



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

Committee Draft 03 **Revision 2**

30 April 2009

Deleted: / Public Review Draft 01

Deleted: 19 March

Specification URIs:

This Version:

<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd03-rev2.html>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd03-rev2.doc>
<http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd03-rev2.pdf> (Authoritative)

Field Code Changed

Field Code Changed

Field Code Changed

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)

Field Code Changed

Field Code Changed

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

Abstract:

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

Deleted: Specification

Deleted: 19 March

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/>.

Deleted: Specification

Deleted: 19 March

Notices

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

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

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

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

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

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

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

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

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

Deleted: Specification

Deleted: 19 March

Table of Contents

1	Introduction	8
1.1	Terminology	8
1.2	Normative References	8
1.3	Conventions	9
1.3.1	Naming Conventions	9
1.3.2	Typographic Conventions	9
2	Basic Component Implementation Model	10
2.1	Implementing a Service	10
2.1.1	Implementing a Remotable Service	11
2.1.2	AllowsPassByReference	11
2.1.3	Implementing a Local Service	12
2.2	Component and Implementation Scopes	12
2.2.1	Stateless scope	13
2.2.2	Composite scope	13
2.3	Implementing a Configuration Property	13
2.4	Component Type and Component	14
2.4.1	Interface.c	15
2.4.2	Function and CallbackFunction	16
2.4.3	Implementation.c	17
2.4.4	Implementation Function	18
2.5	Implementing a Service with a Program	18
3	Basic Client Model	20
3.1	Accessing Services from Component Implementations	20
3.2	Accessing Services from non-SCA component implementations	21
3.3	Calling Service Operations	21
3.3.1	Proxy Functions	21
3.4	Long Running Request-Response Operations	22
3.4.1	Asynchronous Invocation	22
3.4.2	Polling Invocation	24
3.4.3	Synchronous Invocation	25
4	Asynchronous Programming	26
4.1	Non-blocking Calls	26
4.2	Callbacks	26
4.2.1	Using Callbacks	27
4.2.2	Callback Instance Management	28
4.2.3	Implementing Multiple Bidirectional Interfaces	28
5	Error Handling	29
6	C API	30
6.1	SCA Programming Interface	30
6.1.1	SCAGetReference	32
6.1.2	SCAGetReferences	33
6.1.3	SCAInvoke	33
6.1.4	SCAProperty<T>	34

Deleted: Specification

Deleted: 19 March

6.1.5	SCAGetReplyMessage	35
6.1.6	SCAGetFaultMessage	35
6.1.7	SCASetFaultMessage	36
6.1.8	SCASelf	37
6.1.9	SCAGetCallback	37
6.1.10	SCAReleaseCallback	37
6.1.11	SCAInvokeAsync	38
6.1.12	SCAInvokePoll	38
6.1.13	SCACheckResponse	39
6.1.14	SCACancelInvoke	39
6.1.15	SCAEntryPoint	39
6.2	Program-Based Implementation Support	40
6.2.1	SCAService	40
6.2.2	SCAOperation	40
6.2.3	SCAMessageIn	41
6.2.4	SCAMessageOut	41
7	C Contributions	42
7.1	Executable files	42
7.1.1	Executable in contribution	42
7.1.2	Executable shared with other contribution(s) (Export)	42
7.1.3	Executable outside of contribution (Import)	43
7.2	componentType files	44
7.3	C Contribution Extensions	44
7.3.1	Export.c	44
7.3.2	Import.c	45
8	Types Supported in Service Interfaces	46
8.1	Local service	46
8.2	Remotable service	46
9	Restrictions on C header files	47
10	WSDL to C and C to WSDL Mapping	48
10.1	Interpretations for WSDL to C Mapping	48
10.1.1	Definitions	48
10.1.2	PortType	48
10.1.3	Operations	49
10.1.4	Types	50
10.1.5	Fault	50
10.1.6	Service and Port	50
10.1.7	XML Names	50
10.2	Interpretations for C to WSDL Mapping	50
10.2.1	Package	50
10.2.2	Class	50
10.2.3	Interface	50
10.2.4	Method	51
10.2.5	Method Parameters and Return Type	51
10.2.6	Service Specific Exception	52

Deleted: Specification

Deleted: 19 March

10.2.7	Generics	52
10.2.8	Service and Ports	52
10.3	Data Binding.....	53
10.3.1	Simple Content Binding.....	53
10.3.2	Complex Content Binding.....	58
11	Conformance	63
11.1	Conformance Targets	63
11.2	SCA Implementations	63
11.3	SCA Documents.....	64
11.4	C Files	64
11.5	WSDL Files	64
A	C SCA Annotations.....	65
A.1	Application of Annotations to C Program Elements	65
A.2	Interface Header Annotations.....	65
A.2.1	@Interface	65
A.2.2	@Operation	66
A.2.3	@Remotable	67
A.2.4	@Callback	68
A.2.5	@OneWay.....	68
A.3	Implementation Annotations	69
A.3.1	@ComponentType	69
A.3.2	@Service	69
A.3.3	@Reference	70
A.3.4	@Property	71
A.3.5	@Scope.....	72
A.3.6	@Init.....	72
A.3.7	@Destroy	73
A.3.8	@EagerInit	73
A.3.9	@AllowsPassByReference	74
A.4	Base Annotation Grammar	74
B	C SCA Policy Annotations	76
B.1	General Intent Annotations	76
B.2	Specific Intent Annotations	77
B.2.1	Security Interaction.....	78
B.2.2	Security Implementation.....	78
B.2.3	Reliable Messaging	78
B.2.4	Transactions.....	79
B.2.5	Miscellaneous.....	79
B.3	Policy Set Annotations.....	79
B.4	Policy Annotation Grammar Additions.....	80
B.5	Annotation Constants	80
C	C WSDL Annotations.....	81
C.1	Interface Header Annotations.....	81
C.1.1	@WebService	81
C.1.2	@WebFunction	82

Deleted: Specification

Deleted: 19 March

C.1.3 @WebOperation	84
C.1.4 @OneWay	86
C.1.5 @WebParam	87
C.1.6 @WebResult	89
C.1.7 @SOAPBinding	91
C.1.8 @WebFault	92
C.1.9 @WebThrows	94
D C WSDL Mapping Extensions	96
D.1 <sca-c:bindings>	96
D.2 <sca-c:prefix>	96
D.3 <sca-c:enableWrapperStyle>	98
D.4 <sca-c:function>	99
D.5 <sca-c:struct>	100
D.6 <sca-c:parameter>	102
D.7 JAX-WS WSDL Extensions	104
D.8 WSDL Extensions Schema	104
E XML Schemas	106
E.1 sca-interface-c-1.1.xsd	106
E.2 sca-implementation-c-1.1.xsd	106
E.3 sca-contribution-c-1.1.xsd	107
F Conformance Points	109
F.1 Annotation Conformance Points	112
F.2 WSDL Extension Conformance Points	114
F.3 JAX-WS Conformance Points	114
F.3.1 Ignored Conformance Points	117
G Migration	119
G.1 Implementation.c attributes	119
G.2 SCALocate and SCALocateMultiple	119
H Acknowledgements	120
I Revision History	121

Deleted: Specification

Deleted: 19 March

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/200903"	Defined by the SCA specifications
sca-c	"http://docs.oasis-open.org/ns/opencsa/sca-c-c/200901"	

1.2 Normative References

- [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>
- [ASSEMBLY] OASIS Committee Draft 03, *Service Component Architecture Assembly Model Specification Version 1.1*, March 2009. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.pdf>
- [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.pdf>
- [SDO21] OSOA, *Service Data Objects For C Specification*, September 2007. http://www.osoa.org/download/attachments/36/SDO_Specification_C_V2.1.pdf
- [WSDL11] World Wide Web Consortium, *Web Service Description Language (WSDL)*, March 2001. <http://www.w3.org/TR/wsdl>
- [XSD] World Wide Web Consortium, *XML Schema Part 2: Datatypes Second Edition*, October 2004. <http://www.w3.org/TR/xmlschema-2/>

Deleted: .

Deleted: Specification

Deleted: 19 March

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

38 **1.3 Conventions**

39 **1.3.1 Naming Conventions**

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

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

51 **1.3.2 Typographic Conventions**

52 This specification follows some typographic conventions for some specific constructs

- 53 • XML attributes are identified in text as @attribute
54 • Language identifiers used in text are in `courier`
55 • Literals in text are in *italics*

Deleted: Specification
Deleted: 19 March

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]**

Deleted: A C implementation MUST implement all of the operation(s) of the service interface(s) of its componentType.

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

Service interface.

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

Implementation.

```
#include "LoanService.h"

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

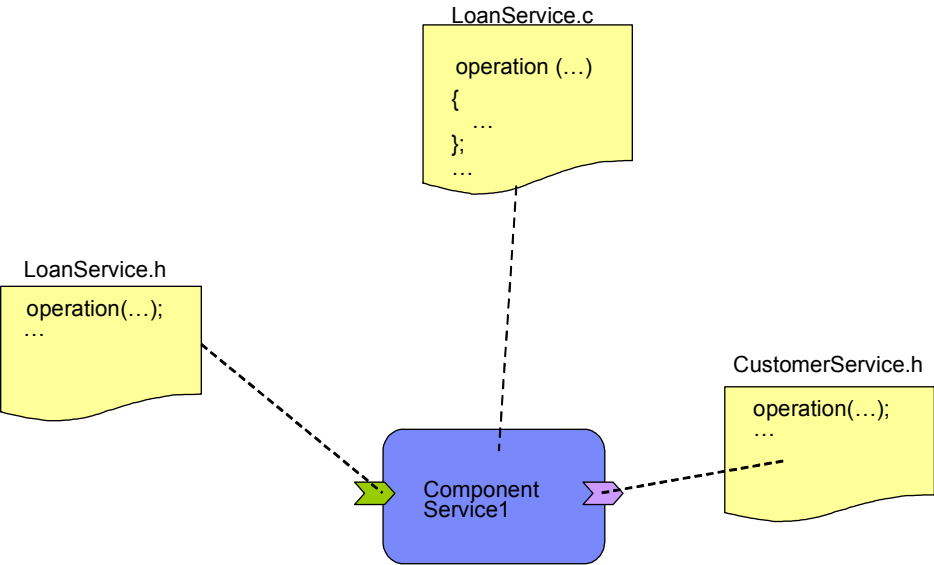
The following snippet shows the component type for this component implementation.

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

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

Deleted: Specification

Deleted: 19 March



103 **2.1.1 Implementing a Remotable Service**

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

