254     ParameterType type;
2255 };
2256
2257 enum ParameterType {
2258     UNSET = 1,
2259     TYPEA,
2260     TYPEB,
2261     TYPEC
2262 };
```

2263 maps to:

```
2264 <xsd:element name="myFunction">
2265 <xsd:complexType>
2266 <xsd:sequence>
2267 <xsd:element name="data" type="DataStruct"/>
2268 </xsd:sequence>
2269 </xsd:complexType>
2270 </xsd:element>
2271
2272 <xsd:complexType name="DataStruct">
2273     <xsd:sequence>
2274         <xsd:element name="name">
2275             <xsd:simpleType>
2276                 <xsd:restriction base="xsd:string">
2277                     <xsd:maxLength value="29"/>
2278                 </xsd:restriction>
2279             </xsd:simpleType>
2280         </xsd:element>
2281         <xsd:element name="values" type="xsd:double" minOccurs=20 maxOccurs=20/>
2282         <xsd:element name="type" type=" ParameterType"/>
2283     </xsd:sequence>
2284 </xsd:complexType>
2285
2286 <xsd:simpleType name="ParameterType">
2287     <xsd:restriction base="xsd:int">
2288         <xs:minInclusive value="1"/>
2289         <xs:maxInclusive value="4"/>
2290     </xsd:restriction>
2291 </xsd:simpleType>
2292
```

2293 *Handling Snippet 9-17: Nested Struct Mapping*

2294

- 2295 • ~~Mapping of C unions is not defined supported by this mapping, and is implementation dependent.~~  
2296 ~~For portability, unions are discouraged in service interfaces specification.~~
- 2297 • Typedefs are resolved when evaluating parameter and return types. Typedefs are resolved before  
2298 the mapping to Schema is done.
- 2299 • ~~Handling for pre-processor directives is not defined by this mapping, and support is implementation~~  
2300 ~~dependent. For portability pre-processor directives are discouraged in service interfaces.~~

2301

## 910 Conformance

2302

The XML schema pointed to by the RDDL document at the SCA namespace URI, defined by the Assembly specification. An SCA implementation MUST reject a composite file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> or <http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd>. [ASSEMBLY]-[C110001]

2306

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

2309

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

2312

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

2314

An SCA implementation MUST reject a componentType or constraining type SCA implementation MUST reject a composite file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> or <http://docs.oasis-open.org/opencsa/sca/200912/sca-implementation-c-1.1.xsd>. [C110002]

2318

[C110001]

2320

An SCA implementation MUST reject a contribution SCA implementation MUST reject a componentType file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200903/sca-contribution-interface-c-1.1.xsd>. [C110003]

2323

[C110002]

2325

An SCA implementation MUST reject a WSDL SCA implementation MUST reject a contribution file that does not conform to [http://docs.oasis-open.org/opencsa/sca-c-~~cpp~~/c/200901/200912/sca-wsdlext-contribution-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca-c-<del>cpp</del>/c/200901/200912/sca-wsdlext-contribution-c-1.1.xsd). [C110003]

2328

An SCA implementation MUST reject a WSDL file that does not conform to [http://docs.oasis-open.org/opencsa/sca-c-~~cpp~~/c/200901/sca-wsdlext-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca-c-<del>cpp</del>/c/200901/sca-wsdlext-c-1.1.xsd). [C110004]

2329

### 9.410.1 Conformance Targets

2330

The conformance targets of this specification are:

2332

- **SCA implementations**, which provide a **runtime** for SCA components and potentially **tools** for authoring SCA artifacts, component descriptions and/or runtime operations.

2333

- **SCA documents**, which describe SCA artifacts, and specific **elements** within these documents.

2334

- **C files**, which define SCA service interfaces and implementations.

2335

- **WSDL files**, which define SCA service interfaces.

2336

### 9.210.2 SCA Implementations

2337

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

2339

1. It MUST conform to the SCA Assembly Model Specification [ASSEMBLY] and the SCA Policy Framework [POLICY].

2341

2. It MUST comply with all statements in ~~Conformance Items~~Table F-1 and Table F-5 related to an SCA implementation, notably all mandatory statements have to be implemented.

2342

3. It MUST implement the SCA C API defined in section ~~C-API~~SCA Programming Interface.

2343

- 2344 4. It MAY support program-based component implementations. If program-based component  
2345 implementations are supported, the implementation MUST implement the Program-Based  
2346 Implementation Support API defined in Program-Based Implementation Support and MUST comply  
2347 with all statements in Table F-2 related to an SCA implementation, notably all mandatory statements  
2348 in that section have to be implemented.
- 2349 4.5. It MUST implement the mapping between C and WSDL 1.1 [WSDL11] defined in WSDL to C and C  
2350 to WSDL Mapping.
- 2351 5.6. It MUST support <interface.c/> and <implementation.c/> elements as defined in Component Type  
2352 and Component in composite, ~~and~~ componentType and ~~constrainingType~~ documents.
- 2353 6.7. It MUST support <export.c/> and <import.c/> elements as defined in C Contributions in contribution  
2354 documents.
- 2355 7.8. It MAY support source file annotations as defined in ~~C~~ SCA Annotations, C SCA Policy Annotations  
2356 and ~~C~~ WSDL Annotations. If source file annotations are supported, the implementation MUST  
2357 comply with all statements in Table F-3 related to an SCA implementation, notably all mandatory  
2358 statements in that section have to be implemented.
- 2359 8.9. It MAY support WSDL extentsions as defined in C WSDL Mapping Extensions. If WSDL extentsions  
2360 are supported, the implementation MUST comply with all statements in Table F-4 related to an SCA  
2361 implementation, notably all mandatory statements in that section have to be implemented.

### 2362 9.310.3 SCA Documents

2363 An SCA document conforms to this specification if it meets ~~the following~~these conditions:

- 2364 1. It MUST conform to the SCA Assembly Model Specification [ASSEMBLY] and, if appropriate, the  
2365 SCA Policy Framework [POLICY].

2366 4.0.2. If it is a composite document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-c-1.1.xsd)  
2367 [open.org/opencsa/sca/200903200912/sca-interface-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-c-1.1.xsd) and [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-implementation-c-1.1.xsd)  
2368 [open.org/opencsa/sca/200903200912/sca-implementation-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-implementation-c-1.1.xsd) schema and MUST comply with  
2369 the additional constraints on the document contents as defined in ~~Conformance Items~~Table F-1.

2370

2371 If it is a componentType ~~or constrainingType~~ document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-c-1.1.xsd)  
2372 [open.org/opencsa/sca/200903200912/sca-interface-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-interface-c-1.1.xsd) schema and MUST comply with the  
2373 additional constraints on the document contents as defined in ~~Conformance Items~~Table F-1.

2374

2375 If it is a contribution document, it MUST conform to the [http://docs.oasis-](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contribution-c-1.1.xsd)  
2376 [open.org/opencsa/sca/200903200912/sca-contribution-c-1.1.xsd](http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contribution-c-1.1.xsd) schema and MUST comply with the  
2377 additional constraints on the document contents as defined in ~~Conformance Items~~Table F-1.

### 2378 9.410.4 C Files

2379 A C file conforms to this specification if it meets the ~~following~~ conditions:

- 2380 1. It MUST comply with all statements in ~~Conformance Items~~Table F-1, Table F-3 and Table F-5 related  
2381 to C contents and annotations, notably all mandatory statements have to be satisfied.

### 2382 9.510.5 WSDL Files

2383 A WSDL conforms to this specification if it meets ~~the following~~these conditions:

- 2384 1. It is a valid WSDL 1.1 [WSDL11] document.
- 2385 2. It MUST comply with all statements in ~~Conformance Items~~Table F-1, Table F-4 and Table F-5 related  
2386 to WSDL contents and extensions, notably all mandatory statements have to be satisfied.

2387

## A C SCA Annotations

2388 To allow developers to define SCA related information directly in source files, without having to separately  
2389 author SCDL files, a set of annotations is defined. If SCA annotations are supported by an  
2390 implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as  
2391 described. The SCA runtime MUST only process the SCDL files and not the annotations. [CA0001]

2392

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

2393 In general an annotation immediately precedes the program element it applies to. -If multiple annotations  
2394 apply to a program element, all of the annotations SHOULD be in the same comment block. [CA0002]

2395

- Function or Function Prototype

2396

The annotation immediately precedes the function definition or declaration.

2397

Example:

2398

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

2400

*[Snippet A-1: Example Function Annotation](#)*

2401

- Variable

2402

The annotation immediately precedes the variable definition.

2403

Example:

2404

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

2405

2406

*[Snippet A-2: Example Variable Annotation](#)*

2407

- Set of Functions Implementing a Service

2408

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

2409

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

2410

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

2411

Example:

2412

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

2413

```
/* @ComponentType
```

2414

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

2415

*[Snippet A-3: Example Set of Functions Annotation](#)*

2416

- Set of Function Prototypes Defining an Interface

2417

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

2418

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

2419

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

2420

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

2421

```
/* @Remoteable
```

2422

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

2423

*[Snippet A-4: Example Set of Function Declarations Annotation](#)*

2424

### A.2 Interface Header Annotations

2425

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



## 2426 A.2.1 @Interface

2427 Annotation that indicates the start of a new interface definition. ~~An SCA implementation MUST treat a file~~  
2428 ~~with a @WebService annotation specified as if @Interface was specified with the name value of the~~  
2429 ~~@WebService annotation used as the name value of the @Interface annotation. [CA0003]~~

2430

2431

2432 **Corresponds to:** *interface.c* element

2433

2434 **Format:**

```
2435 /* @Interface(name="serviceName") */
```

2436 *Snippet A-5: @Interface Annotation Format*

2437 where

- 2438 • **name : NCName (10..1)** – specifies the name of ~~the a~~ service using this interface. The default is the  
2439 root name of the header file containing the annotation.

2440

2441 **Applies to:** Set of functions defining an interface.

2442 Function declarations following this annotation form the definition of this interface. -This annotation also  
2443 serves to bound the scope of the remaining annotations in this section,

2444

2445 Example:

2446 Interface header:

```
2447 /* @Interface(name="LoanService") */
```

2448

2449 Service definition:

```
2450 <service name="LoanService">  
2451 <interface.c header="loans.h" />  
2452 </service>
```

2453 *Snippet A-6: Example of @Interface Annotation*

## 2454 A.2.2 @Function

2455 Annotation that indicates that a function defines an operation of a service. There are two formats for this  
2456 annotation depending on if the service is implemented as a set of subroutines or in a program. An SCA  
2457 implementation MUST treat a function with a @WebFunction annotation specified as if @Function was  
2458 specified with the operationName value of the @WebFunction annotation used as the name value of the  
2459 @Function annotation and the exclude value of the @WebFunction annotation used as the exclude value  
2460 of the @Function annotation. [CA0004]

2461 **Corresponds to:** function or callbackFunction child element of an interface.c element. If the file the  
2462 function is contained in is being processed because it was identified via either  
2463 interface.c/@callbackHeader or a @Callback annotation, then the @Function annotation corresponds to  
2464 a callbackFunction element, otherwise it corresponds to a function element.

2465 **Format:**

```
2466 /* @Function(name="operationName", exclude="true") */
```

2467 *Snippet A-7: @Operation Annotation Format for Functions*

2468 where



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

**Applies to:** Function declaration

**Example:**

**Interface header (loans.h):**

```
short internalFcn(char *param1, short param2);

/* @Function(name="getRate") */
void rateFcn(char *cust, float *rate);
```

**Interface definition:**

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

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

### **A.2.2A.2.3 @Operation**

Annotation that indicates that a functionstruct declaration defines a request message format of an operation of a service. There are two formats for this annotation depending on if the service is implemented as a set of subroutines or in a program. An SCA implementation MUST treat a functionstruct with a @WebFunctionWebOperation annotation specified, unless the exclude value of the @WebFunction annotation is true, as if @Operation was specified with the operationName value of the @WebFunctionWebOperation 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. [CA0004] An SCA implementation MUST treat a struct with a @WebOperation annotation specified, unless the exclude value of the @WebOperation annotation is true, as if @Operation was specified with the struct as the input value, the operationName value of the @WebOperation annotation used as the name value of the @Operation annotation and the response value of the @WebOperation annotation used as the output values of the @Operation annotation. [CA0005]

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

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

**Format:**

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

where

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

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

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

2517  
2518  
2519  
2520  
2521  
2522  
2523  
2524  
2525  
2526  
2527  
2528  
2529  
2530  
2531  
2532  
2533  
2534  
2535  
2536  
2537  
2538  
2539  
2540  
2541  
2542  
2543  
2544  
2545  
2546  
2547  
2548  
2549  
2550  
2551  
2552  
2553  
2554  
2555  
2556  
2557  
2558  
2559  
2560  
2561  
2562  
2563  
2564  
2565

**Example:**

**Interface header (loans.h):**

```
short internalFen(char *param1, short param2);  
  
/* @Operation(name="getRate") */  
void rateFen(char *cust, float *rate);
```

**Interface definition:**

```
<interface.c header="loans.h">  
  <operation name="getRate" />  
</interface.c>
```

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

[CA0005]

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

**Format:**

```
/* @Operation(name="operationName", input="inputStruct", output="outputStruct")  
*/response="outStruct", exclude="true") */
```

Snippet A-9: @Operation Annotation Format for Structs

where

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

**Applies to:** struct declarations

**Example:**

**Interface header (loans.h):**

```
/* @Operation(name="getRate", input="rateInput", outputresponse="rateOutput")  
*/  
struct rateInput {  
    char cust[25];  
    int term;  
};  
struct rateOutput {  
    float rate;  
    int rateClass;  
};
```

2566 Interface definition:

```
2567 <interface.c header="loans.h">  
2568   <operationfunction name="getRate" input="rateInput" output="rateOutput" />  
2569 </interface.c>
```

2570 [Snippet A-10: Example of @Operation Annotation for Structs](#)

## 2571 **A.2.3A.2.4 @Remotable**

2572 Annotation on service interface to indicate that a service is remotable– and implies an @Interface  
2573 annotation applies as well. An SCA implementation MUST treat a file with a @WebService annotation  
2574 specified as if @Remotable and @Interface were specified with the name value of the @WebService  
2575 annotation used as the name value of the @Interface annotation. [CA0003]

2576

2577 **Corresponds to:** @remotable="true" attribute of an *interface.c* element.

2578

2579 **Format:**

```
2580 /* @Remotable */
```

2581 [Snippet A-11: @Remotable Annotation Format](#)

2582 The default is **false** (not remotable).

2583

2584 **Applies to:** Interface

2585

2586 **Example:**

2587 Interface header (LoanService.h):

```
2588 /* @Remotable */
```

2589

2590 Service definition:

```
2591 <service name="LoanService">  
2592   <interface.c header="LoanService.h" remotable="true" />  
2593 </service>
```

2594 [Snippet A-12: Example of @Remotable Annotation](#)

## 2595 **A.2.4A.2.5 @Callback**

2596 Annotation on a service interface to specify the callback interface.

2597

2598 **Corresponds to:** @callbackHeader attribute of an *interface.c* element.

2599

2600 **Format:**

```
2601 /* @Callback(header="headerName") */
```

2602 [Snippet A-13: @Callback Annotation Format](#)

2603 where

2604 • **header : Name (1..1)** – specifies the name of the header defining the callback service interface.

2605

2606 **Applies to:** Interface

2607

2608 Example:

2609 Interface header (MyService.h):

```
2610 /* @Callback(header="MyServiceCallback.h") */
```

2611

2612 Service definition:

```
2613 <service name="MyService">
2614   <interface.c header="MyService.h" callbackHeader="MyServiceCallback.h" />
2615 </service>
```

2616 [Snippet A-14: Example of @Callback Annotation](#)

## 2617 **A.2.5A.2.6 @OneWay**

2618 Annotation on a service interface function declaration to indicate the function is one way. The @OneWay  
2619 annotation also affects the representation of a service in WSDL. See @OneWay.

2620

2621

2622 **Corresponds to:** @oneWay="true" attribute of function element of an *interface.c* element.

2623

2624 **Format:**