```
108 <?xml version="1.0" encoding="ASCII"?>  
109 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
110   <service name="LoanService">  
111     <interface.c header="LoanService.h" remotable="true"/>  
112   </service>  
113 </componentType>
```

114 **2.1.2 AllowsPassByReference**

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

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

Deleted: Specification
Deleted: 19 March

131 copying by-reference values when a remotable service is called locally within the same operating system
132 address space. See Implementation.c and Implementation Function for a description of the
133 @allowsPassByReference attribute and how it is used.

134 2.1.2.1 Marking services and references as “allows pass by reference”

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

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

142
143 Marking a client as “allows pass by reference” asserts that the client observe the following restrictions for
144 all references’ functions:

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

150 2.1.2.2 Using “allows pass by reference” to optimize remotable calls

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

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

Deleted: 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”.

159 2.1.3 Implementing a Local Service

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

161 2.2 Component and Implementation Scopes

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

166
167 Scopes are specified using the @scope attribute of the *implementation.c* element.

168
169 When a scope is not specified in an implementation file, the SCA runtime will interpret the implementation
170 scope as **stateless**.

171
172 An SCA runtime MUST support these scopes; **stateless** and **composite**. Additional scopes MAY be
173 provided by SCA runtimes.[C20003]

Formatted: Pattern: Clear (Gray-20%)

Deleted: An SCA runtime MUST support these scopes; **stateless** and **composite**. Additional scopes MAY be provided by SCA runtimes.

Deleted: Specification

Deleted: 19 March

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

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

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

Formatted: Pattern: Clear (Yellow)

Deleted: A C implementation MUST only designate functions with no arguments and a void return type as lifecycle functions.

2.2.1 Stateless scope

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

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

Formatted: Pattern: Clear (Light Yellow)

Deleted: 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.

Lifecycle functions are not defined for stateless implementations.

2.2.2 Composite scope

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

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

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

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

2.3 Implementing a Configuration Property

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

Deleted: Specification

Deleted: 19 March

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

```
222 #include "SCA.h"
223
224 void clientFunction()
225 {
226
227     ...
228
229     int32_t loanRating;
230     int compCode, reason;
231
232     ...
233
234     SCAPropertyInt(L"maxLoanValue", &loanRating, &compCode, &reason);
235
236     ...
237
238 }
```

239
240 If the property is many valued, an array of the appropriate type is used as the second parameter, and the
241 third parameter would point to an int that would receive the number of values. The type for the property
242 SHOULD NOT allow more values to be defined than the size of the array in the implementation.

243

244 2.4 Component Type and Component

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

249

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

252

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

254

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

257

258 Implementation.

```
259 #include "LoanService.h"
260
261 char approveLoan(long customerNumber, long loanAmount)
262 {
263     ...
264 }
265
```

266

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

268

```
269 <?xml version="1.0" encoding="ASCII"?>
270 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
271   <service name="LoanService">
272     <interface.c header="LoanService.h" />
273   </service>
```

Deleted: Specification

Deleted: 19 March

```
</componentType>
```

The following snippet shows the component using the implementation.

```
<?xml version="1.0" encoding="ASCII"?>
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  name="LoanComposite" >
  ...
  <component name="LoanService">
    <implementation.c module="loan" componentType="LoanService" />
  </component>
  ...
</composite>
```

2.4.1 Interface.c

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- interface.c schema snippet -->
<interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  header="string" remotable="boolean"? callbackHeader="string"?
  requires="listOfQNames"? policySets="listOfQNames"? >
  ...
  <function ... />*
  <callbackFunction ... />*
</interface.c>
```

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

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

Formatted: Bullets and Numbering

Formatted: Default Paragraph Font

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

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

Deleted: Specification

Deleted: 19 March

2.4.2 Function and CallbackFunction

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

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

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- interface.c schema snippet -->
<interface.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"... >

  <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
    oneWay="Boolean"?
    input="NCName"? output="NCName"? />*
  <callbackFunction name="NCName" requires="listOfQNames"?
    policySets="listOfQNames"? oneWay="Boolean"? input="NCName"?
    output="NCName"? />*
</interface.c>
```

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

- **name : NCName (1..1)** – name of the function being decorated or included in the interface. The @name attribute of a <function/> child element of a <interface.c/> MUST be unique amongst the <function/> elements of that <interface.c/>. [C20009]
The @name attribute of a <callbackFunction/> child element of a <interface.c/> MUST be unique amongst the <callbackFunction/> elements of that <interface.c/>. [C20010]
- **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute.
- **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.
- **oneWay : boolean (0..1)** – see Non-blocking Calls
- **input : NCNAME (0..1)** – If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using message formats, then the struct defining the input message format MUST be identified using an @input attribute. [C20011] (See Implementing a Service with a Program)
- **output : NCNAME (0..1)** – If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using

Formatted: Pattern: Clear (Light Yellow)

Deleted: If the header file identified by the @header attribute of an <interface.c/> element contains function declarations that are not operations of the interface, then the functions that define operations of the interface MUST be identified using <function/> child elements of the <interface.c/> element.

Formatted: Pattern: Clear (Turquoise)

Deleted: If the header file identified by the @callbackHeader attribute of an <interface.c/> element contains function declarations that are not operations of the callback interface, then the functions that define operations of the callback interface MUST be identified using <callbackFunction/> child elements of the <interface.c/> element.

Formatted: Pattern: Clear (Pale Blue)

Deleted: If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using message formats, then all functions of the interface (callback interface) MUST be identified using <function/> (<callbackFunction/>) child elements of the <interface.c/> element.

Deleted: ¶

Formatted: Pattern: Clear (Light Green)

Deleted: The @name attribute of a <function/> child element of a <interface.c/> MUST be unique amongst the <function/> elements of that <interface.c/>.

Formatted: Emphasis

Deleted: The @name attribute of a <callbackFunction/> child element of a <interface.c/> MUST be unique amongst the <callbackFunction/> elements of that <interface.c/>.

Formatted: Bullets and Numbering

Deleted: <#>requires : listOfQNames (0..1) – list of intents [POLICY] needed by this function.¶

Formatted: Pattern: Clear (Tan)

Deleted: If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using message formats, then the struct defining the input message format MUST be identified using an @input attribute. [C20011] (See Implementing a Service with a Program)

Formatted: Pattern: Clear (Yellow)

Deleted: Specification

Deleted: 19 March

message formats, then the `struct` defining the output message format MUST be identified using an `@output` attribute. [C20012]

Deleted: If the header file identified by the `@header` or `@callbackHeader` attribute of an `<interface.c>` element defines the operations of the interface (callback interface) using message formats, then the `struct` defining the output message format MUST be identified using an `@input` attribute.

2.4.3 Implementation.c

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- implementation.c schema snippet -->
<implementation.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  module="NCName" library="boolean"? path="string"?
  scope="scope"? componentType="string" allowsPassByReference="Boolean"?
  eagerInit="Boolean"? init="Boolean"? destroy="Boolean"?
  requires="listOfQNames"? policySets="listOfQNames"? >

  <function ... />*
</implementation.c>
```

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

- **module : NCName (1..1)** – name of the binary executable for the service component. This is the root name of the module.
- **library : boolean (0..1)** – indicates whether the service is implemented as a library or a program. The default is library. See Implementing a Service with a Program
- **path : string (0..1)** – path to the module which is either relative to the root of the contribution containing the composite or is prefixed with a contribution import name and is relative to the root of the import. See [C Contributions](#).
- **scope : CImplementationScope (0..1)** – indicates the scope of the component implementation. The default is stateless. See Component and Implementation Scopes
- **componentType : string (1..1)** – name of the componentType file. A “.componentType” extension will be appended. A path to the componentType file which is relative to the root of the contribution containing the composite or is prefixed with a contribution import name and is relative to the root of the import (see [C Contributions](#)) can be included.
- **allowsPassByReference : boolean (0..1)** – indicates the implementation allows pass by reference data exchange semantics on calls to it or from it. These semantics apply to all services provided by and references used by an implementation. See AllowsPassByReference
- **eagerInit : boolean (0..1)** – indicates a composite scoped implementation is to be initialized when it is loaded. See Composite scope
- **init : boolean (0..1)** – indicates program is to be called with an initialize flag to initialize the implementation. See Component and Implementation Scopes
- **destroy : boolean (0..1)** – indicates is to be called with a destroy flag to cleanup the implementation. See Component and Implementation Scopes
- **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute. If intents are specified at both the implementation and function level, the effective intents for the function is determined by merging the combined intents from the function with the combined intents for the implementation according to the Policy Framework rules for merging intents within a structural hierarchy, with the function at the lower level and the implementation at the higher level.
- **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.

Deleted: C Contributions

Deleted: C Contributions

Formatted: Bullets and Numbering

Deleted: Specification

Deleted: 19 March

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

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

2.4.4 Implementation Function

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

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- ImplementationFunction schema snippet -->
<implementation.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"... >

  <function name="NCName" requires="listOfQNames"? policySets="listOfQNames"?
    allowsPassByReference="Boolean"? init="Boolean"?
    destroy="Boolean"? />*

</implementation.c>
```

Deleted: ¶

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

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

Formatted: Pattern: Clear (Gray-10%)

Deleted: The @name attribute of a <function/> child element of a <implementation.c/> MUST be unique amongst the <function/> elements of that <implementation.c/>.

Formatted: Bullets and Numbering

Deleted: <#>requires : **listOfQNames (0..1)** – list of intents [POLICY] needed by this function.¶

2.5 Implementing a Service with a Program

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

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

Since a program can implement multiple services and often will implement multiple operations, the program has to query the runtime to determine which service and operation caused the program to be invoked. This is done using **SCAService()** and **SCAOperation()**. Once the specific service and

Deleted: Specification

Deleted: 19 March

474 operation is known, the proper input message can be retrieved using `SCAMessageIn()`. Once the logic
475 of the operation is finished `SCAMessageOut()` is used to provide the return data to the runtime to be
476 marshalled.

477
478 Since a program does not have a specific prototype for each operation of each service it implements, a C
479 interface definition for the service identifies the operation names and the input and output message
480 formats using functions elements, with input and output attributes, in an *interface.c* element. Alternatively,
481 an external interface definition, such as a WSDL document, is used to describe the operations and
482 message formats.

483
484 The following shows a program implementing a service using these support functions.