```
2625 /* @OneWay */
```

2626 [Snippet A-15: @OneWay Annotation Format](#)

2627 The default is **false** (not OneWay).

2628

2629 **Applies to:** Function [Prototypedeclaration](#)

2630

2631 Example:

2632 Interface header:

```
2633 /* @OneWay */
2634 reportEvent(int eventID);
```

2635

2636 Service definition:

```
2637 <service name="LoanService">
2638   <interface.c header="LoanService.h">
2639     <function name="reportEvent" oneWay="true" />
2640   </interface.c>
2641 </service>
```

2642 [Snippet A-16: Example of @OneWay Annotation](#)

## 2643 **A.3 Implementation Annotations**

2644 This section lists the annotations that can be used in the file that implements a service.

### 2645 **A.3.1 @ComponentType**

2646 Annotation used to indicate the start of a new componentType.

2647

2648 **Corresponds to:** @componentType attribute of an *implementation.c* element.

2649

2650 **Format:**

```
2651 /* @ComponentType */
```

2652

2653 [Snippet A-17: @ComponetType Annotation Format](#)

2654 **Applies to:** Set of services, references and properties

2655

2656 **Example:**

2657 Implementation:

```
2658 /* @ComponentType */
```

2659

2660 Component definition:

```
2661 <component name="LoanService">
2662   <implementation.c module="loan" componentType="LoanService" />
2663 </component>
```

2664 [Snippet A-18: Example of @ComponentType Annotation](#)

### 2665 **A.3.2 @Service**

2666 Annotation that indicates the start of a new service implementation.

2667

2668 **Corresponds to:** *implementation.c* element

2669

2670 **Format:**

```
2671 /* @Service(name="serviceName", interfaceHeader="headerFile") */
```

2672 [Snippet A-19: @Service Annotation Format](#)

2673 where

- 2674 • **name** : *NCName* (**10..1**) – specifies the name of the service. [The default is the service name for the interface.](#)
- 2675 • **interfaceHeader** : *Name* (**1..1**) – specifies the C header defining the interface.

2677

2678 **Applies to:** Set of functions implementing a service

2679 Function definitions following this annotation form the implementation of this service. -This annotation also serves to bound the scope of the remaining annotations in this section,

2681

2682 **Example:**

2683 Implementation:

```
2684 /* @Service(name="LoanService", interfaceHeader="loan.h") */
```

2685

2686 ComponentType definition:

```
2687 <componentType name="LoanService">
2688   <service name="LoanService">
2689     <interface.c header="loans.h" />
2690   </service>
2691 </componentType>
```

2692 [Snippet A-20: Example of @Service Annotation](#)

2693 **A.3.3 @Reference**

2694 Annotation on a service implementation to indicate it depends on another service providing a specified  
2695 interface.

2696  
2697 **Corresponds to:** *reference* element of a *componentType* element.

2698  
2699 **Format:**

```
2700 /* @Reference(name="referenceName", interfaceHeader="headerFile",  
2701             required="true", multiple="true") */  
2702 */
```

2703 [Snippet A-21: @Reference Annotation Format](#)

2704 where

- 2705 • **name : NCName (1..1)** – specifies the name of the reference.
- 2706 • **interfaceHeader : Name (1..1)** – specifies the C header defining the interface.
- 2707 • **required : boolean (0..1)** – specifies whether a value has to be set for this reference. Default is **true**.
- 2708 • **multiple : boolean (0..1)** – specifies whether this reference can be wired to multiple services. Default  
2709 is **false**.

2710  
2711 The multiplicity of the reference is determined from the **required** and **multiple** attributes. If the value of  
2712 the **multiple** attribute is true, then component type has a reference with a multiplicity of either 0..n or 1..n  
2713 depending on the value of the **required** attribute – 1..n applies if **required=true**. Otherwise a multiplicity  
2714 of 0..1 or 1..1 is implied.

2715  
2716 **Applies to:** Service

2717  
2718 **Example:**

2719 Implementation:

```
2720 /* @Reference(name="getRate", interfaceHeader="rates.h") */  
2721 */  
2722 */  
2723 * @Reference(name="publishRate", interfaceHeader="myRates.h",  
2724             required="false", multiple="yes")  
2725 */
```

2726  
2727 **ComponentType definition:**

```
2728 <componentType name="LoanService">  
2729   <reference name="getRate">  
2730     <interface.c header="rates.h">  
2731   </reference>  
2732   <reference name="publishRate" multiplicity="0..n">  
2733     <interface.c header="myRates.h">  
2734   </reference>  
2735 </componentType>
```

2736 [Snippet A-22: Example of @Reference Annotation](#)

2737 **A.3.4 @Property**

2738 Annotation on a service implementation to define a property of the service. Should **immediately**  
2739 **precede** ~~the~~ **immediately precede** the **global** variable that the property is based on. ~~The~~ variable declaration

2740 is only used for determining the type of the property. -The variable will not be populated with the property  
2741 value at runtime. -Programs use the SCAProperty<Type>() functions for accessing property data.

2742

2743 **Corresponds to:** *property* element of a *componentType* element.

2744

2745 **Format:**

```
2746 /* @Property(name="propertyName", type="typeName",  
2747             default="defaultValue", required="true") */  
2748 */
```

2749 *Snippet A-23: @Property Annotation Format*

2750 where

- 2751 • **name : NCName (0..1)** – specifies the name of the property. If name is not specified the property  
2752 name is taken from the name of the **global**-variable.
- 2753 • **type : QName (0..1)** – specifies the type of the property. If not specified the type of the property is  
2754 based on the C mapping of the type of the following global variable to an xsd type as defined in Data  
2755 Binding. -If the variable is an array, then the property is many-valued.
- 2756 • **required : boolean (0..1)** – specifies whether a value has to be set in the component definition for  
2757 this property. Default is **false**.
- 2758 • **default : <type> (0..1)** – specifies a default value and is only needed if **required** is **false**.

2759

2760 **Applies to:** Variable

2761

2762 **An SCA implementation MUST ensure that all variables in a component implementation with the same**  
2763 **name and annotated with @Property have the same type. [CA0007]**

2764 Example:

2765 Implementation:

```
2766 /* @Property */  
2767 long loanType;
```

2768

2769 ComponentType definition:

```
2770 <componentType name="LoanService">  
2771     <property name="loanType" type="xsd:int" />  
2772 </componentType>
```

## 2773 **A.1.1 @Scope**

2774 ~~Annotation on a service implementation to indicate the scope of the service.~~

2775

2776 ~~**Corresponds to:** @scope attribute of an *implementation.c* element.~~

2777

2778 **Format:**

```
2779 /* @Scope("value") */
```

2780 where

- 2781 • ~~**value : [stateless | composite] (1..1)** – specifies the scope of the implementation. The default value  
2782 is **stateless**.~~

2783

2784 **Applies to:** Service

2785  
2786  
2787  
2788  
2789  
2790  
2791  
2792  
2793  
2794  
2795

**Example:**

**Implementation:**

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

**Component definition:**

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

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

2796 **A.3.5 @Init**

2797 Annotation on a service implementation to indicate a function to be called when the service is  
2798 instantiated. -If the service is implemented in a program, this annotation indicates the program is to be  
2799 called with an initialization flag prior to the first operation.

2800  
2801

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

2802  
2803  
2804

**Format:**

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

2805  
2806 *Snippet A-25: @Init Annotation Format*

2807 The default is **false** (the function is not to be called on service initialization).

2808  
2809

**Applies to:** Function or Service

2810  
2811

**Example:**

**Implementation:**

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

**Component definition:**

```
<component name="LoanService">  
  <implementation.c module="loan" componentType="LoanService">  
    <function name="init" init="true" />  
  </implementation.c>  
</component>
```

2812  
2813  
2814  
2815  
2816  
2817  
2818  
2819  
2820  
2821  
2822 *Snippet A-26: Example of @Init Annotation*

2823 **A.3.6 @Destroy**

2824 Annotation on a service implementation to indicate a function to be called when the service is terminated.  
2825 If the service is implemented in a program, this annotation indicates the program is to be called with a  
2826 termination flag after to the final operation.

2827  
2828  
2829

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



2830

2831 **Format:**

```
2832 /* @Destroy */
```

2833 [Snippet A-27: @Destroy Annotation Format](#)

2834 The default is **false** (the function is not to be called on service termination).

2835

2836 **Applies to:** Function or Service

2837

2838 **Example:**

2839 Implementation:

```
2840 /* @Destroy */  
2841 void cleanup();
```

2842

2843 Component definition:

```
2844 <component name="LoanService">  
2845   <implementation.c module="loan" componentType="LoanService">  
2846     <function name="cleanup" destroy="true" />  
2847   </implementation.c>  
2848 </component>
```

2849 [Snippet A-28: Example of @Destroy Annotation](#)

### 2850 **A.3.7 @EagerInit**

2851 Annotation on a service implementation to indicate the service is to be instantiated when its containing  
2852 component is started.

2853

2854 **Corresponds to:** `@eagerInit="true"` attribute of an *implementation.c* element.

2855

2856 **Format:**

```
2857 /* @EagerInit */
```

2858 [Snippet A-29: @EagerInit Annotation Format](#)

2859 The default is **false** (the service is initialized lazily).

2860

2861 **Applies to:** Service

2862

2863 **Example:**

2864 Implementation:

```
2865 /* @EagerInit */
```

2866

2867 Component definition:

```
2868 <component name="LoanService">  
2869   <implementation.c module="loan" componentType="LoanService"  
2870     eagerInit="true" />  
2871 </component>
```

2872 [Snippet A-30: Example of @EagerInit Annotation](#)

### 2873 **A.3.8 @AllowsPassByReference**

2874 Annotation on service implementation or operation to indicate that a service or operation allows pass by  
2875 reference semantics.

2876

2877 **Corresponds to:** `@allowsPassByReference="true"` attribute of an *implementation.c* element or a *function*  
2878 child element of an *implementation.c* element.

2879

2880 **Format:**

```
2881 /* @AllowsPassByReference */
```

2882 [Snippet A-31: @AllowsPassByReference Annotation Format](#)

2883 The default is **false** (the service does not allow by reference parameters).

2884

2885 **Applies to:** Service or Function

2886

2887 **Example:**

2888 Implementation:

```
2889 /* @Service(name="LoanService")  
2890 * @AllowsPassByReference  
2891 */
```

2892

2893 Component definition:

```
2894 <component name="LoanService">  
2895   <implementation.c module="loan" componentType="LoanService"  
2896     allowsPassByReference="true" />  
2897 </component>
```

2898 [Snippet A-32: Example of @AllowsPassByReference Annotation](#)

## 2899 **A.4 Base Annotation Grammar**

2900 ~~While annotations are defined using the `/* ... */` format for comments, if the `// ...` format is supported by a  
2901 C compiler, the `// ...` format MAY be supported by an annotation processor.~~

2902 While annotations are defined using the `/* ... */` format for comments, if the `// ...` format is supported by a  
2903 C compiler, the `// ...` format MAY be supported by an SCA implementation annotation processor.  
2904 [CA0006]

2905

```
2906 <annotation> ::= /* @<baseAnnotation> */  
2907  
2908 <baseAnnotation> ::= <name> [( <params> )]  
2909  
2910 <params> ::= <paramNameValue>[ , <paramNameValue> ]* |  
2911   <paramValue>[ , <paramValue> ]*  
2912  
2913 <paramNameValue> ::= <name>=" <value> "  
2914  
2915 <paramValue> ::= " <value> "  
2916  
2917 <name> ::= NCName  
2918  
2919 <value> ::= string
```

2920 [Snippet A-33: Base Annotation Grammar](#)

- 2921 • Adjacent string constants are concatenated
- 2922 • NCName is as defined by XML schema **[XSD]**
- 2923 • Whitespace including newlines between tokens is ignored.
- 2924 • Annotations with parameters can span multiple lines within a comment, and are considered complete
- 2925 when the terminating “)” is reached.

---

## 2926 B C SCA Policy Annotations

2927 SCA provides facilities for the attachment of policy-related metadata to SCA assemblies, which influence  
2928 how implementations, services and references behave at runtime. -The policy facilities are described in  
2929 **[POLICY]**. -In particular, the facilities include Intents and Policy Sets, where intents express abstract,  
2930 high-level policy requirements and policy sets express low-level detailed concrete policies.

2931  
2932 Policy metadata can be added to SCA assemblies through the means of declarative statements placed  
2933 into Composite documents and into Component Type documents. -These annotations are completely  
2934 independent of implementation code, allowing policy to be applied during the assembly and deployment  
2935 phases of application development.

2936  
2937 However, it can be useful and more natural to attach policy metadata directly to the code of  
2938 implementations. -This is particularly important where the policies concerned are relied on by the code  
2939 itself. -An example of this from the Security domain is where the implementation code expects to run  
2940 under a specific security Role and where any service operations invoked on the implementation have to  
2941 be authorized to ensure that the client has the correct rights to use the operations concerned. -By  
2942 annotating the code with appropriate policy metadata, the developer can rest assured that this metadata  
2943 is not lost or forgotten during the assembly and deployment phases.

2944  
2945 The SCA C policy annotations provide the capability for the developer to attach policy information to C  
2946 implementation code. -The annotations ~~concerned first~~ provide both general facilities for attaching SCA  
2947 Intents and Policy Sets to C code. ~~Secondly, there are further specific and~~ annotations that deal with  
2948 particular for specific policy intents ~~for certain policy domains such as Security. Policy annotation can be~~  
2949 used in files for service interfaces or component implementations.

### 2950 B.1 General Intent Annotations

2951 SCA provides the annotation **@Requires** for the attachment of any intent to a C function, to a C function  
2952 declaration or to sets of functions implementing a service or sets of function declarations defining a  
2953 service interface.

2954  
2955 The @Requires annotation can attach one or multiple intents in a single statement.

2956  
2957 Each intent is expressed as a string. -Intents are XML QNames, which consist of a Namespace URI  
2958 followed by the name of the Intent. The precise form used is ~~as follows:~~

2959  
2960 

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

2961 *Snippet B-1: Intent Format*

2962  
2963 Intents can be qualified, in which case the string consists of the base intent name, followed by a ".",  
2964 followed by the name of the qualifier. -There can also be multiple levels of qualification.

2965  
2966 This representation is quite verbose, so we expect that reusable constants will be defined for the  
2967 namespace part of this string, as well as for each intent that is used by C code. -SCA defines constants  
2968 for intents such as the following:

2969

```

2970     /* @Define SCA_PREFIX "{http://docs.oasis-
2971     pen.org/ns/opencsa/sca/200903200912}" */
2972     /* @Define CONFIDENTIALITY SCA_PREFIX ## "confidentiality" */
2973     /* @Define CONFIDENTIALITY_MESSAGE CONFIDENTIALITY ## ".message" */

```

2974 [Snippet B-2: Example Intent Constants](#)

2975

2976 Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,  
 2977 separated by an underscore. -These intent constants are defined in the file that defines an annotation for  
 2978 the intent (annotations for intents, and the formal definition of these constants, are covered in a following  
 2979 section).

2980

2981 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.

2982

2983 **Corresponds to:** @requires attribute of [a-service](#), [reference](#), [operation](#), [an interface.c](#), [implementation.c](#),  
 2984 [function](#) or [propertycallbackFunction](#) element.

2985

2986 **Format:**

```

2987     /* @Requires("qualifiedIntent" | {"qualifiedIntent" [, "qualifiedIntent"]}) */

```

2988 where

```

2989     qualifiedIntent ::= QName | QName.qualifier | QName.qualifier1.qualifier2

```

2990

2991 [Snippet B-3: @Requires Annotation Format](#)

2992 **Applies to:** Interface, Service, Function, Function Prototype

2993

2994 Examples:

2995 Attaching the intents "confidentiality.message" and "integrity.message".

```

2996     /* @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE}) */

```

2997

2998 [Snippet B-4: Example @Requires Annotation](#)

2999 A reference requiring support for confidentiality:

```

3000     /* @Requires(CONFIDENTIALITY)
3001     * @Reference(interfaceHeader="SetBar.h") */
3002     void setBar(struct barType *bar);

```

3003 [Snippet B-5: @Requires Annotation applied with an @Reference Annotation](#)

3004

3005 Users can also choose to only use constants for the namespace part of the QName, so that they can add  
 3006 new intents without having to define new constants. -In that case, this definition would instead look like  
 3007 this:

3008

```

3009     /* @Requires(SCA_PREFIX "confidentiality")
3010     * @Reference(interfaceHeader="SetBar.h") */
3011     void setBar(struct barType *bar);

```

3012 [Snippet B-6: @Requires Annotation Using Mixed Constants and Literals](#)

3013 **B.2 Specific Intent Annotations**

3014 In addition to the general intent annotation supplied by the @Requires annotation described above, there  
3015 are C annotations that correspond to ~~some~~-specific policy intents.

3016  
3017 The general form of these specific intent annotations is an annotation with a name derived from the name  
3018 of the intent itself. -If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation  
3019 in the form of a string or an array of strings.

3020  
3021 For example, the SCA confidentiality intent described in General Intent Annotations using the  
3022 @Requires(CONFIDENTIALITY) intent can also be specified with the specific @Confidentiality intent  
3023 annotation. -The specific intent annotation for the "integrity" security intent is:

```
3024 /* @Integrity */
```

3026 *Snippet B-7: Example Specific Intent Annotation*

3027  
3028 **Corresponds to:** @requires="*Intent*" attribute of ~~a-service, reference, operationan interface.c,~~  
3029 ~~implementation.c, function~~ or ~~propertycallbackFunction~~ element.

3030  
3031 **Format:**

```
3032 /* @<Intent>[(qualifiers)] */
```

3033 where Intent is an NCName that denotes a particular type of intent.

```
3034 Intent ::= NCName  
3035 qualifiers ::= "qualifier" | {"qualifier" [, "qualifier"] }  
3036 qualifier ::= NCName | NCName/qualifier
```

3037  
3038 *Snippet B-8: @<Intent> Annotation Format*

3039 **Applies to:** Interface, Service, Function, Function Prototype – but see specific intents for restrictions

3040  
3041 **Example:**

```
3042 /* @ClientAuthentication( {"message", "transport"} ) */
```

3043 *Snippet B-9 Example @<Intent> Annotation*

3044  
3045 This annotation attaches the pair of qualified intents: *authentication.message* and *authentication.transport*  
3046 (the sca: namespace is assumed in both of these cases – "http:// docs.oasis-  
3047 open.org/ns/opencsa/sca/~~200903200912~~").

3048  
3049 The Policy Framework **[POLICY]** defines a number of intents and qualifiers. ~~The following~~  
3050 ~~sections~~[Security Interaction – Miscellaneous](#) define the annotations for those intents.

3051 **B.2.1 Security Interaction**

Intent	Annotation
<a href="#">authenticationclientAuthentication</a>	@ClientAuthentication
<a href="#">serverAuthentication</a>	@ServerAuthentication

<a href="#">mutualAuthentication</a>	<a href="#">@MutualAuthentication</a>
confidentiality	@Confidentiality
integrity	@Integrity

3052 [Table B-1: Security Interaction Intent Annotations](#)

3053

3054 These three intents can be qualified with

- 3055
- transport
  - 3056 • message

## 3057 B.2.2 Security Implementation

Intent	Annotation
<a href="#">runAs</a>	<a href="#">@RunAs(role"role")</a>
<a href="#">Allow</a>	<a href="#">@Allow(roles="&lt;comma-separated list of roles&gt;")</a>
<a href="#">permitAll</a>	<a href="#">@PermitAll</a>
<a href="#">denyAll</a>	<a href="#">@DenyAll</a>

3058

3059 ~~In addition to allow roles to defined, an SCA runtime MAY use the following annotation~~

3060 [@DeclareRoles\(<comma-separated list of roles>"\)](#)

Intent	Annotation	Qualifiers
<a href="#">authorization</a>	<a href="#">@Authorization</a>	<a href="#">fine_grain</a>

3061 [Table B-2: Security Implementation Intent Annotations](#)

## 3062 B.2.3 Reliable Messaging

Intent	Annotation
atLeastOnce	@AtLeastOnce
atMostOnce	@AtMostOnce
<del>Ordered</del> <a href="#">ordered</a>	@Ordered
exactlyOnce	@ExactlyOnce

3063

3064 [Table B-3: Reliable Messagng Intent Annotations](#)

## 3065 B.2.4 Transactions

Intent	Annotation	Qualifiers
managedTransaction	@ManagedTransaction	<del>Local</del> <a href="#">local</a> global

<u>noManagedTransaction</u>	<u>@NoManagedTransaction</u>	
transactedOneWay	@TransactedOneWay	
immediateOneWay	@ImmediateOneWay	
propagates Transaction	@PropagatesTransaction	
suspendsTransaction	@SuspendsTransaction	

3066

3067 *Table B-4: Transaction Intent Annotations*

## 3068 B.2.5 Miscellaneous

Intent	Annotation	Qualifiers
SOAP	@SOAP	<u>v1_1</u> <u>v1_2</u>
JMS	@JMS	

## 3069 ~~A.2 Application of Intent Annotations~~

3070 ~~Where multiple intent annotations (general or specific) are applied to the same C element, they are~~  
3071 ~~additive in effect. An example of multiple policy annotations being used together follows:~~

3072

3073 ~~/\* @Authentication~~

3074  ~~\* @Requires({CONFIDENTIALITY\_MESSAGE, INTEGRITY\_MESSAGE}) \*/~~

3075

3076 ~~In this case, the effective intents are authentication, confidentiality.message and integrity.message.~~

3077

3078 ~~If an annotation is specified at both the implementation/interface level and the function or variable level,~~  
3079 ~~then the function or variable level annotation completely overrides the implementation/interface level~~  
3080 ~~annotation of the same type.~~

3081

3082 ~~The intent annotation can be applied either to interface or to functions when adding annotated policy on~~  
3083 ~~SCA services.~~

## 3084 ~~A.3 Policy Annotation Scope~~

3085 ~~The following examples show scope of intents on functions, function declarations and sets of each.~~

3086

3087 ~~/\* @Remotable~~

3088  ~~\* @Integrity("transport")~~

3089  ~~\* @Authentication~~

3090  ~~\* @Service(name="HelloService", interfaceHeader="helloservice.h") \*/~~

3091

3092 ~~/\* @Integrity~~

3093  ~~\* @Authentication("message") \*/~~

3094  ~~wechar\_t\* hello(wechar\_t\* message) {...}~~

3095

3096 ~~/\* @Integrity~~

3097  ~~\* @Authentication("transport") \*/~~

3098  ~~wechar\_t\* helloThere() {...}~~



```

3099 /* @Remotable
3100 * @Confidentiality("message")
3101 * @Service(name="HelloHelperService", interfaceHeader="helloService.h") */
3102
3103
3104 /* @Confidentiality("transport") */
3105 wechar_t* hello(wechar_t* message) {...}
3106
3107 /* @Authentication */
3108 wechar_t* helloWorld(){...}

```

3109 **Example 1a. Usage example of annotated policy and inheritance.**

3110

- 3111 ● ~~The effective intent annotation on the helloWorld function is @Authentication, and~~
- 3112 ~~@Confidentiality("message").~~
- 3113 ● ~~The effective intent annotation on the hello function of the HelloHelperService is~~
- 3114 ~~@Confidentiality("transport"),~~
- 3115 ● ~~The effective intent annotation on the helloThere function of the HelloService is @Integrity and~~
- 3116 ~~@Authentication("transport").~~
- 3117 ● ~~The effective intent annotation on the hello function of the HelloService is @Integrity and~~
- 3118 ~~@Authentication("message")~~

3119

3120 ~~The listing below contains the equivalent declarative security interaction policy of the HelloService and~~  
3121 ~~HelloHelperService implementation corresponding to the Java interfaces and classes shown in Example~~  
3122 ~~1a.~~

3123

```

3124 <?xml version="1.0" encoding="ASCII"?>
3125
3126 <composite xmlns="http://www.ocea.org/xmlns/oca/1.0"
3127 name="HelloServiceComposite">
3128 ...
3129
3130 <component name="HelloServiceComponent">
3131 <service name="HelloService" requires="integrity/transport
3132 authentication">
3133 ...
3134 </service>
3135 <implementation.e module="HelloService.dll"
3136 componentType="LoanService" header="HelloService.h">
3137 <function name="hello" requires="integrity
3138 authentication/message"/>
3139 <function name="helloThere" requires="integrity
3140 authentication/transport"/>
3141 </implementation.e>
3142 </component>
3143 <component name="HelloHelperServiceComponent">
3144 <service name="HelloHelperService" requires="integrity/transport
3145 authentication confidentiality/message">
3146 ...
3147 </service>
3148 <implementation.e module="HelloService.dll"
3149 componentType="LoanService" header="HelloService.h">
3150 <function name="hello" requires="confidentiality/transport"/>
3151 <function name="helloWorld" requires="authentication"/>
3152 </implementation.e>
3153 </component>
3154
3155 ...
3156

```

3157 `</composite>`

3158 ~~Example 1b. Declarative intents equivalent to annotated intents in Example 1a.~~

## 3159 ~~A.4 Relationship of Declarative And Annotated Intents~~

3160 ~~Annotated intents on a C functions or function declarations cannot be overridden by declarative intents~~  
3161 ~~either in a composite document which uses the functions as an implementation or by statements in a~~  
3162 ~~componentType document associated with the functions. This rule follows the general rule for intents that~~  
3163 ~~they represent fundamental requirements of an implementation.~~

3164

3165 ~~An unqualified version of an intent expressed through an annotation in the C function or function~~  
3166 ~~declaration can be qualified by a declarative intent in a using composite document.~~

3167 ~~[Table B-5: Miscellaneous Intent Annotations](#)~~

## 3168 B.3 Policy Set Annotations

3169 The SCA Policy Framework uses Policy Sets to capture detailed low-level concrete policies (for example,  
3170 a concrete policy is the specific encryption algorithm to use when encrypting messages when using a  
3171 specific communication protocol to link a reference to a service).

3172

3173 Policy Sets can be applied directly to C implementations using the **@PolicySets** annotation. -The  
3174 PolicySets annotation either takes the QName of a single policy set as a string or the name of two or  
3175 more policy sets as an array of strings.

3176

3177 **Corresponds to:** ~~@policySets~~ attribute of ~~a service, reference, operationan interface.c,~~  
3178 ~~implementation.c, function~~ or ~~propertycallbackFunction~~ element.

3179

3180 **Format:**

```
3181 /* @PolicySets( "<policy set QName>" |  
3182 * { "<policy set QName>" [, "<policy set QName>" ] }) */
```

3183 ~~[Snippet B-10: @PolicySets Annotation Format](#)~~

3184 As for intents, PolicySet names are QNames – in the form of "{Namespace-URI}localPart".

3185

3186 **Applies to:** Interface, Service, Function, Function Prototype

3187

3188 **Example:**

```
3189 /* @Reference(name="helloService", interfaceHeader="helloService.h",  
3190 *         required=true)  
3191 * @PolicySets({ MY_NS "WS_Encryption_Policy",  
3192 *         MY_NS "WS_Authentication_Policy" }) */  
3193 HelloService* helloService;  
3194 ...  
3195 }
```

3196 ~~[Snippet B-11: Example @PolicySets Annotation](#)~~

3197

3198 In this case, the Policy Sets WS\_Encryption\_Policy and WS\_Authentication\_Policy are applied, both  
3199 using the namespace defined for the constant MY\_NS.

3200

3201 PolicySets satisfy intents expressed for the implementation when both are present, according to the rules  
3202 defined in **[POLICY]**.

## 3203 B.4 Policy Annotation Grammar Additions

```
3204 <annotation> ::= /* @<baseAnnotation> | @<requiresAnnotation> |
3205                @<intentAnnotation> | @<policySetAnnotation> */
3206
3207 <requiresAnnotation> ::= Requires(<intents>)
3208
3209 <intents> ::= "<qualifiedIntent>" |
3210            {"<qualifiedIntent>"[, "<qualifiedIntent>"]*})
3211
3212 <qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |
3213                    <intentName>.<qualifier>.<qualifier>
3214
3215 <intentName> ::= {anyURI}NCName
3216
3217 <intentAnnotation> ::= <intent>[(<qualifiers>)]
3218
3219 <intent> ::= NCName[(param)]
3220
3221 <qualifiers> ::= "<qualifier>" | {"<qualifier>"[, "<qualifier>"]*}
3222
3223 <qualifier> ::= NCName | NCName/<qualifier>
3224
3225 <policySetAnnotation> ::= policySets(<policysets>)
3226
3227 <policySets> ::= "<policySetName>" | {"<policySetName>"[, "<policySetName>"]*}
3228
3229 <policySetName> ::= {anyURI}NCName
```

3230 [Snippet B-12: Annotation Grammar Additions for Policy Annotations](#)

- 3231 • anyURI is as defined by XML schema [XSD]

## 3232 B.5 Annotation Constants

```
3233 <annotationConstant> ::= /* @Define <identifier> <token string> */
3234
3235 <identifier> ::= token
3236
3237 <token string> ::= "string" | "string"[ ## <token string>]
```

3238 [Snippet B-13: Annotation Constants Grammar](#)

- 3239 • Constants are immediately expanded

## 3240 C C WSDL Annotations

3241 To allow developers to control the mapping of C to WSDL, a set of annotations is defined. If WSDL  
3242 mapping annotations are supported by an implementation, the annotations defined here MUST be  
3243 supported and MUST be mapped to WSDL as described. [CC0005]

### 3244 C.1 Interface Header Annotations

#### 3245 C.1.1 @WebService

3246 Annotation on a C header file indicating that it represents a web service. -A second or subsequent  
3247 instance of this annotation in a file, or a first instance after any function declarations indicates the start of  
3248 a new service and has to contain a name value. An SCA implementation MUST treat any instance of a  
3249 @InterfaceRemotable annotation and without an explicit @WebService annotation as if a @WebService  
3250 annotation with a name value equal to the name value of the @Interface annotation, if specified, and no  
3251 other parameters was specified. [CC0001]

3252

3253 **Corresponds to:** javax.jws.WebService annotation in the JAX-WS specification (7.11.1)

3254

3255 **Format:**

```
3256 /* @WebService(name="portTypeName", targetNamespace="namespaceURI",  
3257 *           serviceName="WSDLServiceName", portName="WSDLPortName") */
```

3258 *Snippet C-1: @WebService Annotation Format*

3259 where

- 3260 • **name : NCName (0..1)** – specifies the name of the web service portType. -The default is the root  
3261 name of the header file containing the annotation. The name of the associated binding is also  
3262 determined by the portType. The binding name is the name of the portType suffixed with “Binding”.
- 3263 • **targetNamespace : anyURI (0..1)** – specifies the target namespace for the web service. -The default  
3264 namespace is determined by the implementation.
- 3265 • **serviceName : NCName (0..1)** – specifies the name for the associated WSDL service. -The default  
3266 service name is the name of the header file containing the annotation suffixed with “Service”.~~The~~  
3267 ~~name of the associated binding is also determined by the serviceName. In the case of a SOAP~~  
3268 ~~binding, the binding name is the name of the service suffixed with “SoapBinding”.~~
- 3269 • **portName : NCName (0..1)** – specifies the name for the associated WSDL port for the service.~~If~~  
3270 portName is not specified, the name of the WSDL port is the name of the portType suffixed with  
3271 “Port”. See If a @WebService does not have a portName element, an SCA implementation MUST  
3272 use the value associated with the name element, suffixed with “Port”.[CF0032][GC0008]

3273

3274 **Applies to:** Header file

3275

3276 **Example:**

3277 Input C header file (stockQuote.h):

```
3278 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
3279 *           serviceName="StockQuoteService") */  
3280 ...  
3281
```

3282

3283 Generated WSDL file:

```
3284 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3285             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3286             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3287             xmlns:tns="http://www.example.org/"
3288             targetNamespace="http://www.example.org/">
3289
3290     <portType name="StockQuote">
3291         <sca-c:bindings>
3292             <sca-c:prefix name="stockQuote"/>
3293         </sca-c:bindings>
3294     </portType>
3295
3296     <binding name="StockQuoteServiceSoapBinding">
3297         <soap:binding style="document"
3298             transport="http://schemas.xmlsoap.org/soap/http"/>
3299     </binding>
3300
3301     <service name="StockQuoteService">
3302         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3303             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3304         </port>
3305     </service>
3306 </definitions>
```

3307 [Snippet C-2: Example @WebService Annotation](#)

## 3308 C.1.2 @WebFunction

3309 Annotation on a C function indicating that it represents a web service operation. An SCA implementation  
3310 MUST treat a function annotated with an [@OperationFunction](#) annotation and without an explicit  
3311 [@WebFunction](#) annotation as if a [@WebFunction](#) annotation with with an operationName value equal to  
3312 the name value of the [@OperationFunction](#) annotation, an exclude value equal to the exclude value of  
3313 the [@Function](#) annotation and no other parameters was specified. [CC0002]

3314

3315 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

3316

3317 **Format:**

```
3318 /* @WebFunction(operationName="operation", action="SOAPAction",
3319 *              exclude="false") */
```

3320 [Snippet C-3: @WebFunction Annotation Format](#)

3321 where:

- 3322 • **operationName : NCName (0..1)** – specifies the name of the WSDL operation to associate with this  
3323 function. -The default is the name of the C function the annotation is applied to omitting any preceding  
3324 namespace prefix and portType name.
- 3325 • **action : string (0..1)** – specifies the value associated with the soap:operation/@soapAction attribute  
3326 in the resulting code. -The default value is an empty string.
- 3327 • **exclude : boolean (0..1)** – specifies whether this function is included in the web service interface.  
3328 The default value is "false".

3329

3330 **Applies to:** Function.

3331

3332 Example:

3333 Input C header file:

```

3334 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3335 *           serviceName="StockQuoteService") */
3336
3337 /* @WebFunction(operationName="GetLastTradePrice",
3338 *           action="urn:GetLastTradePrice") */
3339 float getLastTradePrice(const char *tickerSymbol);
3340
3341 /* @WebFunction(exclude="true") */
3342 void setLastTradePrice(const char *tickerSymbol, float value);

```

3343

3344 Generated WSDL file:

```

3345 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3346             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3347             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3348             xmlns:tns="http://www.example.org/"
3349             targetNamespace="http://www.example.org/">
3350
3351     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3352               xmlns:tns="http://www.example.org/"
3353               attributeFormDefault="unqualified"
3354               elementFormDefault="unqualified"
3355               targetNamespace="http://www.example.org/">
3356         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3357         <xs:element name="GetLastTradePriceResponse"
3358                   type="tns:GetLastTradePriceResponse"/>
3359         <xs:complexType name="GetLastTradePrice">
3360             <xs:sequence>
3361                 <xs:element name="tickerSymbol" type="xs:string"/>
3362             </xs:sequence>
3363         </xs:complexType>
3364         <xs:complexType name="GetLastTradePriceResponse">
3365             <xs:sequence>
3366                 <xs:element name="return" type="xs:float"/>
3367             </xs:sequence>
3368         </xs:complexType>
3369     </xs:schema>
3370
3371     <-message name="GetLastTradePrice">
3372         <part name="parameters" element="tns:GetLastTradePrice">
3373             </part>
3374     </message>
3375
3376     <-message name="GetLastTradePriceResponse">
3377         <part name="parameters" element="tns:GetLastTradePriceResponse">
3378             </part>
3379     </-message>
3380
3381     <portType name="StockQuote">
3382         <sca-c:bindings>
3383             <sca-c:prefix name="stockQuote"/>
3384         </sca-c:bindings>
3385         <operation name="GetLastTradePrice">
3386             <sca-c:bindings>
3387                 <sca-c:function name="getLastTradePrice"/>
3388             </sca-c:bindings>
3389             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3390                 </input>
3391             <output name="GetLastTradePriceResponse"
3392                   message="tns:GetLastTradePriceResponse">
3393                 </output>
3394             </operation>
3395     </portType>
3396

```

```

3397     <binding name="StockQuoteServiceSoapBinding">
3398         <soap:binding style="document"
3399             transport="http://schemas.xmlsoap.org/soap/http"/>
3400         <wsdl:operation name="GetLastTradePrice">
3401             <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3402             <wsdl:input name="GetLastTradePrice">
3403                 <soap:body use="literal"/>
3404             </wsdl:input>
3405             <wsdl:output name="GetLastTradePriceResponse">
3406                 <soap:body use="literal"/>
3407             </wsdl:output>
3408         </wsdl:operation>
3409     </binding>
3410
3411     <service name="StockQuoteService">
3412         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3413             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3414         </port>
3415     </service>
3416 </definitions>

```

3417 [Snippet C-4: Example @WebFunction Annotation](#)

### 3418 C.1.3 @WebOperation

3419 Annotation on a C request message struct indicating that it represents a web service operation. An SCA  
3420 implementation MUST treat a struct annotated with an @Operation annotation without an explicit  
3421 @WebOperation annotation as if a @WebOperation annotation with with an operationName value equal  
3422 to the name value of the @Operation annotation, a response value equal to the **output response value of**  
3423 **the @Operation annotation, an exclude value equal to the exclude** value of the @Operation annotation  
3424 and no other parameters was specified **is applied to the struct identified as the input value of the**  
3425 **@Operation annotation.** [CC0003]

3426

3427 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

3428

3429 **Format:**

```

3430 /* @WebOperation(operationName="operation", response="responseStruct",
3431 *               action="SOAPAction", exclude="false") */

```

3432 [Snippet C-5: @WebOperation Annotation Format](#)

3433 where:

- 3434 • **operationName : NCName (0..1)** – specifies the name of the WSDL operation to associate with this  
3435 request message struct. -The default is the name of the C struct the annotation is applied to omitting  
3436 any preceding namespace prefix and portType name.
- 3437 • **response : NMTOKEN (0..1)** – specifies the name of the struct that defines the format of the  
3438 response message.
- 3439 • **action string : (0..1)** – specifies the value associated with the soap:operation/@soapAction attribute  
3440 in the resulting code. -The default value is an empty string.
- 3441 • **exclude binary : (0..1)** – specifies whether this struct is included in the web service interface. -The  
3442 default value is "false".

3443

3444 **Applies to:** Struct.

3445

3446 **Example:**

3447 Input C header file:



```

3448 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3449 *           serviceName="StockQuoteService") */
3450
3451 /* @WebOperation(operationName="GetLastTradePrice",
3452 *           response="getLastTradePriseResponseMsg"
3453 *           action="urn:GetLastTradePrice") */
3454 struct getLastTradePriceMsg {
3455     char tickerSymbol[10];
3456 } getLastTradePrice;
3457
3458 struct getLastTradePriceResponseMsg {
3459     float return;
3460 } getLastTradePriceResponse;

```

3461

#### Generated WSDL file:

```

3463 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3464             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3465             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3466             xmlns:tns="http://www.example.org/"
3467             targetNamespace="http://www.example.org/">
3468
3469     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3470               xmlns:tns="http://www.example.org/"
3471               attributeFormDefault="unqualified"
3472               elementFormDefault="unqualified"
3473               targetNamespace="http://www.example.org/">
3474         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3475         <xs:element name="GetLastTradePriceResponse"
3476                   type="tns:GetLastTradePriceResponse"/>
3477         <xs:simpleType name="TickerSymbolType">
3478           <xs:restriction base="xs:string">
3479             <xsd:maxLength value="9"/>
3480           </xs:restriction>
3481         </xs:simpleType>
3482         <xs:complexType name="GetLastTradePrice">
3483           <xs:sequence>
3484             <xs:element name="tickerSymbol" type="TickerSymbolType"/>
3485           </xs:sequence>
3486         </xs:complexType>
3487         <xs:complexType name="GetLastTradePriceResponse">
3488           <xs:sequence>
3489             <xs:element name="return" type="xs:float"/>
3490           </xs:sequence>
3491         </xs:complexType>
3492     </xs:schema>
3493
3494     <-message name="GetLastTradePrice">
3495       <sca-c:bindings>
3496         <sca-c:struct name="getLastTradePrice"/>
3497       </sca-c:bindings>
3498       <part name="parameters" element="tns:GetLastTradePrice">
3499       </part>
3500     </message>
3501
3502     <-message name="GetLastTradePriceResponse">
3503       <sca-c:bindings>
3504         <sca-c:struct name="getLastTradePriceResponse"/>
3505       </sca-c:bindings>
3506       <part name="parameters" element="tns:GetLastTradePriceResponse">
3507       </part>
3508     </message>
3509
3510     <portType name="StockQuote">

```



```

3511     <sca-c:bindings>
3512         <sca-c:prefix name="stockQuote" />
3513     </sca-c:bindings>
3514     <operation name="GetLastTradePrice">
3515         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3516             </input>
3517         <output name="GetLastTradePriceResponse"
3518             message="tns:GetLastTradePriceResponse">
3519             </output>
3520     </operation>
3521 </portType>
3522
3523 <binding name="StockQuoteServiceSoapBinding">
3524     <soap:binding style="document"
3525         transport="http://schemas.xmlsoap.org/soap/http" />
3526     <wSDL:operation name="GetLastTradePrice">
3527         <soap:operation soapAction="urn:GetLastTradePrice" style="document" />
3528         <wSDL:input name="GetLastTradePrice">
3529             <soap:body use="literal" />
3530         </wSDL:input>
3531         <wSDL:output name="GetLastTradePriceResponse">
3532             <soap:body use="literal" />
3533         </wSDL:output>
3534     </wSDL:operation>
3535 </binding>
3536
3537 <service name="StockQuoteService">
3538     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3539         <soap:address location="REPLACE_WITH_ACTUAL_URL" />
3540     </port>
3541 </service>
3542 </definitions>

```

3543 [Snippet C-6: Example @WebOperation Annotation](#)

## 3544 C.1.4 @OneWay

3545 Annotation on a C function indicating that it represents a one-way request. The @OneWay annotation  
3546 also affects the service interface. See @OneWay.

3547  
3548 **Corresponds to:** javax.jws.OneWay annotation in the JAX-WS specification (7.11.3)

3549  
3550 **Format:**

```
3551 /* @OneWay */
```

3552  
3553 [Snippet C-7: @OneWay Annotation Format](#)

3554 **Applies to:** Function.

3555  
3556 **Example:**

3557 Input C header file:

```

3558 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3559 *             serviceName="StockQuoteService") */
3560
3561 /* @WebFunction(operationName="SetTradePrice",
3562 *             action="urn:SetTradePrice")
3563 * @OneWay */
3564 void setTradePrice(const char *tickerSymbol, float price);

```

3565

3566 Generated WSDL file:

```
3567 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3568     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3569     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3570     xmlns:tns="http://www.example.org/"
3571     targetNamespace="http://www.example.org/">
3572
3573     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3574         xmlns:tns="http://www.example.org/"
3575         attributeFormDefault="unqualified"
3576         elementFormDefault="unqualified"
3577         targetNamespace="http://www.example.org/">
3578         <xs:element name="SetTradePrice" type="tns:SetTradePrice"/>
3579         <xs:complexType name="SetTradePrice">
3580             <xs:sequence>
3581                 <xs:element name="tickerSymbol" type="xs:string"/>
3582                 <xs:element name="price" type="xs:float"/>
3583             </xs:sequence>
3584         </xs:complexType>
3585     </xs:schema>
3586
3587     <-message name="SetTradePrice">
3588         <part name="parameters" element="tns:SetTradePrice">
3589             </part>
3590     </message>
3591
3592     <portType name="StockQuote">
3593         <sca-c:bindings>
3594             <sca-c:prefix name="stockQuote"/>
3595         </sca-c:bindings>
3596         <operation name="SetTradePrice">
3597             <sca-c:bindings>
3598                 <sca-c:function name="setTradePrice"/>
3599             </sca-c:bindings>
3600             <input name="SetTradePrice" message="tns:SetTradePrice">
3601                 </input>
3602             </operation>
3603     </portType>
3604
3605     <binding name="StockQuoteServiceSoapBinding">
3606         <soap:binding style="document"
3607             transport="http://schemas.xmlsoap.org/soap/http"/>
3608         <wSDL:operation name="SetTradePrice">
3609             <soap:operation soapAction="urn:SetTradePrice" style="document"/>
3610             <wSDL:input name="SetTradePrice">
3611                 <soap:body use="literal"/>
3612             </wSDL:input>
3613         </wSDL:operation>
3614     </binding>
3615
3616     <service name="StockQuoteService">
3617         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3618             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3619         </port>
3620     </service>
3621 </definitions>
```

3622 [Snippet C-8: Example @OneWay Annotation](#)

## 3623 C.1.5 @WebParam

3624 Annotation on a C function indicating the mapping of a parameter to the associated input and output  
3625 WSDL messages. Or on a C struct indicating the mapping of a member to the associated WSDL  
3626 message.

3627

3628 **Corresponds to:** javax.jws.WebParam annotation in the JAX-WS specification (7.11.4)

3629

3630 **Format:**

```
3631 /* @WebParam(paramName="parameter", name="WSDLElement",  
3632 *           targetNamespace="namespaceURI", mode="IN"|"OUT"|"INOUT",  
3633 *           header="false", partName="WSDLPart", type="xsdType") */
```

3634 *Snippet C-9: @WebParam Annotation Format*

3635 where:

- 3636 • **paramName : NCName (1..1)** – specifies the name of the parameter that this annotation applies to.  
3637 Only named parameters MAY be referenced by a @WebParam annotation. The value of the  
3638 paramName of a @WebParam annotation MUST be the name of a parameter of the function the  
3639 annotation is applied to. [CC0009]
- 3640 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. -The default  
3641 value is the name of the parameter. -If an @WebParam annotation is not present, and the parameter  
3642 is unnamed, then a name of “argN”, where N is an incrementing value from 1 indicating the position of  
3643 he parameter in the argument list, will be used.
- 3644 • **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default  
3645 namespace is is the namespace of the associated @WebService. -The targetNamespace attribute is  
3646 ignored unless the binding style is document, and the binding parameterStyle is bare. -See  
3647 @SOAPBinding.
- 3648 • **mode : token (0..1)** – specifies whether the parameter is associated with the input message, output  
3649 message, or both. -The default value is determined by the passing mechanism for the parameter. See  
3650 Method Parameters and Return Type.
- 3651 • **header : boolean (0..1)** – specifies whether this parameter is associated with a SOAP header  
3652 element. -The default value is “false”.
- 3653 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The  
3654 default value is the value of name.
- 3655 • **type : NCNameQName (0..1)** – specifies the XML Schema type of the WSDL part or element  
3656 associated with this parameter. The value of the type property of a @WebParam annotation MUST  
3657 be either one of the simpleTypes defined in namespace  
3658 http://www.w3.org/2001/XMLSchema-~~http://www.w3.org/2001/XMLSchema~~ or, if the type of the  
3659 parameter is a struct, the QName of a XSD complex type following the mapping specified in Complex  
3660 Content Binding. [CC0006] The default type is determined by the mapping defined in Simple  
3661 ContentData Binding.

3662

3663 **Applies to:** Function parameter or struct member.

3664

3665 Example:

3666 Input C header file:

```
3667 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
3668 *           serviceName="StockQuoteService") */  
3669  
3670 /* @WebFunction(operationName="GetLastTradePrice",  
3671 *           action="urn:GetLastTradePrice")
```

```
3672 * @WebParam(paramName="tickerSymbol", name="symbol", mode="IN") */
3673 float getLastTradePrice(char *tickerSymbol);
```

3674

3675 Generated WSDL file:

```
3676 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3677             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3678             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3679             xmlns:tns="http://www.example.org/"
3680             targetNamespace="http://www.example.org/">
3681
3682     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3683               xmlns:tns="http://www.example.org/"
3684               attributeFormDefault="unqualified"
3685               elementFormDefault="unqualified"
3686               targetNamespace="http://www.example.org/">
3687       <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3688       <xs:element name="GetLastTradePriceResponse"
3689                 type="tns:GetLastTradePriceResponse"/>
3690       <xs:complexType name="GetLastTradePrice">
3691         <xs:sequence>
3692           <xs:element name="symbol" type="xs:string"/>
3693         </xs:sequence>
3694       </xs:complexType>
3695       <xs:complexType name="GetLastTradePriceResponse">
3696         <xs:sequence>
3697           <xs:element name="return" type="xs:float"/>
3698         </xs:sequence>
3699       </xs:complexType>
3700     </xs:schema>
3701
3702     <-message name="GetLastTradePrice">
3703       <part name="parameters" element="tns:GetLastTradePrice">
3704         </part>
3705     </message>
3706
3707     <-message name="GetLastTradePriceResponse">
3708       <part name="parameters" element="tns:GetLastTradePriceResponse">
3709         </part>
3710     </-message>
3711
3712     <portType name="StockQuote">
3713       <sca-c:bindings>
3714         <sca-c:prefix name="stockQuote"/>
3715       </sca-c:bindings>
3716       <operation name="GetLastTradePrice">
3717         <sca-c:bindings>
3718           <sca-c:function name="getLastTradePrice"/>
3719           <sca-c:parameter name="tickerSymbol"
3720                         part="tns:GetLastTradePrice/parameter"
3721                         childElementName="symbol"/>
3722         </sca-c:bindings>
3723         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3724           </input>
3725         <output name="GetLastTradePriceResponse"
3726                message="tns:GetLastTradePriceResponse">
3727           </output>
3728         </operation>
3729     </portType>
3730
3731     <binding name="StockQuoteServiceSoapBinding">
3732       <soap:binding style="document"
3733                   transport="http://schemas.xmlsoap.org/soap/http"/>
3734     <wsdl:operation name="GetLastTradePrice">
```

```

3735     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3736     <wSDL:input name="GetLastTradePrice">
3737         <soap:body use="literal"/>
3738     </wSDL:input>
3739     <wSDL:output name="GetLastTradePriceResponse">
3740         <soap:body use="literal"/>
3741     </wSDL:output>
3742     </wSDL:operation>
3743 </binding>
3744
3745     <service name="StockQuoteService">
3746         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3747             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3748         </port>
3749     </service>
3750 </definitions>

```

3751 [Snippet C-10: Example @WebParam Annotation](#)

## 3752 C.1.6 @WebResult

3753 Annotation on a C function indicating the mapping of the function's return type to the associated output  
3754 WSDL message.

3755

3756 **Corresponds to:** `javax.jws.WebResult` annotation in the JAX-WS specification (7.11.5)

3757

3758 **Format:**

```

3759     /* @WebResult(name="WSDLElement", targetNamespace="namespaceURI",
3760     *             header="false", partName="WSDLPart", type="xsdType") */

```

3761 [Snippet C-11: @WebResult Annotation Format](#)

3762 where:

- 3763 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. -The default  
3764 value is “return”.
- 3765 • **targetNamespace : string (0..1)** – specifies the target namespace for the part. -The default  
3766 namespace is the namespace of the associated @WebService. -The targetNamespace attribute is  
3767 ignored unless the binding style is document, and the binding parameterStyle is bare. -(See  
3768 @SOAPBinding).
- 3769 • **header : boolean (0..1)** – specifies whether the result is associated with a SOAP header element.  
3770 The default value is “false”.
- 3771 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The  
3772 default value is the value of name.
- 3773 • **type : NCName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with  
3774 this parameter. The value of the type property of a @WebResult annotation MUST be one of the  
3775 simpleTypes defined in namespace `http://www.w3.org/2001/XMLSchema`. [CC0007] The default type  
3776 is determined by the mapping defined in 11.3.1.

3777

3778 **Applies to:** Function. \_\_\_\_\_

3779

3780 Example:

3781 Input C header file:

```

3782     /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3783     *             serviceName="StockQuoteService") */
3784

```

```
3785 /* @WebFunction(operationName="GetLastTradePrice",
3786 *             action="urn:GetLastTradePrice")
3787 * @WebResult(name="price") */
3788 float getLastTradePrice(const char *tickerSymbol);
```

3789

3790 Generated WSDL file:

```
3791 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3792             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3793             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3794             xmlns:tns="http://www.example.org/"
3795             targetNamespace="http://www.example.org/">
3796
3797     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3798               xmlns:tns="http://www.example.org/"
3799               attributeFormDefault="unqualified"
3800               elementFormDefault="unqualified"
3801               targetNamespace="http://www.example.org/">
3802         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3803         <xs:element name="GetLastTradePriceResponse"
3804                   type="tns:GetLastTradePriceResponse"/>
3805         <xs:complexType name="GetLastTradePrice">
3806             <xs:sequence>
3807                 <xs:element name="tickerSymbol" type="xs:string"/>
3808             </xs:sequence>
3809         </xs:complexType>
3810         <xs:complexType name="GetLastTradePriceResponse">
3811             <xs:sequence>
3812                 <xs:element name="price" type="xs:float"/>
3813             </xs:sequence>
3814         </xs:complexType>
3815     </xs:schema>
3816
3817     <-message name="GetLastTradePrice">
3818         <part name="parameters" element="tns:GetLastTradePrice">
3819             </part>
3820     </message>
3821
3822     <-message name="GetLastTradePriceResponse">
3823         <part name="parameters" element="tns:GetLastTradePriceResponse">
3824             </part>
3825     </message>
3826
3827     <portType name="StockQuote">
3828         <sca-c:bindings>
3829             <sca-c:prefix name="stockQuote"/>
3830         </sca-c:bindings>
3831         <operation name="GetLastTradePrice">
3832             <sca-c:bindings>
3833                 <sca-c:function name="getLastTradePrice"/>
3834             </sca-c:bindings>
3835             <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3836                 </input>
3837             <output name="GetLastTradePriceResponse"
3838                    message="tns:GetLastTradePriceResponse">
3839                 </output>
3840             </operation>
3841         </portType>
3842
3843     <binding name="StockQuoteServiceSoapBinding">
3844         <soap:binding style="document"
3845                      transport="http://schemas.xmlsoap.org/soap/http"/>
3846         <wSDL+operation name="GetLastTradePrice">
3847             <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
```

```

3848 |         <wSDL:input name="GetLastTradePrice">
3849 |             <soap:body use="literal"/>
3850 |         </wSDL:input>
3851 |         <wSDL:output name="GetLastTradePriceResponse">
3852 |             <soap:body use="literal"/>
3853 |         </wSDL:output>
3854 |     </wSDL:operation>
3855 | </binding>
3856 |
3857 |     <service name="StockQuoteService">
3858 |         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3859 |             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3860 |         </port>
3861 |     </service>
3862 | </definitions>

```

3863 | [Snippet C-12: Example @WebResult Annotation](#)

## 3864 | C.1.7 @SOAPBinding

3865 | Annotation on a C WebService or function specifying the mapping of the web service onto the SOAP  
3866 | message protocol.

3867 |

3868 | **Corresponds to:** javax.jws.SOAPBinding annotation in the JAX-WS specification (7.11.6)

3869 |

3870 | **Format:**

```

3871 | /* @SOAPBinding(style="DOCUMENT" | "RPC", use="LITERAL" | "ENCODED",
3872 | *             parameterStyle="BARE" | "WRAPPED") */

```

3873 | [Snippet C-13: @SOAPBinding Annotation Format](#)

3874 | where:

- 3875 | • **style : token (0..1)** – specifies the WSDL binding style. -The default value is “DOCUMENT”.
- 3876 | • **use : token (0..1)** – specifies the WSDL binding use. -The default value is “LITERAL”.
- 3877 | • **parameterStyle : token (0..1)** – specifies the WSDL parameter style. -The default value is  
3878 | “WRAPPED”.

3879 |

3880 | **Applies to:** WebService, Function.

3881 |

3882 | **Example:**

3883 | Input C header file:

```

3884 | /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3885 | *             serviceName="StockQuoteService") */
3886 | * @SOAPBinding(style="RPC") */
3887 |
3888 | ...

```

3889 |

3890 | **Generated WSDL file:**

```

3891 | <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3892 |             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3893 |             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3894 |             xmlns:tns="http://www.example.org/"
3895 |             targetNamespace="http://www.example.org/">
3896 |
3897 |     <portType name="StockQuote">
3898 |         <sca-c:bindings>

```



```

3899     <sca-c:prefix name="stockQuote" />
3900     </sca-c:bindings>
3901 </portType>
3902
3903 <binding name="StockQuoteServiceSoapBinding">
3904     <soap:binding style="rpc"
3905         transport="http://schemas.xmlsoap.org/soap/http" />
3906 </binding>
3907
3908 <service name="StockQuoteService">
3909     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3910         <soap:address location="REPLACE_WITH_ACTUAL_URL" />
3911     </port>
3912 </service>
3913 </definitions>

```

3914 [Snippet C-14: Example @SOAPBinding Annotation](#)

### 3915 C.1.8 @WebFault

3916 Annotation on a C struct indicating that it format of a fault message.

3917

3918 **Corresponds to:** javax.xml.ws.WebFault annotation in the JAX-WS specification (7.2)

3919

3920 **Format:**

```
3921 /* @WebFault(name="WSDL_Element", targetNamespace="namespaceURI") */
```

3922 [Snippet C-15: @WebFault Annotation Format](#)

3923 where:

- 3924 • **name : NCName (1..1)** – specifies the local name of the global element mapped to this fault.
- 3925 • **targetNamespace : string (0..1)** – specifies the namespace of the global element mapped to this fault. -The default namespace is determined by the implementation.

3927

3928 **Applies to:** struct.

3929

3930 Example:

3931 Input C header file:

```

3932 /* @WebFault(name="UnknownSymbolFault",
3933 *           targetNamespace="http://www.example.org/")
3934 struct UnkSymMsg {
3935     char faultInfo[10];
3936 } unkSymInfo;
3937
3938 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3939 *           serviceName="StockQuoteService") */
3940
3941 /* @WebFunction(operationName="GetLastTradePrice",
3942 *           action="urn:GetLastTradePrice")
3943 * @WebThrows(faults="unkSymInfoUnkSymMsg") */
3944 float getLastTradePrice(const char *tickerSymbol);

```

3945

3946 Generated WSDL file:

```

3947 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3948     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3949     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"

```



```

3950     xmlns:tns="http://www.example.org/"
3951     targetNamespace="http://www.example.org/"
3952
3953 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3954     xmlns:tns="http://www.example.org/"
3955     attributeFormDefault="unqualified"
3956     elementFormDefault="unqualified"
3957     targetNamespace="http://www.example.org/">
3958   <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3959   <xs:element name="GetLastTradePriceResponse"
3960     type="tns:GetLastTradePriceResponse"/>
3961   <xs:complexType name="GetLastTradePrice">
3962     <xs:sequence>
3963       <xs:element name="tickerSymbol" type="xs:string"/>
3964     </xs:sequence>
3965   </xs:complexType>
3966   <xs:complexType name="GetLastTradePriceResponse">
3967     <xs:sequence>
3968       <xs:element name="return" type="xs:float"/>
3969     </xs:sequence>
3970   </xs:complexType>
3971   <xs:simpleType name="UnknownSymbolFaultType">
3972     <xs:restriction base="xs:string">
3973       <xsd:maxLength value="9"/>
3974     </xs:restriction>
3975   </xs:simpleType>
3976   <xs:element name="UnknownSymbolFault" type="UnknownSymbolFaultType"/>
3977 </xs:schema>
3978
3979 <message name="GetLastTradePrice">
3980   <part name="parameters" element="tns:GetLastTradePrice">
3981     </part>
3982 </message>
3983
3984 <message name="GetLastTradePriceResponse">
3985   <part name="parameters" element="tns:GetLastTradePriceResponse">
3986     </part>
3987 </message>
3988
3989 <message name="UnknownSymbol">
3990   <sca-c:bindings>
3991     <sca-c:struct name="unkSymInfounkSymMsg"/>
3992   </sca-c:bindings>
3993   <part name="parameters" element="tns:UnknownSymbolFault">
3994     </part>
3995 </message>
3996
3997 <portType name="StockQuote">
3998   <sca-c:bindings>
3999     <sca-c:prefix name="stockQuote"/>
4000 </sca-c:bindings>
4001   <operation name="GetLastTradePrice">
4002     <sca-c:bindings>
4003       <sca-c:function name="getLastTradePrice"/>
4004     </sca-c:bindings>
4005     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
4006     </input>
4007     <output name="GetLastTradePriceResponse"
4008       message="tns:GetLastTradePriceResponse">
4009     </output>
4010     <fault name="UnknownSymbol" message="tns:UnknownSymbol">
4011     </fault>
4012   </operation>
4013 </portType>

```

```

4014
4015     <binding name="StockQuoteServiceSoapBinding">
4016         <soap:binding style="document"
4017             transport="http://schemas.xmlsoap.org/soap/http"/>
4018         <wSDL:operation name="GetLastTradePrice">
4019             <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
4020             <wSDL:input name="GetLastTradePrice">
4021                 <soap:body use="literal"/>
4022             </wSDL:input>
4023             <wSDL:output name="GetLastTradePriceResponse">
4024                 <soap:body use="literal"/>
4025             </wSDL:output>
4026             <wSDL:fault>
4027                 <soap:fault name="UnknownSymbol" use="literal"/>
4028             </wSDL:fault>
4029         </wSDL:operation>
4030     </binding>
4031
4032     <service name="StockQuoteService">
4033         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
4034             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
4035         </port>
4036     </service>
4037 </definitions>

```

4038 [Snippet C-16: Example @WebFault Annotation](#)

## 4039 C.1.9 @WebThrows

4040 Annotation on a C function or operation indicating which faults might be thrown by this function or  
4041 operation.

4042

4043 **Corresponds to:** No equivalent in JAX-WS.

4044

4045 **Format:**

```
4046 /* @WebThrows(faults="faultMsg1" [, "faultMsgn"]*) */
```

4047 [Snippet C-17: @WebThrows Annotation Format](#)

4048 where:

- 4049 • **faults : NMTOKEN (1..n)** – specifies the names of all faults that might be thrown by this function or  
4050 operation. -The name of the fault is the name of its associated C struct name. -A C struct that is listed  
4051 in a @WebThrows annotation MUST itself have a @WebFault annotation. [CC0004]

4052

4053 **Applies to:** Function or Operation

4054

4055 **Example:**

4056 See @WebFault.

## 4057 D C WSDL Mapping Extensions

4058 | The following WSDL extensions are used to augment the conversion process from WSDL to C. -All of  
4059 | these extensions are defined in the namespace `http://docs.oasis-open.org/ns/opencsa/sca-c-`  
4060 | `cpp/c/200901`. -For brevity, all definitions of these extensions will be fully qualified, and all references to  
4061 | the "sca-c" prefix are associated with the namespace above. If WSDL extensions are supported by an  
4062 | implementation, all the extensions defined here MUST be supported and MUST be mapped to C as  
4063 | described. [CD0001]

### 4064 D.1 <sca-c:bindings>

4065 | <sca-c:bindings> is a container type which can be used as a WSDL extension. -All other SCA wsdl  
4066 | extensions will be specified as children of a <sca-c:bindings> element. -An <sca-c:bindings> element can  
4067 | be used as an extension to any WSDL type that accepts extensions.

### 4068 D.2 <sca-c:prefix>

4069 | <sca-c:prefix> provides a mechanism for defining an alternate prefix for the functions or structs  
4070 | implementing the operations of a portType.

4071

4072 **Format:**

```
4073 <sca-c:prefix name="portTypePrefix"/>
```

4074 *Snippet D-1: <sca-c:prefix> Element Format*

4075 where:

- 4076 • **prefix/@name : string (1..1)** – specifies the string to prepend to an operation name when generating  
4077 a C function or structure name.

4078

4079 **Applicable WSDL element(s):**

- 4080 • `wSDL:portType`

4081

4082 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:prefix/> child element. [CD0003]

4083

4084 **Example:**

4085 **Input WSDL file:**