```
485  
486 #include "SCA.h"  
487 #include "myInterface.h"  
488 main () {  
489     wchar_t myService [255];  
490     wchar_t myOperation [255];  
491     int compCode, reason;  
492     struct FirstInputMsg myFirstIn;  
493     struct FirstOutputMsg myFirstOut;  
494  
495  
496     SCAService(myService, &compCode, &reason);  
497  
498     SCAOperation(myOperation, &compCode, &reason);  
499  
500     if (wcscmp(myOperation, L"myFirstOperation")==0) {  
501         SCAMessageIn(myService, myOperation,  
502                     sizeof(struct FirstInputMsg), (void *)&myFirstIn,  
503                     &compCode, &reason);  
504         ...  
505         SCAMessageOut(myService, myOperation,  
506                      sizeof(struct FirstOutputMsg), (void *)&myFirstOut,  
507                      &compCode, &reason);  
508     }  
509     else  
510     {  
511         ...  
512     }  
513 }
```

Deleted: wcsncmp

Deleted: Specification

Deleted: 19 March

3 Basic Client Model

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

3.1 Accessing Services from Component Implementations

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

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

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

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

```
#include "SCA.h"

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

    ...
    SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
    SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),
              (void *)&custNum, sizeof(rating), (void *)&rating,
              &compCode, &reason);
}
```

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

```
SCAREF *tokens;
int num_targets;
...
myFunction(...) {
    int compCode, reason;
    ...
    SCAGetReferences(L"myReference", &tokens, &num_targets, &compCode,
                    &reason);
    for (i = 0; i < num_targets; i++)
    {
        SCAInvoke(tokens[i], L"myOperation", sizeof(inputMsg),
                  (void *)&inputMsg, 0, NULL, &compCode, &reason);
    };
};
```

Deleted: Specification

Deleted: 19 March

565

566 **3.2 Accessing Services from non-SCA component implementations**

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

569

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

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

```
#include "SCA.h"

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

    SCAEntryPoint(L"customerService", L"http://example.com/mydomain",
        &serviceToken, &compCode, &reason);
    SCAInvoke(serviceToken, L"getCreditRating", sizeof(custNum),
        (void *)&custNum, sizeof(rating), (void *)&rating,
        &compCode, &reason);
}
```

587

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

590

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

- 592
- componentName/serviceName/bindingName

593 **3.3 Calling Service Operations**

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

596

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

600 **3.3.1 Proxy Functions**

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

606

607 For SCA, proxy functions have the form:

608

609

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

610 where:

- 611
- **<functionName>** is the name of interface function

Deleted: Specification

Deleted: 19 March

- 612 • **<functionParameters>** are the parameters of the interface function
613 • **<functionReturn>** is the return type of the interface function
614

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

- 616 • `ENOENT` if a remote service is unavailable
617 • `EFAULT` if a fault is returned by the operation
618

619 The following shows a sample of using a proxy function.

```
620
621 #include "SCA.h"
622
623 void clientFunction()
624 {
625
626     SCAREF serviceToken;
627     int compCode, reason;
628     long custNum = 1234;
629     short rating;
630
631     ...
632     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
633     errno = 0;
634     rating = SCA_getCreditRating(serviceToken, custNum);
635     if (errno) {
636         /* handle error or fault */
637     }
638     else {
639         ...
640     }
641 }
642
```

643
644 An SCA implementation MAY support proxy functions. [C30001]

645 **3.4 Long Running Request-Response Operations**

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

655 **3.4.1 Asynchronous Invocation**

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

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

Deleted: Specification
Deleted: 19 March

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

664

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

668

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

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

671 the asynchronous invocation style includes a proxy function

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

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

674

675 The following shows a sample of how the asynchronous invocation style is used in a C component
676 implementation.

677

```
678 #include "SCA.h"
679 #include "TravelService.h"
680
681 SCAREF serviceToken;
682 int compCode, reason;
683
684 void makeReservationsHandler(short rspType)
685 {
686     struct confirmationData cd;
687     wchar_t *fault, *faultDetails;
688
689     if (rspType == SCA_REPLY_MESSAGE {
690         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
691         ...
692     }
693     else {
694         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
695                             &faultDetails, &compCode, &reason);
696         if (wscmp(*fault, L"noFlight") {
697             ...
698         }
699         else {
700             ...
701         }
702     }
703
704     return;
705 }
706
707 void clientFunction()
708 {
709     struct itineraryData id;
710     ...
711
712     void (*ah)(short) = &makeReservationsHandler;
713
714     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
715
716
717
```

Deleted: wstrcmp

Deleted: Specification

Deleted: 19 March

```

718     SCAInvokeAsync(serviceToken, L"makeReservations", sizeof(itineraryData),
719                   ah, &compCode, &reason);
720
721     return;
722 }

```

723 3.4.2 Polling Invocation

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

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

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

731 the polling invocation style includes a proxy function

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

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

734
735 The following shows a sample of how the polling invocation style is used in a C component implementation.

```

737
738 #include "SCA.h"
739 #include "TravelService.h"
740
741 void pollingClientFunction()
742 {
743     SCAREF serviceToken;
744     int compCode, reason;
745     short rspType;
746
747     struct itineraryData id;
748     struct confirmationData cd;
749     wchar_t *fault, *faultDetails;
750
751     ...
752
753     SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);
754
755     SCAInvokePoll(serviceToken, L"makeReservations", sizeof(itineraryData),
756                  &compCode, &reason);
757
758     SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
759     while (!rspType) {
760         // do something, then wait for some time...
761         SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
762     }
763     if (rspType == SCA_REPLY_MESSAGE {
764         SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
765         ...
766     }
767     else {
768         SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
769                           &faultDetails, &compCode, &reason);
770         if (wcsncmp(*fault, L"noFlight") {
771             ...
772         }
773         else {

```

Deleted: wcsncmp

Deleted: Specification

Deleted: 19 March

774
775
776
777
778
779

```
    ...  
    }  
}  
  
    return;  
}
```

780 **3.4.3 Synchronous Invocation**

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

786

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

789

790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824

```
#include "SCA.h"  
#include "TravelService.h"  
  
void synchronousClientFunction()  
{  
    SCAREF serviceToken;  
    int compCode, reason;  
  
    struct itineraryData id;  
    struct confirmationData *cd;  
    wchar_t *fault, *faultDetails;  
  
    ...  
  
    SCAGetReference(L"customerService", &serviceToken, &compCode, &reason);  
  
    SCAInvoke(serviceToken, L"makeReservations", sizeof(itineraryData),  
              (void *)&id, sizeof(confirmationData), (void *)&cd,  
              &compCode, &reason);  
    if (compCode == SCA_FAULT) {  
        ...  
    }  
    else {  
        SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,  
                           &faultDetails, &compCode, &reason);  
        if (wcsncmp(*fault, L"noFlight")) {  
            ...  
        }  
        else {  
            ...  
        }  
    }  
}  
  
    return;  
}
```

Deleted: wcsncmp

Deleted: Specification

Deleted: 19 March

4 Asynchronous Programming

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

4.1 Non-blocking Calls

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

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

Deleted: 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.

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

```
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
  <service name="LoanService">
    <interface.c header="LoanService.h">
      <function name="reportEvent" oneWay="true" />
    </interface.c>
  </service>
</componentType>
```

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

4.2 Callbacks

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

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

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

Deleted: Specification

Deleted: 19 March

4.2.1 Using Callbacks

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

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

```
double requestQuotation(char *productCode,int quantity);

char *getState();
char *getZipCode();
char *getCreditRating();
```

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

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

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

double requestQuotation(char *productCode,int quantity) {
    double price, discount = 0;
    char state[3], creditRating[4];
    SCAREF callbackRef;
    int compCode, reason;

    price = getPrice(productQuote, quantity);

    SCAGetCallback(L"", &callbackRef, &compCode, &reason);
    SCAInvoke(callbackRef, L"getState", 0, NULL, sizeof(state), state,
        &compCode, &reason);
    if (quantity > 1000 && strcmp(state,"FL") == 0)
        discount = 0.05;
    SCAInvoke(callbackRef, L"getCreditRating", 0, NULL, sizeof(creditRating),
        creditRating, &compCode, &reason);
    if (quantity > 10000 && creditRating[0] == 'A')
        discount += 0.05;
    SCAReleaseCallback(callbackRef, &compCode, &reason);
    return price * (1-discount);
}
```

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

Deleted: Specification

Deleted: 19 March

```

926 #include "QuotationCallback.h"
927 #include "SCA.h"
928
929 char state[3] = "TX", zipCode[6] = "78746", creditRating[3] = "AA";
930
931 ClientFunction() {
932     SCAREF serviceToken;
933     int compCode, reason;
934
935     SCAGetReference(L"quotationService", &serviceToken, &compCode, &reason);
936
937     SCA_requestQuotation(serviceToken, "AB123", 2000);
938 }
939
940 char *getState() {
941     return state;
942 }
943 char *getZipCode() {
944     return zipCode;
945 }
946 char *getCreditRating() {
947     return creditRating;
948 }
949

```

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

954 4.2.2 Callback Instance Management

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

960
961 As described Using Callbacks, a stateful callback can obtain information relating to the original service
962 request from parameters on the callback request. Alternatively, a composite-scoped client could store
963 information relating to the original request as instance data and retrieve it when the callback request is
964 received. These approaches could be combined by using a key passed on the callback request (e.g., an
965 order ID) to retrieve information that was stored in a composite-scoped instance by the client code that
966 made the original request.

Deleted: Using Callbacks

967 4.2.3 Implementing Multiple Bidirectional Interfaces

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

Deleted: Specification

Deleted: 19 March

971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003

5 Error Handling

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

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

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

```
/* Completion Codes */
#define SCACC_OK          0
#define SCACC_WARNING    1
#define SCACC_FAULT      2
#define SCACC_ERROR      3

/* Reason Codes */
#define SCA_SERVICE_UNAVAILABLE 1
#define SCA_MULTIPLE_SERVICES  2
#define SCA_DATA_TRUNCATED     3
#define SCA_PRAMATER_ERROR     4
#define SCA_BUSY                5
#define SCA_RUNTIME_ERROR      6

/* Response Types */
#define SCA_NO_RESPONSE        0
#define SCA_REPLY_MESSAGE     1
#define SCA_FAULT_MESSAGE     2
```

Deleted: 4

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

Deleted: Specification
Deleted: 19 March

1004 **6 C API**

1005 **6.1 SCA Programming Interface**

1006 The following shows the C interface declarations for synchronous programming.

1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060

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

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

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

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

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

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

Deleted: c

Deleted: Specification

Deleted: 19 March

```

1061 void SCAPPropertyCChars(wchar_t *propertyName,
1062                          char **value,
1063                          int *size,
1064                          int *compCode,
1065                          int *reason);
1066
1067 void SCAPPropertyShort(wchar_t *propertyName,
1068                       int16_t *value,
1069                       int *compCode,
1070                       int *reason);
1071
1072 void SCAPPropertyInt(wchar_t *propertyName,
1073                    int32_t *value,
1074                    int *compCode,
1075                    int *reason);
1076
1077 void SCAPPropertyLong(wchar_t *propertyName,
1078                     int64_t *value,
1079                     int *compCode,
1080                     int *reason);
1081
1082 void SCAPPropertyFloat(wchar_t *propertyName,
1083                      float *value,
1084                      int *compCode,
1085                      int *reason);
1086
1087 void SCAPPropertyDouble(wchar_t *propertyName,
1088                       double *value,
1089                       int *compCode,
1090                       int *reason);
1091
1092 void SCAPPropertyString(wchar_t *propertyName,
1093                       wchar_t **value,
1094                       int *size,
1095                       int *compCode,
1096                       int *reason);
1097
1098 void SCAPPropertyCString(wchar_t *propertyName,
1099                        char **value,
1100                        int *size,
1101                        int *compCode,
1102                        int *reason);
1103
1104 void SCAPPropertyStruct(wchar_t *propertyName,
1105                       void **value,
1106                       int *compCode,
1107                       int *reason);
1108
1109 void SCAGetReplyMessage(SCAREF token,
1110                       int *bufferLen,
1111                       char *buffer,
1112                       int *compCode,
1113                       int *reason);
1114
1115 void SCAGetFaultMessage(SCAREF token,
1116                       int *bufferLen,
1117                       wchar_t **faultName,
1118                       char *buffer,
1119                       int *compCode,
1120                       int *reason);
1121
1122 void SCASetFaultMessage(wchar_t *serviceName,
1123                       wchar_t *operationName,
1124