```
4086 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
4087     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
4088     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
4089     xmlns:tns="http://www.example.org/"  
4090     targetNamespace="http://www.example.org/">  
4091  
4092     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
4093         xmlns:tns="http://www.example.org/"  
4094         attributeFormDefault="unqualified"  
4095         elementFormDefault="unqualified"  
4096         targetNamespace="http://www.example.org/">  
4097         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
4098         <xs:element name="GetLastTradePriceResponse"  
4099             type="tns:GetLastTradePriceResponse"/>  
4100         <xs:complexType name="GetLastTradePrice">  
4101             <xs:sequence>
```

```

4102         <xs:element name="tickerSymbol" type="xs:string"/>
4103     </xs:sequence>
4104 </xs:complexType>
4105 <xs:complexType name="GetLastTradePriceResponse">
4106     <xs:sequence>
4107         <xs:element name="return" type="xs:float"/>
4108     </xs:sequence>
4109 </xs:complexType>
4110 </xs:schema>
4111
4112 <-message name="GetLastTradePrice">
4113     <part name="parameters" element="tns:GetLastTradePrice">
4114         </part>
4115 </message>
4116
4117 <-message name="GetLastTradePriceResponse">
4118     <part name="parameters" element="tns:GetLastTradePriceResponse">
4119         </part>
4120 </-message>
4121
4122 <portType name="StockQuote">
4123     <sca-c:bindings>
4124         <sca-c:prefix name="stockQuote"/>
4125     </sca-c:bindings>
4126     <operation name="GetLastTradePrice">
4127         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
4128             </input>
4129         <output name="GetLastTradePriceResponse"
4130             message="tns:GetLastTradePriceResponse">
4131             </output>
4132     </operation>
4133 </portType>
4134
4135 <binding name="StockQuoteServiceSoapBinding">
4136     <soap:binding style="document"
4137         transport="http://schemas.xmlsoap.org/soap/http"/>
4138     <wSDL:operation name="GetLastTradePrice">
4139         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
4140         <wSDL:input name="GetLastTradePrice">
4141             <soap:body use="literal"/>
4142         </wSDL:input>
4143         <wSDL:output name="GetLastTradePriceResponse">
4144             <soap:body use="literal"/>
4145         </wSDL:output>
4146     </wSDL:operation>
4147 </binding>
4148
4149 <service name="StockQuoteService">
4150     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
4151         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
4152     </port>
4153 </service>
4154 </definitions>

```

4155

#### Generated C header file:

```

4157 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
4158 *     serviceName="StockQuoteService") */
4159
4160 /* @WebFunction(operationName="GetLastTradePrice",
4161 *     action="urn:GetLastTradePrice") */
4162 float stockQuoteGetLastTradePrice(const char *tickerSymbol);

```

4163 [Snippet D-2: Example <sca-c:prefix> Element](#)

### 4164 **D.3 <sca-c:enableWrapperStyle>**

4165 <sca-c:enableWrapperStyle> indicates whether or not the wrapper style for messages is applied, when  
4166 otherwise applicable. -If false, the wrapper style will never be applied.

4167

4168 **Format:**

```
4169 <sca-c:enableWrapperStyle>value</sca-c:enableWrapperStyle>
```

4170 *Snippet D-3: <sca-c:enableWrapperStyle> Element Format*

4171 where:

- 4172 • **enableWrapperStyle/text() : boolean (1..1)** – specifies whether wrapper style is enabled or disabled  
4173 for this element and any of its children. -The default value is "true".

4174

4175 **Applicable WSDL element(s):**

- 4176 • wsdl:definitions
- 4177 • wsdl:portType – overrides a binding applied to wsdl:definitions
- 4178 • wsdl:portType/wsdl:operation – overrides a binding applied to wsdl:definitions or the enclosing  
4179 wsdl:portType

4180

4181 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:enableWrapperStyle/> child  
4182 element. [CD0004]

4183

4184 **Example:**

4185 Input WSDL file:

```
4186 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
4187   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
4188   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
4189   xmlns:tns="http://www.example.org/"  
4190   targetNamespace="http://www.example.org/">  
4191  
4192   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
4193     xmlns:tns="http://www.example.org/"  
4194     attributeFormDefault="unqualified"  
4195     elementFormDefault="unqualified"  
4196     targetNamespace="http://www.example.org/">  
4197     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice" />  
4198     <xs:element name="GetLastTradePriceResponse"  
4199       type="tns:GetLastTradePriceResponse" />  
4200     <xs:complexType name="GetLastTradePrice">  
4201       <xs:sequence>  
4202         <xs:element name="tickerSymbol" type="xs:string" />  
4203       </xs:sequence>  
4204     </xs:complexType>  
4205     <xs:complexType name="GetLastTradePriceResponse">  
4206       <xs:sequence>  
4207         <xs:element name="return" type="xs:float" />  
4208       </xs:sequence>  
4209     </xs:complexType>  
4210   </xs:schema>  
4211  
4212   <-message name="GetLastTradePrice">  
4213     <part name="parameters" element="tns:GetLastTradePrice">  
4214     </part>  
4215   </message>  
4216
```

```

4217 | <-message name="GetLastTradePriceResponse">
4218 |   <part name="parameters" element="tns:GetLastTradePriceResponse">
4219 |     </part>
4220 | </-message>
4221 |
4222 | <portType name="StockQuote">
4223 |   <sca-c:bindings>
4224 |     <sca-c:prefix name="stockQuote"/>
4225 |     <sca-c:enableWrapperStyle>>false</sca-c:enableWrapperStyle>
4226 |   </sca-c:bindings>
4227 |   <operation name="GetLastTradePrice">
4228 |     <sca-c:bindings>
4229 |       <sca-c:function name="getLastTradePrice"/>
4230 |     </sca-c:bindings>
4231 |     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
4232 |       </input>
4233 |     <output name="GetLastTradePriceResponse"
4234 |       message="tns:GetLastTradePriceResponse">
4235 |       </output>
4236 |     </operation>
4237 |   </portType>
4238 | </definitions>

```

4239

Generated C header file:

```

4241 | /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
4242 | *           serviceName="StockQuoteService") */
4243 |
4244 | /* @WebFunction(operationName="GetLastTradePrice",
4245 | *           action="urn:GetLastTradePrice") */
4246 | DATAOBJECT getLastTradePrice(DATAOBJECT parameters);

```

4247 | [Snippet D-4: Example <sca-c:enableWrapperStyle> Element](#)

## 4248 | D.4 <sca-c:function>

4249 | <sca-c:function> specifies the name of the C function that the associated WSDL operation is associated  
4250 | with. -If <sca-c:function> is used, the portType prefix, either default or a specified with <sca-c:prefix> is  
4251 | not prepended to the function name.

4252 |

4253 | **Format:**

```

4254 | <sca-c:function name="myFunction"/>

```

4255 | [Snippet D-5: <sca-c:function> Element Format](#)

4256 | where:

- 4257 | • **function/@name : NCName (1..1)** – specifies the name of the C function associated with this WSDL  
4258 | operation.

4259 |

4260 | **Applicable WSDL element(s):**

- 4261 | • wsdl:portType/wsdl:operation

4262 |

4263 | A <sca-c:bindings/> element MUST NOT have more than one <sca-c:function/> child element. [CD0005]

4264 |

4265 | Example:

4266 | Input WSDL file:

```

4267 | <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"

```

```

4268     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4269     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
4270     xmlns:tns="http://www.example.org/"
4271     targetNamespace="http://www.example.org/"
4272
4273     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
4274             xmlns:tns="http://www.example.org/"
4275             attributeFormDefault="unqualified"
4276             elementFormDefault="unqualified"
4277             targetNamespace="http://www.example.org/">
4278     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
4279     <xs:element name="GetLastTradePriceResponse"
4280             type="tns:GetLastTradePriceResponse"/>
4281     <xs:complexType name="GetLastTradePrice">
4282         <xs:sequence>
4283             <xs:element name="tickerSymbol" type="xs:string"/>
4284         </xs:sequence>
4285     </xs:complexType>
4286     <xs:complexType name="GetLastTradePriceResponse">
4287         <xs:sequence>
4288             <xs:element name="return" type="xs:float"/>
4289         </xs:sequence>
4290     </xs:complexType>
4291 </xs:schema>
4292
4293 <-message name="GetLastTradePrice">
4294     <part name="parameters" element="tns:GetLastTradePrice">
4295     </part>
4296 </message>
4297
4298 <-message name="GetLastTradePriceResponse">
4299     <part name="parameters" element="tns:GetLastTradePriceResponse">
4300     </part>
4301 </-message>
4302
4303 <portType name="StockQuote">
4304     <sca-c:bindings>
4305         <sca-c:prefix name="stockQuote"/>
4306     </sca-c:bindings>
4307     <operation name="GetLastTradePrice">
4308         <sca-c:bindings>
4309             <sca-c:function name="getTradePrice"/>
4310         </sca-c:bindings>
4311         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
4312         </input>
4313         <output name="GetLastTradePriceResponse"
4314             message="tns:GetLastTradePriceResponse">
4315         </output>
4316     </operation>
4317 </portType>
4318 </definitions>

```

4319

#### Generated C header file:

```

4321 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
4322 *         serviceName="StockQuoteService") */
4323
4324 /* @WebFunction(operationName="GetLastTradePrice",
4325 *         action="urn:GetLastTradePrice") */
4326 float getTradePrice(const wchar_t *tickerSymbol);

```

4327 [Snippet D-6: Example <sca-c:function> Element](#)



## 4328 D.5 <sca-c:struct>

4329 <sca-c:struct> specifies the name of the C struct that the associated WSDL message is associated with. If  
4330 <sca-c:struct> is used for an operation request or response message, the portType prefix, either default  
4331 or a specified with <sca-c:prefix> is not prepended to the struct name.

4332

4333

4334 **Format:**

```
4335 <sca-c:struct name="myStruct" />
```

4336 *Snippet D-7: <sca-c:struct> Element Format*

4337 where:

- 4338 • **struct/@name : NCName (1..1)** – specifies the name of the C struct associated with this WSDL  
4339 message.

4340

4341 **Applicable WSDL element(s):**

- 4342 • wsdl:message

4343

4344 A <sca-c:bindings/> element MUST NOT have more than one <sca-c:struct/> child element. [CD0006]

4345

4346 **Example:**

4347 Input WSDL file:

```
4348 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
4349   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
4350   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
4351   xmlns:tns="http://www.example.org/"  
4352   targetNamespace="http://www.example.org/">  
4353  
4354   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
4355     xmlns:tns="http://www.example.org/"  
4356     attributeFormDefault="unqualified"  
4357     elementFormDefault="unqualified"  
4358     targetNamespace="http://www.example.org/">  
4359     <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice" />  
4360     <xs:element name="GetLastTradePriceResponse"  
4361       type="tns:GetLastTradePriceResponse" />  
4362     <xs:complexType name="GetLastTradePrice">  
4363       <xs:sequence>  
4364         <xs:element name="tickerSymbol" type="xs:string" />  
4365       </xs:sequence>  
4366     </xs:complexType>  
4367     <xs:complexType name="GetLastTradePriceResponse">  
4368       <xs:sequence>  
4369         <xs:element name="return" type="xs:float" />  
4370       </xs:sequence>  
4371     </xs:complexType>  
4372   </xs:schema>  
4373  
4374   <-message name="GetLastTradePrice">  
4375     <sca-c:bindings>  
4376       <sca-c:struct name="getTradePrice" />  
4377     </sca-c:bindings>  
4378     <part name="parameters" element="tns:GetLastTradePrice">  
4379     </part>  
4380   </message>  
4381
```



```

4382 | <-message name="GetLastTradePriceResponse">
4383 |   <sca-c:bindings>
4384 |     <sca-c:struct name="getTradePriceResponse"/>
4385 |   </sca-c:bindings>
4386 |   <part name="parameters" element="tns:GetLastTradePriceResponse">
4387 |     </part>
4388 | </-message>
4389 |
4390 | <portType name="StockQuote">
4391 |   <sca-c:bindings>
4392 |     <sca-c:prefix name="stockQuote"/>
4393 |   </sca-c:bindings>
4394 |   <operation name="GetLastTradePrice">
4395 |     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
4396 |       </input>
4397 |     <output name="GetLastTradePriceResponse"
4398 |       message="tns:GetLastTradePriceResponse">
4399 |       </output>
4400 |   </operation>
4401 | </portType>
4402 | </definitions>

```

4403

4404 Generated C header file:

```

4405 | /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
4406 | *           serviceName="StockQuoteService") */
4407 |
4408 | /* @WebOperation(operationName="GetLastTradePrice",
4409 | *           response="getLastTradePriseResponse"
4410 | *           action="urn:GetLastTradePrice") */
4411 | struct getLastTradePrice {
4412 |     wchar_t *tickerSymbol; /* Since the length of the element is not
4413 | *           * restricted, a pointer is returned with the
4414 | *           * actual value held by the SCA runtime. */
4415 | };
4416 |
4417 | struct getLastTradePriceResponse {
4418 |     float return;
4419 | };

```

4420 [Snippet D-8: Example <sca-c:struct> Element](#)

## 4421 D.6 <sca-c:parameter>

4422 <sca-c:parameter> specifies the name of the C function parameter or struct member associated with a  
4423 specific WSDL message part or wrapper child element.

4424

4425 **Format:**

```

4426 | <sca-c:parameter name="CParameter" part="WSDLPart"
4427 |   childElementName="WSDLElement" type="CType"/>

```

4428 [Snippet D-9: <sca-c:parameter> Element Format](#)

4429 where:

- 4430 • **parameter/@name : NCName (1..1)** – specifies the name of the C function parameter or struct  
4431 member associated with this WSDL operation part or wrapper child element. “return” is used to  
4432 denote the return value.
- 4433 • **parameter/@part : string (1..1)** - an XPath expression identifying the wsdl:part of a wsdl:message.
- 4434 • **parameter/@childElementName : QName (1..1)** – specifies the qualified name of a child element of  
4435 the global element identified by parameter/@part.

- 4436 • **parameter/@type : NCNamestring (0..1)** – specifies the type of the parameter or struct member or  
4437 return type. The @type attribute of a <parameter/> element MUST be either a valid C type, specified  
4438 in Simple Content Binding or, if the message part has complex content, a struct following the mapping  
4439 specified in Complex Content Binding. [CD0002] The default type is determined by the mapping  
4440 defined in Data Binding.

4441

4442 **Applicable WSDL element(s):**

- 4443 • wsdl:portType/wsdl:operation

4444

4445

4446 **Example:**

4447 **Input WSDL file:**

```
4448 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
4449     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4450     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
4451     xmlns:tns="http://www.example.org/"
4452     targetNamespace="http://www.example.org/">
4453
4454     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
4455         xmlns:tns="http://www.example.org/"
4456         attributeFormDefault="unqualified"
4457         elementFormDefault="unqualified"
4458         targetNamespace="http://www.example.org/">
4459         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
4460         <xs:element name="GetLastTradePriceResponse"
4461             type="tns:GetLastTradePriceResponse"/>
4462         <xs:complexType name="GetLastTradePrice">
4463             <xs:sequence>
4464                 <xs:element name="symbol" type="xs:string"/>
4465             </xs:sequence>
4466         </xs:complexType>
4467         <xs:complexType name="GetLastTradePriceResponse">
4468             <xs:sequence>
4469                 <xs:element name="return" type="xs:float"/>
4470             </xs:sequence>
4471         </xs:complexType>
4472     </xs:schema>
4473
4474     <-message name="GetLastTradePrice">
4475         <part name="parameters" element="tns:GetLastTradePrice">
4476             </part>
4477     </message>
4478
4479     <-message name="GetLastTradePriceResponse">
4480         <part name="parameters" element="tns:GetLastTradePriceResponse">
4481             </part>
4482     </message>
4483
4484     <portType name="StockQuote">
4485         <sca-c:bindings>
4486             <sca-c:prefix name="stockQuote"/>
4487         </sca-c:bindings>
4488         <operation name="GetLastTradePrice">
4489             <sca-c:bindings>
4490                 <sca-c:function name="getLastTradePrice"/>
4491                 <sca-c:parameter name="tickerSymbol"
4492                     part="tns:GetLastTradePrice/parameters"
4493                     childElementName="symbol"/>
4494             </sca-c:bindings>
4495         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
```

```

4496     </input>
4497     <output name="GetLastTradePriceResponse"
4498           message="tns:GetLastTradePriceResponse">
4499     </output>
4500   </operation>
4501 </portType>
4502
4503   <binding name="StockQuoteServiceSoapBinding">
4504     <soap:binding style="document"
4505               transport="http://schemas.xmlsoap.org/soap/http"/>
4506     <wSDL:operation name="GetLastTradePrice">
4507       <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
4508       <wSDL:input name="GetLastTradePrice">
4509         <soap:body use="literal"/>
4510       </wSDL:input>
4511       <wSDL:output name="GetLastTradePriceResponse">
4512         <soap:body use="literal"/>
4513       </wSDL:output>
4514     </wSDL:operation>
4515   </binding>
4516
4517   <service name="StockQuoteService">
4518     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
4519       <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
4520     </port>
4521   </service>
4522 </definitions>

```

4523

Generated C header file:

```

4525 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
4526 *           serviceName="StockQuoteService") */
4527
4528 /* @WebFunction(operationName="GetLastTradePrice",
4529 *           action="urn:GetLastTradePrice")
4530 * @WebParam(paramName="tickerSymbol", name="symbol") */
4531 float getLastTradePrice(const wchar_t *tickerSymbol);

```

4532 *Snippet D-10: Example <sca-c:parameter> Element*

## 4533 D.7 JAX-WS WSDL Extensions

4534 An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL  
4535 extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.  
4536 Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted, and their corresponding SCA  
4537 WSDL extension. [CD0007] The following is a list of JAX-WS WSDL extensions that MAY be recognized,  
4538 and their corresponding SCA WSDL extension.

4539

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

4540 **A.5 WSDL Extensions Schema**

4541 The XML schema pointed to by the RDDL document at the SCA C namespace URI, defined by this  
4542 specification, is considered to be authoritative and takes precedence over the XML schema in this  
4543 appendix.

4544

4545 [Table D-1: Allowed JAX-WS Extensions](#)

4546 **D.8 sca-wsdlext-c-1.1.xsd**

```
4547 <?xml version="1.0" encoding="UTF-8"?>
4548 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4549   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
4550   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
4551   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4552   elementFormDefault="qualified">
4553
4554   <element name="bindings" type="sca-c:BindingsType" />
4555   <complexType name="BindingsType">
4556     <choice minOccurs="0" maxOccurs="unbounded">
4557       <element ref="sca-c:prefix" />
4558       <element ref="sca-c:enableWrapperStyle" />
4559       <element ref="sca-c:function" />
4560       <element ref="sca-c:struct" />
4561       <element ref="sca-c:parameter" />
4562     </choice>
4563   </complexType>
4564
4565   <element name="prefix" type="sca-c:PrefixType" />
4566   <complexType name="PrefixType">
4567     <attribute name="name" type="xsd:string" use="required" />
4568   </complexType>
4569
4570   <element name="function" type="sca-c:FunctionType" />
4571   <complexType name="FunctionType">
4572     <attribute name="name" type="xsd:NCName" use="required" />
4573   </complexType>
4574
4575   <element name="struct" type="sca-c:StructType" />
4576   <complexType name="StructType">
4577     <attribute name="name" type="xsd:NCName" use="required" />
4578   </complexType>
4579
4580   <element name="parameter" type="sca-c:ParameterType" />
4581   <complexType name="ParameterType">
4582     <attribute name="part" type="xsd:string" use="required" />
4583     <attribute name="childElementName" type="xsd:QName" use="required" />
4584     <attribute name="name" type="xsd:NCName" use="required" />
4585     <attribute name="type" type="xsd:string" use="optional" />
4586   </complexType>
4587
4588   <element name="enableWrapperStyle" type="xsd:boolean" />
4589
4590 </schema>
```

4591 [Snippet D-11: SCA C WSDL Extension Schema](#)

4592

## E XML Schemas

4593

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

4594

4595

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

4596

4597

4598

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

4599

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

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

  <element name="interface.c" type="sca:CInterface"
    substitutionGroup="sca:interface" />

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

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

4623

4624

4625

4626

4627

4628

4629

4630

4631

4632

4633

4634

4635

4636

4637

4638

4639

4640

4641

4642

4643

4644

4645

4646

4647 </schema>

4648 [Snippet E-1: SCA <interface.c> Schema](#)

## 4649 E.2 sca-implementation-c-1.1.xsd

```
4650 <?xml version="1.0" encoding="UTF-8"?>
4651 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4652         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
4653         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
4654         elementFormDefault="qualified">
4655
4656     <include schemaLocation="sca-core.xsd" />
4657
4658     <element name="implementation.c" type="sca:CImplementation"
4659             substitutionGroup="sca:implementation" />
4660
4661     <complexType name="CImplementation">
4662         <complexContent>
4663             <extension base="sca:Implementation">
4664                 <sequence>
4665                     <element name="operation" type="sca:CImplementationFunction"
4666                             minOccurs="0" maxOccurs="unbounded" />
4667                     <any namespace="##other" processContents="lax"
4668                             minOccurs="0" maxOccurs="unbounded" />
4669                 </sequence>
4670                 <attribute name="module" type="NCName" use="required" />
4671                 <attribute name="path" type="string" use="optional" />
4672                 <attribute name="library" type="boolean" use="optional" />
4673                 <attribute name="componentType" type="string" use="required" />
4674                 <attribute name="scope" type="sca:CImplementationScope"
4675                         use="optional" />
4676                 <attribute name="eagerInit" type="boolean" use="optional" />
4677                 <attribute name="init" type="boolean" use="optional" />
4678                 <attribute name="destroy" type="boolean" use="optional" />
4679                 <attribute name="allowsPassByReference" type="boolean"
4680                         use="optional" />
4681                 <anyAttribute namespace="##other" processContents="lax" />
4682             </extension>
4683         </complexContent>
4684     </complexType>
4685
4686     <simpleType name="CImplementationScope">
4687         <restriction base="string">
4688             <enumeration value="stateless" />
4689             <enumeration value="composite" />
4690         </restriction>
4691     </simpleType>
4692
4693     <complexType name="CImplementationFunction">
4694         <sequence>
4695             <choice minOccurs="0" maxOccurs="unbounded">
4696                 <element ref="sca:requires" />
4697                 <element ref="sca:policySetAttachment" />
4698             </choice>
4699             <any namespace="##other" processContents="lax" minOccurs="0"
4700                 maxOccurs="unbounded" />
4701         </sequence>
4702         <attribute name="name" type="NCName" use="required" />
4703         <attribute name="requires" type="sca:listOfQNames" use="optional" />
4704         <attribute name="policySets" type="sca:listOfQNames" use="optional" />
```

```
4705     <attribute name="allowsPassByReference" type="boolean"
4706         use="optional"/>
4707     <attribute name="init" type="boolean" use="optional"/>
4708     <attribute name="destroy" type="boolean" use="optional"/>
4709     <anyAttribute namespace="##other" processContents="lax"/>
4710 </complexType>
4711
4712 </schema>
```

4713 [Snippet E-2: SCA <implementation.c> Schema](#)

## 4714 E.3 sca-contribution-c-1.1.xsd