```

Deleted: Specification

Deleted: 19 March

```
1125         wchar_t *faultName,
1126         int bufferLen,
1127         char *buffer,
1128         int *compCode,
1129         int *reason);
1130
1131 void SCASelf(wchar_t *serviceName,
1132             SCAREF *serviceToken,
1133             int *compCode,
1134             int *reason);
1135
1136 void SCAGetCallback(wchar_t *serviceName,
1137                    SCAREF *serviceToken,
1138                    int *compCode,
1139                    int *reason);
1140
1141 void SCAReleaseCallback(SCAREF serviceToken,
1142                        int *compCode,
1143                        int *reason);
1144
1145 void SCAInvokeAsync(SCAREF token,
1146                    wchar_t *operationName,
1147                    int inputMsgLen,
1148                    void *inputMsg,
1149                    void (*handler)(short),
1150                    int *compCode,
1151                    int *reason);
1152
1153 void SCAInvokePoll(SCAREF token,
1154                   wchar_t *operationName,
1155                   int inputMsgLen,
1156                   void *inputMsg,
1157                   int *compCode,
1158                   int *reason);
1159
1160 void SCACheckResponse(SCAREF token,
1161                      short *responseType,
1162                      int *compCode,
1163                      int *reason);
1164
1165 void SCACancelInvoke(SCAREF token,
1166                     int *compCode,
1167                     int *reason);
1168
1169 void SCAEntryPoint(wchar_t *serviceURI,
1170                   wchar_t *domainURI,
1171                   SCAREF *serviceToken,
1172                   int *compCode,
1173                   int *reason);
```

Deleted: ;

1175 The C synchronous programming interface has the following functions:

1176 **6.1.1 SCAGetReference**

1177 A C component implementation uses SCAGetReference() to initialize a Reference before invoking any
1178 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

Deleted: Specification
Deleted: 19 March

		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.	

1179 **6.1.2 SCAGetReferences**

1180 A C component implementation uses `SCAGetReferences()` to initialize a Reference that might be
1181 bound to multiple Services before invoking any operations of the Reference.

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

1182 **6.1.3 SCAInvoke**

1183 A C component implementation uses `SCAInvoke()` to invoke an operation of an interface.

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

Deleted: Specification

Deleted: 19 March

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.	

Deleted: R

6.1.4 SCAProperty<T>

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

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

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. [C60002]

If <T> is one of: Boolean, Byte, Char, CChar, Short, Int, Long, Float, Double or Struct

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

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

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

Deleted: Specification

Deleted: 19 March

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

1198 6.1.5 SCAGetReplyMessage

1199 A C component implementation uses SCAGetReplyMessage() to retrieve the reply message of an
1200 operation invocation if the length of the message exceeded the buffer size provided on SCAInvoke().

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

1201 6.1.6 SCAGetFaultMessage

1202 A C component implementation uses SCAGetFaultMessage() to retrieve the details of a business fault
1203 received in response to an operation invocation.

Deleted: Specification

Deleted: 19 March

Precondition	C component instance is running, has a valid token and an <code>SCAInvoke()</code> returned a <code>SCACC_FAULT compCode</code>	
Input Parameter	token	Token returned by prior <code>SCAGetReference()</code> , <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
In/Out Parameter	bufferLen	Input: Maximum number of bytes that can be returned Output: Actual number of bytes returned or size needed to hold entire message
Output Parameters	faultName	Name of the business fault
	buffer	Fault message
	compCode	SCACC_OK, if the call is successful SCACC_WARNING, if the fault data was truncated. The buffer size needs to be increased and the call repeated with the larger buffer. SCACC_ERROR, otherwise – see reason for details
	reason	SCA_DATA_TRUNCATED if the fault data was truncated. SCA_PARAMETER_ERROR if the last operation invoked on the Reference did <u>not</u> return a business fault
Post Condition	The <code>referenceToken</code> remains valid for subsequent calls.	

Deleted: B

Deleted: R

1204 6.1.7 SCASetFaultMessage

1205 A C component implementation uses `SCASetFaultMessage()` to return a business fault in response to
1206 a request.

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

Deleted: Specification

Deleted: 19 March

1207 **6.1.8 SCASelf**

1208 A C component implementation uses SCASelf () to access a Service it provides.

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

1209 **6.1.9 SCAGetCallback**

1210 A C component implementation uses SCAGetCallback () to initialize a Service before invoking any
1211 callback operations of the Service.

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

1212 **6.1.10 SCAReleaseCallback**

1213 A C component implementation uses SCAReleaseCallback () to tell the SCA runtime it has completed
1214 callback processing and the EndPointReference can be released.

Precondition	C component instance is running and has a valid serviceToken	
Input Parameter	serviceToken	Token returned by prior SCAGetCallback () call.
Output Parameters	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	reason	SCA_PARAMETER_ERROR if the serviceToken is not valid

Deleted: Specification

Deleted: 19 March

Post Condition	The token becomes invalid for subsequent calls.
----------------	---

1215 6.1.11 SCAInvokeAsync

1216 A C component implementation uses `SCAInvokeAsync()` to invoke a long running operation of an
1217 interface using the asynchronous style.

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

1218 6.1.12 SCAInvokePoll

1219 A C component implementation uses `SCAInvokePoll()` to invoke a long running operation of a
1220 Reference using the polling style.

Precondition	C component instance is running and has a valid token	
Input Parameters	token	Token returned by prior <code>SCAGetReference()</code> , <code>SCAGetReferences()</code> , <code>SCASelf()</code> or <code>SCAGetCallback()</code> call.
	operationName	Name of the operation to invoke
	inputMsgLen	Length of the request message buffer
	inputMsg	Request message
Output Parameters	compCode	<code>SCACC_OK</code> , if the call is successful <code>SCACC_ERROR</code> , otherwise – see reason for details
	reason	<code>SCA_BUSY</code> if an operation is already outstanding for this Reference or Callback

Deleted: Specification

Deleted: 19 March

		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.	

1221 6.1.13 SCACheckResponse

1222 A C component implementation uses SCACheckResponse() to determine if a response to a long
1223 running operation request has been received.

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

1224 6.1.14 SCACancelInvoke

1225 A C component implementation uses SCACancelInvoke() to cancel a long running operation request.

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

1226 6.1.15 SCAEntryPoint

1227 Non-SCA C code uses SCAEntryPoint() to access a Service before invoking any operations of the
1228 Service.

Precondition	None	
Input Parameter	serviceURI	URI of the Service to access

Deleted: Specification

Deleted: 19 March

	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.	

6.2 Program-Based Implementation Support

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

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

```
void SCAService(wchar_t *serviceName, int *compCode, int *reason);
void SCAOperation(wchar_t *operationName, int *compCode, int *reason);
void SCAMessageIn(wchar_t *serviceName,
                  wchar_t *operationName,
                  int *bufferLen,
                  void *buffer,
                  int *compCode,
                  int *reason);
void SCAMessageOut(wchar_t *serviceName,
                   wchar_t *operationName,
                   int *bufferLen,
                   void *buffer,
                   int *CompCode,
                   int *Reason);
```

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

6.2.1 SCAService

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

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

6.2.2 SCAOperation

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

Deleted: Specification

Deleted: 19 March

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.	

1258 6.2.3 SCAMessageIn

1259 A program-based C component implementation uses `SCAMessageIn()` to retrieve its request message.

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

1260 6.2.4 SCAMessageOut

1261 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.	

1262
1263
1264
1265
1266
1267

1268

1269
1270
1271
1272

1273

1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292

1293

1294
1295
1296
1297
1298
1299

1300

1301
1302
1303
1304
1305
1306
1307

7 C Contributions

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

7.1 Executable files

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

7.1.1 Executable in contribution

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

The following shows a contribution containing a DLL.

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

The SCDL for the AutoInsuranceService component is:

```
<component name="AutoInsuranceService">
  <implementation.c module="autoinsurance" path="bin/"
    componetType="AutoInsurance" />
</component>
```

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

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

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

Deleted: Specification
Deleted: 19 March

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

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

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

An export of the form:

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

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

7.1.3 Executable outside of contribution (Import)

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

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

The SCDL for the UnderwritingService component is:

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

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

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

Deleted: Specification

Deleted: 19 March

1358 **7.2 componentType files**

1359 As stated in section 2.5, each component implemented in C has a corresponding componentType file.
1360 This componentType file is, by default, located in the root directory of the composite containing the
1361 component or a subdirectory of the composite root with a name specified on the @componentType
1362 attribute as shown in the following example.

```
1363  
1364 META-INF/  
1365   sca-contribution.xml  
1366 bin/  
1367   autoinsurance.dll  
1368 AutoInsurance/  
1369   AutoInsurance.composite  
1370   AutoInsuranceService/  
1371     AutoInsurance.h  
1372     AutoInsurance.componentType
```

1373
1374 The SCDL for the AutoInsuranceService component is:
1375

```
1376 <component name="AutoInsuranceService">  
1377   <implementation.c module="autoinsurance" path="bin/"  
1378     componentType="AutoInsurance" />  
1379 </component>
```

1380
1381 Since there is a one-to-one correspondence between implementations and componentTypes, when an
1382 implementation is shared between contributions, it is desirable to also share the componentType file.
1383 ComponentType files can be exported and imported in the same manner as executable files. The
1384 location of a .componentType file can be specified using the @componentType attribute of the
1385 implementation.c element.

```
1386  
1387 <component name="UnderwritingService">  
1388   <implementation.c library="rates" path="rates:bin/"  
1389     componentType="rates:types/Underwriting" />  
1390 </component>
```

1391 **7.3 C Contribution Extensions**

1392 **7.3.1 Export.c**

1393 The following snippet shows the schema for the C export element used to make an executable or
1394 componentType file visible outside of a contribution.

```
1395  
1396 <?xml version="1.0" encoding="ASCII"?>  
1397 <!-- export.c schema snippet -->  
1398 <export.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
1399   name="QName" path="string"? >
```

1400
1401 The **export.c** element has the following **attributes**:

- 1402
- 1403 • **name : QName (1..1)** – name of the export. The @name attribute of a <export.c/> element MUST be
1404 unique amongst the <export.c/> elements in a domain, [C70001]
 - 1405 • **path : string (0..1)** – path of the exported executable relative to the root of the contribution. If not
present, the entire contribution is exported.

Deleted: The @name attribute of a
<export.c/> element MUST be unique
amongst the <export.c/> elements in
a domain.

Deleted: Specification

Deleted: 19 March

1406 **7.3.2 Import.c**

1407 The following snippet shows the schema for the C import element used to reference an executable or
1408 componentType file that is outside of a contribution.

1409
1410

```
<?xml version="1.0" encoding="ASCII"?>  
1411 <!-- import.c schema snippet -->  
1412 <import.c xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
1413       name="QName" location="string" >
```

1414
1415 The **import.c** element has the following **attributes**:

- 1416 | • **name : QName (1..1)** – name of the import. The @name attribute of a <import.c/> child element of a
1417 | <contribution/> MUST be unique amongst the <import.c/> elements in of that contribution. [C70002]
1418 | • **location : string (1..1)** – either the QName of a export or a file system location. If the value does not
1419 | match an export name it is taken as an absolute file system path.

Deleted: The @name attribute of a <import.c/> child element of a <contribution/> MUST be unique amongst the <import.c/> elements in of that contribution.

Deleted: Specification

Deleted: 19 March

1420
1421
1422

1423
1424
1425

1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436

1437

8 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.

8.1 Local service

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

- Any fundamental or compound types as defined by C. [C80001]

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

Formatted: Pattern: Clear (Turquoise)

Deleted: 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.

Formatted: Pattern: Clear (Turquoise)

Formatted: Pattern: Clear (Pale Blue)

Formatted: Pattern: Clear (Pale Blue)

Formatted: Bullets and Numbering

Formatted: Pattern: Clear (Pale Blue)

Formatted: Pattern: Clear (Pale Blue)

Formatted: Pattern: Clear (Pale Blue)

Formatted: Pattern: Clear (Pale Blue)

Formatted: Pattern: Clear (Pale Blue)

Deleted: 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.

Deleted: Unless the interface is marked as allowing pass by reference semantics, the behavior of the following are not defined:¶
<#>Pointers.¶

Formatted: Bullets and Numbering

Deleted: Specification

Deleted: 19 March

1438
1439
1440
1441
1442
1443
1444

9 Restrictions on C header files

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

A C header file used to define an interface MUST:

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

Deleted: A C header file used to define an interface MUST:
Declare at least one function or message format struct

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

- Macros [C90002]

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

Deleted: Specification
Deleted: 19 March

1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464

1465
1466
1467
1468
1469
1470
1471

1472
1473
1474
1475
1476

1477
1478
1479
1480
1481
1482
1483
1484
1485

10WSDL 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](#).

Deleted: Data Binding

The following general rules apply to 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 that implement an interface.
- References to Java methods are considered references to C functions.
- References to Java interfaces are considered references to a collection of C function 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.

10.1 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.

10.1.1 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.

10.1.2 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.

Deleted: Complex Content Binding

Deleted: Specification

Deleted: 19 March

1486 If the implementation is contained in a program, the declarations are all struct declarations. See the next
1487 section for details.
1488
1489 In the absence of customizations, an SCA implementation SHOULD map each portType to separate
1490 header file. An SCA implementation MAY use any sca-c:prefix binding declarations to control this
1491 mapping. [C100001] For example, all portTypes in a WSDL document with a common sca-c:prefix binding
1492 declaration could be mapped to a single header file..
1493
1494 Header file naming is implementation dependent.

1495 **10.1.3 Operations**

1496 Asynchronous mapping is not supported.

1497 **10.1.3.1 Operation Names**

1498 WSDL operation names are only guaranteed to be unique with a portType. C requires function and struct
1499 names loaded into an address space to be distinct. The mapping of operation names to function or struct
1500 names have to take this into account.

1501
1502 For components implemented in libraries, in the absence of customizations, an SCA implementation
1503 MUST concatenate the portType name, with the first character converted to lower case, and the operation
1504 name, with the first character converted to upper case, to form the function. [C100002]

1505
1506 An application can customize this mapping using the sca-c:prefix and/or sca-c:function binding
1507 declarations.

1508
1509 For program-based service implementations:

- 1510 • If the number of **In** parameters plus the number of **In/Out** parameters is greater than one there will be
1511 a request struct.
1512 • If the number of **Out** parameters plus the number of **In/Out** parameters is greater than one there will
1513 be a response struct.

1514
1515 For components implemented in a program, in the absence of customizations, an SCA implementation
1516 MUST concatenate the portType name, with the first character converted to lower case, and the operation
1517 name, with the first character converted to upper case, to form the request struct name. Additionally an
1518 SCA implementation MUST append "Response" to the request struct name to form the response struct
1519 name.[C100005]

1520
1521 An application can customize this mapping using the sca-c:prefix and/or sca-c:struct binding declarations.

1522 **10.1.3.2 Message and Part**

1523 In the absence of any customizations for a WSDL operation that does not meet the requirements for the
1524 wrapped style, the name of a mapped function parameter or struct member MUST be the value of the
1525 name attribute of the wsdl:part element with the first character converted to lower case.[C100003]

1526
1527 In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped
1528 style, the name of a mapped function parameter or struct member MUST be the value of the local name
1529 of the wrapper child with the first character converted to lower case.[C100004]

1530
SCA Client and Implementation Model for C Specification Version 1.1
Copyright © OASIS® 2007, 2009. All Rights Reserved.

30 April 2009
Page 49 of 121

Deleted: For components implemented in a program, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the request struct name. Additionally an SCA implementation MUST append "Response" to the request struct name to form the response struct name.

Deleted: 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.

Deleted: 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.

Deleted: Specification

Deleted: 19 March

1531 An application can customize this mapping using the sca-c:parameter binding declaration.

1532

1533 For library-based service implementations, an SCA implementation MUST map In parameters as pass
1534 by-value and In/Out and Out parameters as pass via pointers. [C100019]

1535

1536 For program-based service implementations, an SCA implementation MUST map all values in the input
1537 message as pass by-value and the updated values for In/Out parameters and all Out parameters in the
1538 response message as pass by-value. [C100020]

Deleted: For library-based service implementations, an SCA implementation MUST map In parameters as pass by-value and In/Out and Out parameters as pass via pointers.

Deleted: 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.

Deleted: Data Binding

1539 10.1.4 Types

1540 As per section Data Binding, (based on SDO type mapping).

1541

1542 MTOM/XOP content processing is left to the application.

1543 10.1.5 Fault

1544 C has no exceptions so an API is provided for getting and setting fault messages (see
1545 SCAGetFaultMessage and SCASetFaultMessage). Fault messages are mapped in same manner as
1546 input and output messages.

1547

1548 In the absence of customizations, an SCA implementation MUST map the name of the message element
1549 referred to by a fault element to name of the struct describing the fault message content. If necessary, to
1550 avoid name collisions, an implementation MAY append "Fault" to the name of the message element when
1551 mapping to the struct name. [C100006]

Deleted: 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.

1552
1553 An application can customize this mapping using the sca-c:struct binding declaration.

1554 10.1.6 Service and Port

1555 This mapping does not define generation of client side code.

1556 10.1.7 XML Names

1557 See comments in Operations

1558 10.2 Interpretations for C to WSDL Mapping

1559 10.2.1 Package

1560 Not relevant.

1561

1562 An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be
1563 configurable. [C100007]

1564 10.2.2 Class

1565 Not relevant since mapping is only based on declarations.

1566 10.2.3 Interface

1567 The declarations in a header file are used to define an interface. A header file can be used to define an
1568 interface if it satisfies either (for components implemented in libraries):

Deleted: Specification

Deleted: 19 March

- 1569 • Contains one or more function declarations
- 1570 • Any of these functions declarations might carry a @WebFunction annotation
- 1571 • The parameters and return types of these function declarations are compatible with the C to XML
- 1572 Schema mapping in Data Binding.
- 1573 or (for components implemented in programs):
- 1574 • Contains one request message struct declarations
- 1575 • Any of the request message struct declarations might carry a @WebOperation annotation
- 1576 • Any of the request message struct declarations can have a corresponding response message struct,
- 1577 identified by either having a name with "Response" appended to the request message struct name or
- 1578 identified in a @WebOperation annotation
- 1579 • Members of these struct declarations are compatible with the C to XML Schema mapping in Data
- 1580 Binding.

Deleted: Data Binding

1582 In the absence of customizations, an SCA implementation MUST map the header file name to the

1583 portType name. An implementation MAY append "PortType" to the header file name in the mapping to

1584 the portType name. [C100008]

Deleted: 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.

1586 An application can customize this mapping using the @WebService annotation.

1587 **10.2.4 Method**

1588 For components implemented in libraries, functions map to operations.

1590 In the absence of customizations, an SCA implementation MUST map the function name to the operation

1591 name, stripping the portType name, if present and any namespace prefix from the front of function name

1592 before mapping it to the operation name. [C100009]

Formatted: Pattern: Clear (Light Green)

Deleted: 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.

1594 An application can customize function to operation mapping or exclude a function from an interface using

1595 the @WebFunction annotation.

1597 For components implemented in programs, operations are mapped from request structs.

1599 In the absence of customizations, a struct with a name that does not end in "Response" or "Fault" is

1600 considered to be a request message struct and an SCA implementation MUST map the struct name to

1601 the operation name, stripping the portType name, if present, and any namespace prefix from the front of

1602 the struct name before mapping it to the operation name. [C100010]

Deleted: 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.

1604 An application can customize struct to operation mapping or exclude a struct from an interface using the

1605 @WebOperation annotation.

1606 **10.2.5 Method Parameters and Return Type**

1607 For components implemented in libraries, function parameters and return type map to either message or

1608 global element components.

1610 In the absence of customizations, an SCA implementation MUST map the parameter name, if present, to

1611 the part or global element component name. If the parameter does not have a name the SCA

1612 implementation MUST use argN as the part or global element child name. [C100011]

Deleted: Specification

Deleted: 19 March

1613

1614 An application can customize parameter to message or global element component mapping using the
1615 @WebParam annotation.

1616
1617 In the absence of customizations, an SCA implementation MUST map the return type to a part or global
1618 element child named "return". [C100012]

Deleted: In the absence of customizations, an SCA implementation MUST map the return type to a part or global element child named "return".

1619
1620 An application can customize return type to message or global element component mapping using the
1621 @WebReturn annotation.

1622
1623 An SCA implementation MUST map:

- 1624 • a function's return value as an out parameter.
- 1625 • by-value and const parameters as in parameters.
- 1626 • in the absence of customizations, pointer parameters as in/out parameters. [C100017]

Formatted: Bullets and Numbering

Deleted: An SCA implementation MUST map:
[<#>a function's return value as an out parameter.
[<#>by-value and const parameters as in parameters.
[in the absence of customizations, pointer parameters as in/out parameters.

1627
1628 An application can customize parameter classification using the @WebParam annotation.

1629
1630 Program based implementation SHOULD use the Document-Literal style and encoding. [C100013]

1631
1632 In the absence of customizations, an SCA implementation MUST map the struct member name to the
1633 part or global element child name. [C100014]

1634
1635 An application can customize struct member to message or global element component mapping using the
1636 @WebParam annotation.