```
4715 <?xml version="1.0" encoding="UTF-8"?>
4716 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4717     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
4718     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903200912"
4719     elementFormDefault="qualified">
4720
4721     <include schemaLocation="sca-contributions.xsd"/>
4722
4723     <element name="export.c" type="sca:CExport"
4724         substitutionGroup="sca:Export"/>
4725
4726     <complexType name="CExport">
4727         <complexContent>
4728             <attribute name="name" type="QName" use="required"/>
4729             <attribute name="path" type="string" use="optional"/>
4730         </complexContent>
4731     </complexType>
4732
4733     <element name="import.c" type="sca:CImport"
4734         substitutionGroup="sca:Import"/>
4735
4736     <complexType name="CImport">
4737         <complexContent>
4738             <attribute name="name" type="QName" use="required"/>
4739             <attribute name="location" type="string" use="required"/>
4740         </complexContent>
4741     </complexType>
4742
4743 </schema>
```

4744

---

## ~~B Conformance Items~~

4745

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

4746

4747

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



## F Normative Statement Summary

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.
[C20003]	<del>An SCA runtime MUST support these scopes; <b>stateless</b> and <b>composite</b>. Additional scopes MAY be provided by SCA runtimes.</del>
[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 <code>@header</code> attribute of an <code>&lt;interface.c/&gt;</code> element contains function <u>or struct</u> declarations that are not operations of the interface, then the functions <u>or structs</u> that <del>define</del> <u>are not</u> operations of the interface MUST be <del>identified</del> <u>excluded</u> using <code>&lt;function/&gt;</code> child elements of the <code>&lt;interface.c/&gt;</code> element with <code>@exclude="true"</code> .
[C20007]	If the header file identified by the <code>@callbackHeader</code> attribute of an <code>&lt;interface.c/&gt;</code> element contains function <u>or struct</u> declarations that are not operations of the callback interface, then the functions <u>or structs</u> that <del>define</del> <u>are not</u> operations of the callback interface MUST be <del>identified</del> <u>excluded</u> using <code>&lt;callbackFunction/&gt;</code> child elements of the <code>&lt;interface.c/&gt;</code> element— with <code>@exclude="true"</code> .
[C20008]	<del>If the header file identified by the <code>@header</code> or <code>@callbackHeader</code> attribute of an <code>&lt;interface.c/&gt;</code> element defines the operations of the interface (callback interface) using message formats, then all functions of the interface (callback interface) MUST be identified using <code>&lt;function/&gt;</code> (<code>&lt;callbackFunction/&gt;</code>) child elements of the <code>&lt;interface.c/&gt;</code> element.</del>
[C20009]	The <code>@name</code> attribute of a <code>&lt;function/&gt;</code> child element of a <code>&lt;interface.c/&gt;</code> MUST be unique amongst the <code>&lt;function/&gt;</code> elements of that <code>&lt;interface.c/&gt;</code> .
[C20010]	The <code>@name</code> attribute of a <code>&lt;callbackFunction/&gt;</code> child element of a <code>&lt;interface.c/&gt;</code> MUST be unique amongst the <code>&lt;callbackFunction/&gt;</code> elements of that <code>&lt;interface.c/&gt;</code> .
[C20011]	<del>If the header file identified by the <code>@header</code> or <code>@callbackHeader</code> attribute of an <code>&lt;interface.c/&gt;</code> element defines the operations of the interface (callback interface) using message formats, then the <code>struct</code> defining the input message format MUST be identified using an <code>@input</code> attribute.</del>
[C20012]	<del>If the header file identified by the <code>@header</code> or <code>@callbackHeader</code> attribute of an <code>&lt;interface.c/&gt;</code> element defines the operations of the interface (callback interface) using message formats, then the <code>struct</code> defining the output message format MUST be identified using an <code>@input</code> attribute.</del>
[C20013]	The <code>@name</code> attribute of a <code>&lt;function/&gt;</code> child element of a <code>&lt;implementation.c/&gt;</code> MUST be unique amongst the <code>&lt;function/&gt;</code> elements of that <code>&lt;implementation.c/&gt;</code> .
[C20014][C20015]	<del>An SCA runtime MUST ensure that a stateless scoped implementation instance object is only ever dispatched on one thread at any one time. In addition, within the SCA lifecycle of an instance, an SCA runtime MUST only make a single invocation of one business method. An SCA runtime MUST NOT perform any synchronization of access to component implementations.</del>

Conformance ID	Description
[C20015]	<del>An SCA runtime MAY run multiple threads in a single composite-scoped implementation instance object and it MUST NOT perform any synchronization.</del>
[C20016]	The SCA runtime MAY use by-reference semantics when passing input parameters, return values or exceptions on calls to remotable services within the same system address space if both the service function implementation and the client are marked "allows pass by reference".
[C20017]	The SCA runtime MUST use by-value semantics when passing input parameters, return values and exceptions on calls to remotable services within the same system address space if the service function implementation is not marked "allows pass by reference" or the client is not marked "allows pass by reference".
[C30001]	An SCA implementation MAY support proxy functions.
[C40001]	An operation marked as oneWay is considered non-blocking and the SCA runtime MAY use a binding that buffers the requests to the function and sends them at some time after they are made.
[C50001]	Vendor defined reason codes SHOULD start at 101.
[C60002]	An SCA runtime MAY additionally provide a DataObject variant of this API for handling properties with complex XML types. The type of the value parameter in this variant is DATAOBJECT.
[C60003]	<del>A SCA runtime MAY provide the functions SCAService(), SCAOperation(), SCAMessageIn() and SCAMessageOut() to support C implementations in programs.</del>
[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]	<p><del>The return type and types of the parameters of a function of a local service interface MUST be one of:</del></p> <ul style="list-style-type: none"> <li><del>• Any of the C primitive types (for example, int, short, char). In this case the data will be passed by value as is normal for C.</del></li> <li><del>• Pointers to any of the C primitive types (for example, int *, short *, char *).</del></li> </ul> <p><del>DATAOBJECT. An SDO handle.</del> An SCA implementation MUST translate declarations to tokens as part of conversion to WSDL or compatibility testing.</p>

Conformance ID	Description
[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"> <li>• Any of the C primitive types (for example, int, short, char). This will be copied.</li> <li>• DATAOBJECT. An SDO handle. The SDO will be copied specified in Simple Content Binding and passed to the destination. 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.</li> <li>• 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.</li> </ul>
<del>[C90004]</del> [C80003]	<p>A C header file used to define an interface MUST:  <del>Declare</del> declare at least one function or message format struct</p>
<del>[C90002]</del> [C10001 1]	<p>A C header file used to define an interface MUST NOT use the following constructs:  <del>Macros</del>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.</p>
<del>[C100004]</del> [C10000 2]	<p><del>In</del>For components implemented in libraries, in the absence of customizations, an SCA implementation SHOULD<del>MUST</del> map each portType an operation name, with the first character converted to separate header file. An SCA implementation MAY use any sca-c:prefix binding declarations to control this mapping, 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.</p>
[C100002]	<p>For components implemented in libraries, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the function.</p>
[C100003]	<p>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.</p>
[C100004]	<p>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.</p>

Conformance ID	Description
[C100005][C100006]	For components implemented in a program, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with of the first character converted to lower case, and by a fault element to the operation name, with of the first character converted to upper case, to form a struct describing the request struct fault message content. If necessary, to avoid name collisions, an SCA implementation MUST MAY append "ResponseFault" to the request struct name to form of the response message element when mapping to the struct name.
[C100006]	In the absence of customizations, an SCA implementation MUST map the name of the message element referred to by a fault element to name of the struct describing the fault message content. If necessary, to avoid name collisions, an implementation MAY append "Fault" to the name of the message element when mapping to the struct name.
[C100007]	An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be configurable.
[C100008]	In the absence of customizations, an SCA implementation MUST map the header file name to the portType name. An implementation MAY append "PortType" to the header file name in the mapping to the portType name.
[C100009]	In the absence of customizations, an SCA implementation MUST map the function name to the operation name, stripping the portType name, if present, and any namespace prefix from the function name from the front of function name before mapping it to the operation name.
[C100010][C100011]	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. In the absence of customizations, an SCA implementation MUST map a parameter name, if present, to a part or global element component name. If the parameter does not have a name the SCA implementation MUST use argN as the part or global element child name.
[C100011]	In the absence of customizations, an SCA implementation MUST map the parameter name, if present, to the part or global element component name. If the parameter does not have a name the SCA implementation MUST use argN as the part or global element child name.
[C100012]	In the absence of customizations, an SCA implementation MUST map the return type to a part or global element child named "return".
[C100013][C100016]	Program based implementation SHOULD use the Document Literal style and encoding. 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.
[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.

Conformance ID	Description
[C100016]	An SCA implementation <del>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.</del>
[C100017]	An SCA implementation MUST map: <ul style="list-style-type: none"> <li>a function's return value as an <b>out</b> parameter.</li> <li>by-value and const parameters as <b>in</b> parameters.</li> <li>in the absence of customizations, pointer parameters as <b>in/out</b> parameters.</li> </ul>
[C100019]	For library-based service implementations, an SCA implementation MUST map <b>In</b> parameters as pass by-value <del>or const</del> and <b>In/Out</b> and <b>Out</b> parameters as pass via pointers.
[C100020][C100021]	<del>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 <b>In/Out</b> parameters and all <b>Out</b> parameters in the response message as pass by value. An SCA implementation MUST map simple types as defined in Table 9-1 and Table 9-2 by default.</del>
[C100021]	<del>An SCA implementation MUST map simple types as defined in Table 1 and Table 2 by default.</del>
[C100022]	An SCA implementation MAY map boolean to <b>_Bool</b> by default.
[C110001][C100023]	An SCA implementation MUST reject a composite file that does not conform to <a href="http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd</a> or <a href="http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd</a> . An SCA implementation MUST map a WSDL portType to a remotable C interface definition.
[C110002][C100024]	An SCA implementation MUST reject a componentType or constraining type file that does not conform to <a href="http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd</a> . An SCA implementation MUST map a C interface definition to WSDL as if it has a @WebService annotation with all default values.
[C110003][C110001]	An SCA implementation MUST reject a contribution file that does not conform to <del>An SCA implementation MUST reject a composite file that does not conform to</del> <a href="http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contributioninterface-c-1.1.xsd">http://docs.oasis-open.org/opencsa/sca/200903200912/sca-contributioninterface-c-1.1.xsd</a> ; or <a href="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</a> .
[C110004][C110002]	An SCA implementation MUST reject a WSDL file that does not conform to <del>An SCA implementation MUST reject a componentType file that does not conform to</del> <a href="http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200904/200912/sca-wsdlextinterface-c-1.1.xsd">http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200904/200912/sca-wsdlextinterface-c-1.1.xsd</a> ;
[C110003]	An SCA implementation MUST reject a contribution file that does not conform to <a href="http://docs.oasis-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</a> .
[C110004]	An SCA implementation MUST reject a WSDL file that does not conform to <a href="http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200901/sca-wsdlext-c-1.1.xsd">http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200901/sca-wsdlext-c-1.1.xsd</a> .

4750

Table F-1: SCA C Core Normative Statements

4751  
4752  
4753

## F.1 Program-Based Normative Statements Summary

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

<u>Conformance ID</u>	<u>Description</u>
[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.

4754 *Table F-2: SCA C Program-Based Normative Statements*

## F.2 Annotation Normative Statement Summary

4755  
4756 This section contains a list of normative statements related to source file annotations for this specification.

<u>Conformance ID</u>	<u>Description</u>
[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 <del>was</del> 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, <del>unless the exclude value of the @WebFunction annotation is true,</del> as if @OperationFunction was specified with the operationName value of the @WebFunction annotation used as the name value of the @OperationFunction 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, <del>unless the exclude value of the @WebOperation annotation is true,</del> as if @Operation was specified with the <del>struct as the input value, the</del> operationName value of the @WebOperation annotation used as the name value of the @Operation annotation <del>and,</del> the response value of the @WebOperation annotation used as the <del>output value</del> response value of the @Operation annotation and the exclude value of the @WebFunction annotation used as the exclude value of the @Operation annotation.
[CC0004][CA0006]	An SCA implementation MUST treat any instance of a @Interface annotation and without an explicit @WebService annotation as if a @WebService annotation with a name value equal to the name value of the @Interface annotation and no other parameters was specified. <del>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.</del>
[CC0002][CA0007]	An SCA implementation MUST treat a function annotated with an @Operation annotation and without an explicit @WebFunction annotation as if a @WebFunction annotation with with an operationName value equal to the name value of the <del>@Operation annotation and no other parameters was specified.</del> An SCA implementation MUST ensure that all variables in a component implementation with the same name and annotated with @Property have the same type.
[CC0003][CC0001]	An SCA implementation MUST treat <del>an @Operation</del> any instance of a @Remotable annotation and without an explicit <del>@WebOperation</del> @WebService annotation as if a <del>@WebOperation</del> @WebService annotation with with an operationName a name value equal to the name value of the <del>@Operation</del> @Interface annotation, a response value equal to the output value of the <del>@Operation annotation and if specified,</del> and no other parameters was specified <del>is applied to the struct identified as the input value of the @Operation annotation.</del>
[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 <a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a> <del>http://www.w3.org/2001/XMLSchema</del> 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.

<u>Conformance ID</u>	<u>Description</u>
[CC0007]	The value of the type property of a @WebResult annotation MUST be one of the simpleTypes defined in namespace <a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a> .
<del>[CC0008]</del> [CC0009]	<del>If a @WebService does not have a portName element, an SCA implementation MUST use the value associated with the name element, suffixed with "Port".</del> The value of the paramName of a @WebParam annotation MUST be the name of a parameter of the function the annotation is applied to.

4757 *Table F-3: SCA C Annotation Normative Statements*

### 4758 **F.3 WSDL Extension Normative Statement Summary**

4759 This section contains a list of normative statements related to WSDL extensions for this specification.

<u>Conformance ID</u>	<u>Description</u>
[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 valid 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. <a href="#">Table D-1 is a list of JAX-WS WSDL extensions that MAY be interpreted, and their corresponding SCA WSDL extension.</a>

4760 *Table F-4: SCA C WSDL Extension Normative Statements*

### 4761 **E.4F.4 JAX-WS Conformance Normative Statements**

4762 The JAX-WS 2.1 specification [JAXWS21] defines conformance normative statements for various  
 4763 requirements defined by that specification. ~~The following table~~ [Table F-5](#) outlines those  
 4764 conformance normative statements, ~~and describes whether the conformance statement applies which~~  
 4765 apply to the WSDL binding mapping described in this specification.

<u>Section Number</u>	<u>Conformance Statement Point</u>	<u>Notes</u>	<u>Conformance ID</u>
<u>2.1</u>	WSDL 1.1 support	[A]	[CF0001]



<u>SectionNumber</u>	<u>Conformance StatementPoint</u>	<u>Notes</u>	<u>Conformance ID</u>
<u>2.2</u>	Customization required	[CD0001] The reference to the JAX-WS binding language <del>are</del> is treated as a reference to the C WSDL extensions defined in C WSDL Mapping Extensions	
<u>2.3</u>	Annotations on generated classes		[CF0002]
<u>2.45</u>	WSDL and XML Schema import directives		[CF0003]
<u>2.4.46</u>	Optional WSDL extensions		[CF0004]
<u>2.27</u>	SEI naming	[C100001]	
<u>2.28</u>	javax.jws.WebService required	[B] References to javax.jws.WebService in the conformance statement are treated as the C annotation @WebService.	[CF0005]
<u>2.310</u>	Method naming	[C100002] and [C100005]	
<u>2.311</u>	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]
<u>2.312</u>	Transmission primitive support		[CF0007]
<u>2.313</u>	Using javax.jws.OneWay	[A], [B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0008]
<u>2.3.414</u>	Using javax.jws.SOAPBinding	[A], [B] References to javax.jws.SOAPBinding in the conformance statement are treated as the C annotation @SOAPBinding.	[CF0009]
<u>2.3.415</u>	Using javax.jws.WebParam	[A], [B] References to javax.jws.WebParam in the conformance statement are treated as the C annotation @WebParam.	[CF0010]
<u>2.3.416</u>	Using javax.jws.WebResult	[A], [B] References to javax.jws.WebResult in the conformance statement are treated as the C annotation @WebResult.	[CF0011]

<u>SectionNumber</u>	<u>Conformance StatementPoint</u>	<u>Notes</u>	<u>Conformance ID</u>
<a href="#">2.3.1.418</a>	Non-wrapped parameter naming	[C100003]	
<a href="#">2.3.1.219</a>	Default mapping mode		[CF0012]
<a href="#">2.3.1.220</a>	Disabling wrapper style	[B] References to jaxws:enableWrapperStyle in the conformance statement are treated as the C annotation sca-c:enableWrapperStyle.	[CF0013]
<a href="#">2.3.1.221</a>	Wrapped parameter naming	[C100004]	
<a href="#">2.3.1.222</a>	Parameter name clash	[A]	[CF0014]
<a href="#">2.538</a>	javax.xml.ws.WebFault required	[B] References to javax.jws.WebFault in the conformance statement are treated as the C annotation @WebFault.	[CF0015]
<a href="#">2.539</a>	Exception naming	[C100006]	
<a href="#">2.540</a>	Fault equivalence	[A] References to fault exception classes are treated as references to fault message structs.	[CF0016]
<a href="#">2.642</a>	Required WSDL extensions	MIME Binding not necessary	[CF0018]
<a href="#">2.6.143</a>	Unbound message parts	[A]	[CF0019]
<a href="#">2.6.2.144</a>	Duplicate headers in binding		[CF0020]
<a href="#">2.6.2.145</a>	Duplicate headers in message		[CF0021]
<a href="#">3.1</a>	WSDL 1.1 support	[A]	[CF0022]
<a href="#">3.2</a>	Standard annotations	[A] [CC0005]	
<a href="#">3.43</a>	Java identifier mapping	[A]	[CF0023]
<a href="#">3.26</a>	WSDL and XML Schema import directives		[CF0024]
<a href="#">3.48</a>	portType naming	[C100008]	
<a href="#">3.511</a>	Operation naming	[C100009] and [C100010]	

<u>SectionNumber</u>	<u>Conformance StatementPoint</u>	<u>Notes</u>	<u>Conformance ID</u>
<u>3.5.412</u>	One-way mapping	[B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0025]
<u>3.5.413</u>	One-way mapping errors		[CF0026]
<u>3.6.415</u>	Parameter classification	[C100017]	
<u>3.6.416</u>	Parameter naming	[C100011] and [C100014]	
<u>3.6.417</u>	Result naming	[C100012]	
<u>3.6.418</u>	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]
<u>3.724</u>	Exception naming	[CC0004]	
<u>3.827</u>	Binding selection	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CF0029]
<u>3.4028</u>	SOAP binding support	[A]	[CF0030]
<u>3.40.429</u>	SOAP binding style required		[CF0031]
<u>3.4431</u>	Port selection		[CF0032]
<u>3.4432</u>	Port binding	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY].	[CF0033]

4766 [A] All references to Java in the conformance statementpoint are treated as references to C.

4767 [B] Annotation generation is only necessary if annotations are supported by an SCA implementation.

### 4768 **B.1.1 Ignored Conformance Statements**

<u>Section</u>	<u>Conformance Statement</u>	<u>Notes</u>
<u>2.1</u>	<u>Definitions mapping</u>	
<u>2.2</u>	<u>javax.xml.bind.XmlSeeAlso required</u>	
<u>2.3.1</u>	<u>use of JAXB annotations</u>	
<u>2.3.1.2</u>	<u>Using javax.xml.ws.RequestWrapper</u>	
<u>2.3.1.2</u>	<u>Using javax.xml.ws.ResponseWrapper</u>	

2.3.3	Use of Holder	
2.3.4	Asynchronous mapping required	
2.3.4	Asynchronous mapping option	
2.3.4.2	Asynchronous method naming	
2.3.4.2	Asynchronous parameter naming	
2.3.4.2	Failed method invocation	
2.3.4.4	Response bean naming	
2.3.4.5	Asynchronous fault reporting	
2.3.4.5	Asynchronous fault cause	
2.4	JAXB class mapping	
2.4	JAXB customization use	
2.4	JAXB customization clash	
2.4.1	javax.xml.ws.wsaddressing.W3CEndpointReference	
2.5	Fault Equivalence	
2.6.3.1	Use of MIME type information	
2.6.3.1	MIME type mismatch	
2.6.3.1	MIME part identification	
2.7	Service superclass required	
2.7	Service class naming	
2.7	javax.xml.ws.WebServiceClient required	
2.7	Default constructor required	
2.7	2 argument constructor required	
2.7	Failed getPort Method	
2.7	javax.xml.ws.WebEndpoint required	
3.1.1	Method name disambiguation	
3.2	Package name mapping	
3.3	Class mapping	
3.4.1	Inheritance flattening	
3.4.1	Inherited interface mapping	
3.6	use of JAXB annotations	
3.6.2.1	Default wrapper bean names	
3.6.2.1	Default wrapper bean package	
3.6.2.3	Null Values in rpc/literal	

3.7	<del>java.lang.RuntimeExceptions and java.rmi.RemoteExceptions</del>	
3.7	<del>Fault bean name clash</del>	
3.11	<del>Service creation</del>	

4769

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

4770

### **F.4.1 Ignored Normative Statments**

<b><u>Number</u></b>	<b><u>Conformance Point</u></b>
<a href="#">2.4</a>	<a href="#">Definitions mapping</a>
<a href="#">2.9</a>	<a href="#">javax.xml.bind.XmlSeeAlso required</a>
<a href="#">2.17</a>	<a href="#">use of JAXB annotations</a>
<a href="#">2.23</a>	<a href="#">Using javax.xml.ws.RequestWrapper</a>
<a href="#">2.24</a>	<a href="#">Using javax.xml.ws.ResponseWrapper</a>
<a href="#">2.25</a>	<a href="#">Use of Holder</a>
<a href="#">2.26</a>	<a href="#">Asynchronous mapping required</a>
<a href="#">2.27</a>	<a href="#">Asynchronous mapping option</a>
<a href="#">2.28</a>	<a href="#">Asynchronous method naming</a>
<a href="#">2.29</a>	<a href="#">Asynchronous parameter naming</a>
<a href="#">2.30</a>	<a href="#">Failed method invocation</a>
<a href="#">2.31</a>	<a href="#">Response bean naming</a>
<a href="#">2.32</a>	<a href="#">Asynchronous fault reporting</a>
<a href="#">2.33</a>	<a href="#">Asynchronous fault cause</a>
<a href="#">2.34</a>	<a href="#">JAXB class mapping</a>
<a href="#">2.35</a>	<a href="#">JAXB customization use</a>
<a href="#">2.36</a>	<a href="#">JAXB customization clash</a>
<a href="#">2.37</a>	<a href="#">javax.xml.ws.wsaddressing.W3CEndpointReference</a>
<a href="#">2.41</a>	<a href="#">Fault Equivalence</a>
<a href="#">2.46</a>	<a href="#">Use of MIME type information</a>
<a href="#">2.47</a>	<a href="#">MIME type mismatch</a>
<a href="#">2.48</a>	<a href="#">MIME part identification</a>
<a href="#">2.49</a>	<a href="#">Service superclass required</a>
<a href="#">2.50</a>	<a href="#">Service class naming</a>
<a href="#">2.51</a>	<a href="#">javax.xml.ws.WebServiceClient required</a>

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

4771

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

4772

---

## **FG Migration**

4773

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

4774

4775

4776

### **F.1G.1 Implementation.c attributes**

4777

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

4778

### **F.2G.2 SCALocate and SCALocateMultiple**

4779

SCALocate() and SCALocateMultiple() have been renamed to SCAGetReference()

4780

SCAGetReferences() respectively.

4781

---

## **GH Acknowledgements**

4782 The following individuals have participated in the creation of this specification and are gratefully  
4783 acknowledged:

4784 **Participants:**

4785

<b>Participant Name</b>	<b>Affiliation</b>
Bryan Aupperle	IBM
Andrew Borley	IBM
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
David Haney	Individual
Mark Little	Red Hat
Jeff Mischkinsky	Oracle Corporation
Peter Robbins	IBM



4786

---

## HI Revision History

4787

[optional; should not be included in OASIS Standards]

4788

**Revision**

**Date**

**Editor**

**Changes Made**

•

4789