- 1637
- 1638 • Members of the request struct that are not members of the response struct are **in** parameters
 - 1639 • Members of the response struct that are not members of the request struct are **out** parameters
 - 1640 • Members of both the request and response structs are **in/out** parameters. Matching is done by
 - 1641 member name. An SCA implementation MUST ensure that in/out parameters have the same type in
 - 1642 the request and response structs. [C100015]

Deleted: An SCA implementation MUST ensure that in/out parameters have the same type in the request and response structs.

1643 10.2.6 Service Specific Exception

1644 C has no exceptions. A struct can be annotated as a fault message type. A function or operation
1645 declaration can be annotated to indicate that it potentially generates a specific fault.

1646
1647 An application can define a fault message format using the @WebFault annotation.

1648
1649 An application can indicate that a WSDL fault might be generated by a function or operation using the
1650 @WebThrows annotation.

1651 10.2.7 Generics

1652 Not relevant.

1653 10.2.8 Service and Ports

1654 An SCA runtime invokes function (or programs) as a result of receiving an operation request. No
1655 mapping to Service or Ports is defined by this specification.

Deleted: Specification

Deleted: 19 March

1656 **10.3 Data Binding**

1657 The data in wsdl:parts or wrapper children is mapped to and from C function parameters and return
1658 values (for library-based component implementations), or struct members (for program-based component
1659 implementations and fault messages).

1660 **10.3.1 Simple Content Binding**

1661 The mapping between XSD simple content types and C types follows the convention defined in the SDO
1662 specification [SDO21]. The following table summarizes that mapping as it applies to SCA services.
1663

<i>XSD Schema Type →</i>	<i>C Type</i>	<i>→ XSD Schema Type</i>
anySimpleType	wchar_t *	string
anyType	DATAOBJECT	anyType
anyURI	wchar_t *	string
base64Binary	char *	string
boolean	char	string
byte	int8_t	byte
date	wchar_t *	string
dateTime	wchar_t *	string
decimal	wchar_t *	string
double	double	double
duration	wchar_t *	string
ENTITIES	wchar_t *	string
ENTITY	wchar_t *	string
float	float	float
gDay	wchar_t *	string
gMonth	wchar_t *	string
gMonthDay	wchar_t *	string
gYear	wchar_t *	string
gYearMonth	wchar_t *	string
hexBinary	char *	string
ID	wchar_t *	string
IDREF	wchar_t *	string
IDREFS	wchar_t *	string
int	int32_t	int
integer	wchar_t *	string

Deleted: Specification

Deleted: 19 March

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

1664 Table 1: XSD simple type to C type mapping
1665

C Type →	XSD Schema Type
_Bool	boolean
wchar_t	string
signed char	byte
unsigned char	unsignedByte
short	short
unsigned short	unsignedShort
int	int
unsigned int	unsignedInt
long	long

Deleted: Specification
Deleted: 19 March

unsigned long	unsignedLong
long long	long
unsigned long long	unsignedLong
wchar_t*	string
long double	decimal
time_t	time
struct tm	dateTime

1666 *Table 2: C type to XSD type mapping*

1667
1668 The C standard does not define value ranges for integer types so it is possible that on a platform
1669 parameters or return values could have values that are out of range for the default XSD schema type. In
1670 these circumstances, the mapping would need to be customized, using @WebParam or @WebResult if
1671 supported, or some other implementation-specific mechanism.

1672
1673 An SCA implementation MUST map simple types as defined in Table 1 and Table 2 by default. [C100021]

1674
1675 An SCA implementation MAY map boolean to _Bool by default. [C100022]

1676 **10.3.1.1 WSDL to C Mapping Details**

1677 In general, when xsd:string and types derived from xsd:string map to a struct member, the
1678 mapping is to a combination of a wchar_t * and a separately allocated data array. If either the length
1679 or maxLength facet is used, then a wchar_t[] is used. If the pattern facet is used, this might allow
1680 the use of char and/or also constrain the length.

1681 Example:

```
1682     <xsd:element name="myString" type="xsd:string"/>  
1683     maps to:  
1684     wchar_t *myString;  
1685     /* this points to a dymically allocated buffer with the data */  
1686  
1687     <xsd:simpleType name="boundedString25">  
1688     <xsd:restriction base="xsd:string">  
1689     <xsd:length value="25"/>  
1690     </xsd:restriction>  
1691   </xsd:simpletype>  
1692   ...  
1693   <xsd:element name="myString" type="boundedString25"/>
```

```
1694     maps to:  
1695     wchar_t myString[26];  
1696
```

- 1697 • When unbounded binary data maps to a struct member, the mapping is to a char * that points to
1698 the location where the actual data is located. Like strings, if the binary data is bounded in length, a
1699 char[] is used.

1700
1701 Examples:

```
1702     <xsd:element name="myData" type="xsd:hexBinary"/>
```

Deleted: Specification

Deleted: 19 March

1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751

maps to:

```
char *myData;
/* this points to a dymically allocated buffer with the data */

<xsd:simpleType name="boundedData25">
  <xsd:restriction base="xsd:hexBinary">
    <xsd:length value="25"/>
  </xsd:restriction>
</xsd:simpletype>
...
<xsd:element name="myData" type="boundedData25"/>
```

maps to:

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

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

Examples:

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

maps to:

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

```
<xsd:simpleType name="lineNumList">
  <xsd:list itemType="xsd:int"/>
</xsd:simpleType>
<xsd:simpleType name="lineNumList6">
  <xsd:restriction base="lineNumList ">
    <xsd:minLength value="1"/>
    <xsd:maxLength value="6"/>
  </xsd:restriction>
</xsd:simpletype>
...
<xsd:element name="lineNums" type="lineNumList6"/>
```

maps to:

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

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

Examples:

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

maps to:

```
size_t counts_num;
int *counts;
```

Deleted: Specification
Deleted: 19 March

1752 /* this points to a dynamically allocated array of struct tm's */

1753

```
1754 <xsd:simpleType name="lineNumList">
1755   <xsd:list itemType="xsd:int"/>
1756 </xsd:simpleType>
1757 ...
1758 <xsd:element name="lineNums" type="lineNumList"/>
```

1759 maps to:

```
1760 size_t lineNums_num;
1761 long *lineNums;
1762 /* this points to a dynamically allocated array of longs */
1763
```

- 1764
- Union Types are not supported.

1765 10.3.1.2 C to WSDL Mapping Details

- 1766
- wchar_t[] and char[] map to xsd:string with a maxLength facet.
 - 1767 • C arrays map as normal elements but with multiplicity allowed via the minOccurs and maxOccurs facets.
 - 1768

1769

1770 Example:

```
1771 long myFunction(char* name, int idList[], double value);
```

1772 maps to:

```
1773 <xsd:element name="myFunction">
1774   <xsd:complexType>
1775     <xsd:sequence>
1776       <xsd:element name="name" type="xsd:string"/>
1777       <xsd:element name="idList" type="xsd:short"
1778         minOccurs="0" maxOccurs="unbounded"/>
1779       <xsd:element name="value" type="xsd:double"/>
1780     </xsd:sequence>
1781   </xsd:complexType>
1782 </xsd:element>
1783
```

- 1784
- Multi-dimensional arrays map into nested elements.

1785

1786 Example:

```
1787 long myFunction(int multiIdArray[][4][2]);
```

1788 maps to:

```
1789 <xsd:element name="myFunction">
1790   <xsd:complexType>
1791     <xsd:sequence>
1792       <xsd:element name="multiIdArray"
1793         minOccurs="0" maxOccurs="unbounded"/>
1794       <xsd:complexType>
1795         <xsd:sequence>
1796           <xsd:element name="multiIdArray"
1797             minOccurs="4" maxOccurs="4"/>
1798         <xsd:complexType>
1799           <xsd:sequence>
1800             <xsd:element name="multiIdArray" type="xsd:short"

```

Deleted: Specification

Deleted: 19 March

```

minOccurs="2" maxOccurs="2" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

```

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

10.3.2 Complex Content Binding

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

Formatted: Pattern: Clear (Tan)

Deleted: 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.

Formatted: Pattern: Clear (Tan)

10.3.2.1 WSDL to C Mapping Details

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

Example:

```

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

```

maps to:

```

struct myType {
  wchar_t name[26];
  size_t idList_num;
  long *idList;
}

```

Deleted: Specification

Deleted: 19 March

```

1851     /* this points to a dynamically allocated array of longs */
1852     double value;
1853 };
1854

```

- 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.

Example:

```

1860 <xsd:element name="myVariable">
1861   <xsd:complexType name="myType">
1862     <xsd:all>
1863       <xsd:element name="name" type="xsd:string"/>
1864       <xsd:element name="idList" type="xsd:int"
1865         minOccurs="0" maxOccurs="unbounded"/>
1866       <xsd:element name="value" type="xsd:double"/>
1867     </xsd:all>
1868   </xsd:complexType>
1869 </xsd:element>

```

maps to:

```

1871 struct myType {
1872     wchar_t *name;
1873     /* this points to a dynamically allocated string */
1874     size_t idList_num;
1875     long *idList;
1876     /* this points to a dynamically allocated array of longs */
1877     double *value;
1878     /* this points to a dynamically allocated long */
1879 } *pmyVariable, myVariable;
1880

```

- 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.

Example:

```

1887 <xsd:element name="priority" type="xsd:short" nillable="true"/>

```

maps to:

```

1889 int16_t *pprioiry, priority;
1890

```

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

10.3.2.2 C to WSDL Mapping Details

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

Example:

```

1896 char *myFunction(struct DataStruct data, int id);

```

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

```

1898 struct DataStruct {

```

Deleted: Specification

Deleted: 19 March

```
1899 char *name;
1900 double value;
1901 };
```

1902 maps to:

```
1903 <xsd:element name="myFunction">
1904   <xsd:complexType>
1905     <xsd:sequence>
1906       <xsd:element name="data" type="DataStruct" />
1907       <xsd:element name="id" type="xsd:int"/>
1908     </xsd:sequence>
1909   </xsd:complexType>
1910 </xsd:element>
1911
1912 <xsd:complexType name="DataStruct">
1913   <xsd:sequence>
1914     <xsd:element name="name" type="xsd:string"/>
1915     <xsd:element name="value" type="xsd:double"/>
1916   </xsd:sequence>
1917 </xsd:complexType>
1918
```

- 1919 • char and wchar_t arrays inside of structs are mapped to a restricted subtype of xsd:string that
1920 limits the length the space allowed in the array.

1921
1922 Example:

```
1923 struct DataStruct {
1924   char name[256];
1925   double value;
1926 };
```

1927 maps to:

```
1928 <xsd:element name="myFunction">
1929   <xsd:complexType>
1930     <xsd:sequence>
1931       <xsd:element name="data" type="DataStruct" />
1932       <xsd:element name="id" type="xsd:int"/>
1933     </xsd:sequence>
1934   </xsd:complexType>
1935 </xsd:element>
1936
1937 <xsd:complexType name="DataStruct">
1938   <xsd:sequence>
1939     <xsd:element name="name">
1940       <xsd:simpleType>
1941         <xsd:restriction base="xsd:string">
1942           <xsd:maxLength value="255"/>
1943         </xsd:restriction>
1944       </xsd:simpleType>
1945     </xsd:element>
1946     <xsd:element name="value" type="xsd:double"/>
1947   </xsd:sequence>
1948 </xsd:complexType>
1949
```

- 1950 • C enums define a list of named symbols that map to values. If a function uses an enum type, this is
1951 mapped to a restricted element in the WSDL schema.

Deleted: Specification
Deleted: 19 March

Example:

```
char *getValueFromType(enum ParameterType type);
```

with the ParameterType type defined as an enum:

```
enum ParameterType {  
    UNSET = 1,  
    TYPEA,  
    TYPEB,  
    TYPEC  
};
```

maps to:

```
<xsd:element name="getValueFromType">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element name="type" type="ParameterType"/>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>  
  
<xsd:simpleType name="ParameterType">  
  <xsd:restriction base="xsd:int">  
    <xs:minInclusive value="1"/>  
    <xs:maxInclusive value="4"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

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

Example:

```
enum ParameterType {  
    UNSET = 'u',  
    TYPEA = 'A',  
    TYPEB = 'B',  
    TYPEC = 'C'  
};
```

maps to:

```
<xsd:simpleType name="ParameterType">  
  <xsd:restriction base="xsd:int">  
    <xsd:enumeration value="86"/> <!-- Character 'u' -->  
    <xsd:enumeration value="65"/> <!-- Character 'A' -->  
    <xsd:enumeration value="66"/> <!-- Character 'B' -->  
    <xsd:enumeration value="67"/> <!-- Character 'C' -->  
  </xsd:restriction>  
</xsd:simpleType>
```

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

Example:

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

with types defined as follows:

Deleted: Specification

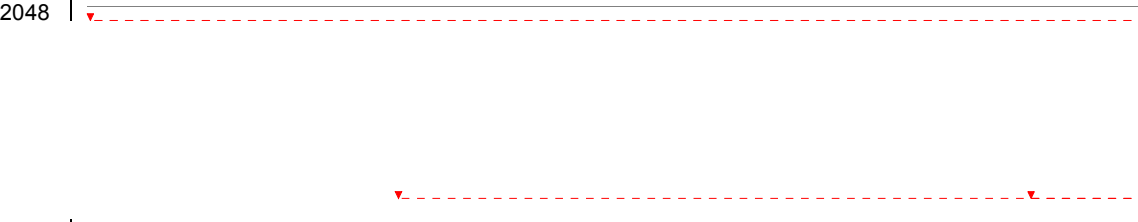
Deleted: 19 March

```
2002 struct DataStruct {
2003     char name[30];
2004     double values[20];
2005     ParameterType type;
2006 };
2007
2008 enum ParameterType {
2009     UNSET = 1,
2010     TYPEA,
2011     TYPEB,
2012     TYPEC
2013 };
```

2014 maps to:

```
2015 <xsd:element name="myFunction">
2016     <xsd:complexType>
2017         <xsd:sequence>
2018             <xsd:element name="data" type="DataStruct"/>
2019         </xsd:sequence>
2020     </xsd:complexType>
2021 </xsd:element>
2022
2023 <xsd:complexType name="DataStruct">
2024     <xsd:sequence>
2025         <xsd:element name="name">
2026             <xsd:simpleType>
2027                 <xsd:restriction base="xsd:string">
2028                     <xsd:maxLength value="29"/>
2029                 </xsd:restriction>
2030             </xsd:simpleType>
2031         </xsd:element>
2032         <xsd:element name="values" type="xsd:double" minOccurs=20
2033 maxOccurs=20/>
2034         <xsd:element name="type" type=" ParameterType"/>
2035     </xsd:sequence>
2036 </xsd:complexType>
2037
2038 <xsd:simpleType name="ParameterType">
2039     <xsd:restriction base="xsd:int">
2040         <xs:minInclusive value="1"/>
2041         <xs:maxInclusive value="4"/>
2042     </xsd:restriction>
2043 </xsd:simpleType>
2044
```

- 2045 • Mapping of C unions is not supported by this specification.
- 2046 • Typedefs are resolved when evaluating parameter and return types. Typedefs are resolved before
- 2047 the mapping to Schema is done.



Deleted: Handling

Deleted: defined

Deleted: mapping

Deleted: , and is implementation dependent. For portability, unions are discouraged in service interfaces

Deleted: <#>Handling for pre-processor directives is not defined by this mapping, and support is implementation dependent. For portability pre-processor directives are discouraged in service interfaces.

Formatted: Bullets and Numbering

Deleted: Specification

Deleted: 19 March

2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072

2073
2074
2075
2076
2077
2078
2079

2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090

11 Conformance

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 document.

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

For code artifacts related to this specification, the specification text is considered to be authoritative and takes precedence over the code artifacts.

An SCA implementation MUST reject a composite file that does not conform to <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> or <http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd>. [C110001]

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

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

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

Deleted: An SCA implementation MUST reject a c 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>.

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

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

Deleted: An SCA implementation MUST reject a WSDL file that does not conform to <http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200901/sca-wsdlex-c-1.1.xsd>.

11.1 Conformance Targets

The conformance targets of this specification are:

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

11.2 SCA Implementations

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

1. It MUST conform to the SCA Assembly Model Specification [ASSEMBLY] and the SCA Policy Framework [POLICY].
2. It MUST comply with all statements in Conformance Points and JAX-WS Conformance Points, related to an SCA implementation, notably all mandatory statements have to be implemented.
3. It MUST implement the SCA C API defined in section C API.
4. It MUST implement the mapping between C and WSDL 1.1 [WSDL11] defined in WSDL to C and C to WSDL Mapping.
5. It MUST support <interface.c/> and <implementation.c/> elements as defined in Component Type and Component in composite, componentType and constrainingType documents.

Deleted: Conformance Items

Deleted: Specification
Deleted: 19 March

- 2091 | 6. It MUST support <export.c/> and <import.c/> elements as defined in C Contributions, in contribution documents.
- 2092
- 2093 | 7. It MAY support source file annotations as defined in C SCA Annotations, C SCA Policy Annotations, and C WSDL Annotations. If source file annotations are supported, the implementation MUST comply with all statements in Annotation Conformance Points related to an SCA implementation, notably all mandatory statements in that section have to be implemented.
- 2094
- 2095
- 2096
- 2097 | 8. It MAY support WSDL extensions as defined in C WSDL Mapping Extensions. If WSDL extensions are supported, the implementation MUST comply with all statements in WSDL Extension Conformance Points related to an SCA implementation, notably all mandatory statements in that section have to be implemented.
- 2098
- 2099
- 2100
- Deleted: C Contributions
- Deleted: C SCA Annotations
- Deleted: C SCA Policy Annotations
- Deleted: C WSDL Annotations
- Deleted: C WSDL Mapping Extensions

2101

11.3 SCA Documents

- 2102 An SCA document conforms to this specification if it meets the following conditions:
- 2103 | 1. It MUST conform to the SCA Assembly Model Specification [ASSEMBLY] and, if appropriate, the SCA Policy Framework [POLICY].
- 2104
- 2105 | 2. If it is a composite document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> and <http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd> schema and MUST comply with the additional constraints on the document contents as defined in Conformance Points.
- 2106
- 2107
- 2108
- 2109
- Deleted: Bullets and Numbering
- Deleted: Conformance Items.
- 2110 If it is a componentType or constrainingType document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd> schema and MUST comply with the additional constraints on the document contents as defined in Conformance Points.
- 2111
- 2112
- 2113
- 2114
- Deleted: Conformance Items
- 2115 If it is a contribution document, it MUST conform to the <http://docs.oasis-open.org/opencsa/sca/200903/sca-contribution-c-1.1.xsd> schema and MUST comply with the additional constraints on the document contents as defined in Conformance Points.
- 2116
- 2117
- 2118
- Deleted: Conformance Items

2119

11.4 C Files

- 2120 A C file conforms to this specification if it meets the following conditions:
- 2121 | 1. It MUST comply with all statements in Conformance Points, JAX-WS Conformance Points and Annotation Conformance Points related to C contents and annotations, notably all mandatory statements have to be satisfied.
- 2122
- 2123
- Deleted: Conformance Items
- Deleted:

2124

11.5 WSDL Files

- 2125 A WSDL conforms to this specification if it meets the following conditions:
- 2126 | 1. It is a valid WSDL 1.1 [WSDL11] document.
- 2127 | 2. It MUST comply with all statements in Conformance Points, JAX-WS Conformance Points and WSDL Extension Conformance Points related to WSDL contents and extensions, notably all mandatory statements have to be satisfied.
- 2128
- 2129
- Deleted: Conformance Items

2130

A C SCA Annotations

2131 To allow developers to define SCA related information directly in source files, without having to separately
2132 author SCDL files, a set of annotations is defined. If SCA annotations are supported by an
2133 implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as
2134 described. The SCA runtime MUST only process the SCDL files and not the annotations.[CA0001]

Formatted: Pattern: Clear
(Gray-20%)

Deleted: 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.

2135

A.1 Application of Annotations to C Program Elements

2136 In general an annotation immediately precedes the program element it applies to. If multiple annotations
2137 apply to a program element, all of the annotations SHOULD be in the same comment block.[CA0002]

Formatted: Pattern: Clear
(Gray-10%)

Deleted: If multiple annotations apply to a program element, all of the annotations SHOULD be in the same comment block.

- 2138 • Function or Function Prototype
2139 The annotation immediately precedes the function definition or declaration.

2140 Example:

```
2141 /* @OneWay */  
2142 reportEvent(int eventID);
```

- 2143 • Variable
2144 The annotation immediately precedes the variable definition.

2145 Example:

```
2146 /* @Property */  
2147 long loanType;
```

- 2148 • Set of Functions Implementing a Service
2149 A set of functions implementing a service begins with an @Service annotations. Any
2150 annotations applying to this service as a whole immediately precede the @Service
2151 annotation. These annotations SHOULD be in the same comment block as the @Service
2152 annotation.

2153 Example:

```
2154 /* @Scope("composite")  
2155 * @Service(name="LoanService", interfaceHeader="loan.h") */
```

- 2156 • Set of Function Prototypes Defining an Interface
2157 To avoid any ambiguity about the application of an annotation to a specific function or
2158 the set of functions defining an interface, if an annotation is to apply to the interface as
2159 a whole, then the @Interface annotation is used, even in the case where there is just
2160 one interface defined in a header file. Any annotations applying to the interface
2161 immediately precede the @Interface annotation.

```
2162 /* @Remoteable  
2163 * @Interface(name="LoanService" */
```

2164

A.2 Interface Header Annotations

2165 This section lists the annotations that can be used in the header file that defines a service interface.

2166

A.2.1 @Interface

2167 Annotation that indicates the start of a new interface definition. An SCA implementation MUST treat a file
2168 with a @WebService annotation specified as if @Interface was specified with the name value of the
2169 @WebService annotation used as the name value of the @Interface annotation. [CA0003]

Deleted: Specification

Deleted: 19 March

2171
2172 **Corresponds to:** *interface.c* element
2173
2174 **Format:**
2175

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


2176 where
2177 • **name : NCName (1..1)** – specifies the name of the service.
2178
2179 **Applies to:** Set of functions defining an interface.
2180 Function declarations following this annotation form the definition of this interface. This annotation also
2181 serves to bound the scope of the remaining annotations in this section,
2182
2183 **Example:**
2184 **Interface header:**
2185

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


2186
2187 **Service definition:**
2188

```
<service name="LoanService">  
2189   <interface.c header="loans.h" />  
2190 </service>
```

2191 **A.2.2 @Operation**

2192 Annotation that indicates that a function defines an operation of a service. There are two formats for this
2193 annotation depending on if the service is implemented as a set of subroutines or in a program. An SCA
2194 implementation MUST treat a function with a @WebFunction annotation specified, unless the exclude
2195 value of the @WebFunction annotation is true, as if @Operation was specified with the operationName
2196 value of the @WebFunction annotation used as the name value of the @Operation annotation. [CA0004]
2197 An SCA implementation MUST treat a struct with a @WebOperation annotation specified, unless the
2198 exclude value of the @WebOperation annotation is true, as if @Operation was specified with the struct as
2199 the input value, the operationName value of the @WebOperation annotation used as the name value of
2200 the @Operation annotation and the response value of the @WebOperation annotation used as the output
2201 values of the @Operation annotation. [CA0005]

2202
2203
2204 **Corresponds to:** *function* child element of an *interface.c* element
2205
2206 If the service is implemented as a set of subroutines, this format is used.
2207

2208 **Format:**
2209

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


2210 where
2211 • **name : NCName (0..1)** – gives the operation a different name than the function name.
2212

2213 **Applies to (library based implementations):** Function declaration
2214 The function declaration following this annotation defines an operation of the current service. If no
2215 @Operation annotation exists in an interface definition, all the function declarations in a header file or

Deleted: Specification
Deleted: 19 March

2216 following an @Interface annotation define the operations of a service, otherwise only the annotated
2217 function declarations define operations for the service.

2218
2219 Example:

2220 Interface header (loans.h):

```
2221 short internalFcn(char *param1, short param2);  
2222  
2223 /* @Operation(name="getRate") */  
2224 void rateFcn(char *cust, float *rate);
```

2225
2226 Interface definition:

```
2227 <interface.c header="loans.h">  
2228   <functions name="getRate" />  
2229 </interface.c>
```

Deleted: operation

2230
2231 If the service is implemented in a program, the following format is used. In this format, all operations are
2232 be defined via annotations.

2233
2234 Format:

```
2235 /* @Operation(name="operationName", input="inputStuct", output="outputStruct")  
2236 */
```

2237 where

- 2238 • **name: NCName (1..1)** – specifies the name of the operation.
- 2239 • **input : NCName (1..1)** – specifies the name of a struct that defines the format of the input message.
- 2240 • **output : NCName (0..1)** – specifies the name of a struct that defined the format of the output
2241 message if one is used.

2242
2243 Applies to (program based implementations): stuct declarations
2244

2245 Example:

2246 Interface header (loans.h):

```
2247 /* @Operation(name="getRate", input="rateInput", output="rateOutput") */  
2248 struct rateInput {  
2249     char cust[25];  
2250     int term;  
2251 };  
2252 struct rateOutput {  
2253     float rate;  
2254     int rateClass;  
2255 };
```

2256
2257 Interface definition:

```
2258 <interface.c header="loans.h">  
2259   <function name="getRate" input="rateInput" output="rateOutput"/>  
2260 </interface.c>
```

Deleted: operation

2261 **A.2.3 @Remotable**

2262 Annotation on service interface to indicate that a service is remotable.

2263
2264

Deleted: Specification

Deleted: 19 March

2264 **Corresponds to:** `@remotable="true"` attribute of an *interface.c* element.

2265

2266 **Format:**

2267 `/* @Remotable */`

2268 The default is **false** (not remotable).

2269

2270 **Applies to:** Interface

2271

2272 Example:

2273 Interface header (LoanService.h):

2274 `/* @Remotable */`

2275

2276 Service definition:

2277 `<service name="LoanService">`
2278 `<interface.c header="LoanService.h" remotable="true" />`
2279 `</service>`

2280 **A.2.4 @Callback**

2281 Annotation on a service interface to specify the callback interface.

2282

2283 **Corresponds to:** `@callbackHeader` attribute of an *interface.c* element.

2284

2285 **Format:**

2286 `/* @Callback(header="headerName") */`

2287 where

- 2288 • **header : Name (1..1)** – specifies the name of the header defining the callback service interface.

2289

2290 **Applies to:** Interface

2291

2292 Example:

2293 Interface header (MyService.h):

2294 `/* @Callback(header="MyServiceCallback.h") */`

2295

2296 Service definition:

2297 `<service name="MyService">`
2298 `<interface.c header="MyService.h" callbackHeader="MyServiceCallback.h" />`
2299 `</service>`

2300 **A.2.5 @OneWay**

2301 Annotation on a service interface function declaration to indicate the function is one way. The `@OneWay`
2302 annotation also affects the representation of a service in WSDL. See `@OneWay`.

2303

2304

2305 **Corresponds to:** `@oneWay="true"` attribute of function element of an *interface.c* element.

2306

Deleted: @OneWay

Deleted: Specification

Deleted: 19 March

2307 **Format:**

2308 `/* @OneWay */`

2309 The default is **false** (not OneWay).

2310

2311 **Applies to:** Function Prototype

2312

2313 **Example:**

2314 Interface header:

2315 `/* @OneWay */`

2316 `reportEvent(int eventID);`

2317

2318 Service definition:

2319 `<service name="LoanService">`

2320 `<interface.c header="LoanService.h">`

2321 `<function name="reportEvent" oneWay="true" />`

2322 `</interface.c>`

2323 `</service>`

2324 **A.3 Implementation Annotations**

2325 This section lists the annotations that can be used in the file that implements a service.

2326 **A.3.1 @ComponentType**

2327 Annotation used to indicate the start of a new componentType.

2328

2329 **Corresponds to:** *@componentType* attribute of an *implementation.c* element.

2330

2331 **Format:**

2332 `/* @ComponentType */`

2333

2334 **Applies to:** Set of services, references and properties

2335

2336 **Example:**

2337 Implementation:

2338 `/* @ComponentType */`

2339

2340 Component definition:

2341 `<component name="LoanService">`

2342 `<implementation.c module="loan" componentType="LoanService" />`

2343 `</component>`

2344 **A.3.2 @Service**

2345 Annotation that indicates the start of a new service implementation.

2346

2347 **Corresponds to:** *implementation.c* element

2348

2349 **Format:**

SCA Client and Implementation Model for C Specification Version 1.1
Copyright © OASIS® 2007, 2009. All Rights Reserved.

30 April 2009
Page 69 of 121

Deleted: Specification

Deleted: 19 March

2350 `/* @Service(name="serviceName", interfaceHeader="headerFile") */`
2351 where
2352 • **name : NCName (1..1)** – specifies the name of the service.
2353 • **interfaceHeader : Name (1..1)** – specifies the C header defining the interface.
2354
2355 **Applies to:** Set of functions implementing a service
2356 Function definitions following this annotation form the implementation of this service. This annotation also
2357 serves to bound the scope of the remaining annotations in this section,

2358
2359 Example:
2360 Implementation:
2361 `/* @Service(name="LoanService", interfaceHeader="loan.h") */`
2362
2363 ComponentType definition:
2364 `<componentType name="LoanService">`
2365 `<service name="LoanService">`
2366 `<interface.c header="loans.h" />`
2367 `</service>`
2368 `</componentType>`

2369 **A.3.3 @Reference**

2370 Annotation on a service implementation to indicate it depends on another service providing a specified
2371 interface.

2372
2373 **Corresponds to:** *reference* element of a *componentType* element.
2374

2375 **Format:**
2376 `/* @Reference(name="referenceName", interfaceHeader="headerFile",`
2377 `required="true", multiple="true")`
2378 `*/`

2379 where
2380 • **name : NCName (1..1)** – specifies the name of the reference.
2381 • **interfaceHeader : Name (1..1)** – specifies the C header defining the interface.
2382 • **required : boolean (0..1)** – specifies whether a value has to be set for this reference. Default is **true**.
2383 • **multiple : boolean (0..1)** – specifies whether this reference can be wired to multiple services. Default
2384 is **false**.

2385
2386 The multiplicity of the reference is determined from the **required** and **multiple** attributes. If the value of
2387 the **multiple** attribute is true, then component type has a reference with a multiplicity of either 0..n or 1..n
2388 depending on the value of the **required** attribute – 1..n applies if **required=true**. Otherwise a multiplicity
2389 of 0..1 or 1..1 is implied.

2390
2391 **Applies to:** Service
2392

2393 Example:
2394 Implementation:

Deleted: Specification

Deleted: 19 March

```
2395  /* @Reference(name="getRate", interfaceHeader="rates.h") */
2396
2397  /* @Reference(name="publishRate", interfaceHeader="myRates.h",
2398             required="false", multiple="yes")
2399  */
```

2400

2401 ComponentType definition:

```
2402  <componentType name="LoanService">
2403    <reference name="getRate">
2404      <interface.c header="rates.h">
2405    </reference>
2406    <reference name="publishRate" multiplicity="0..n">
2407      <interface.c header="myRates.h">
2408    </reference>
2409  </componentType>
```

2410 **A.3.4 @Property**

2411 Annotation on a service implementation to define a property of the service. Should immediately precedes
2412 the global variable that the property is based on. The variable declaration is only used for determining the
2413 type of the property. The variable will not be populated with the property value at runtime. Programs use
2414 the SCAProperty<Type>() functions for accessing property data.

2415

2416 **Corresponds to:** *property* element of a *componentType* element.

2417

2418 **Format:**

```
2419  /* @Property(name="propertyName", type="typeName",
2420             default="defaultValue", required="true")
2421  */
```

2422 where

- 2423 • **name : NCName (0..1)** – specifies the name of the property. If name is not specified the property
2424 name is taken from the name of the global variable.
- 2425 • **type : QName (0..1)** – specifies the type of the property. If not specified the type of the property is
2426 based on the C mapping of the type of the following global variable to an xsd type as defined in [Data](#)
2427 [Binding](#). If the variable is an array, then the property is many-valued.
- 2428 • **required : boolean (0..1)** – specifies whether a value has to be set in the component definition for
2429 this property. Default is **false**.
- 2430 • **default : <type> (0..1)** – specifies a default value and is only needed if **required** is **false**.

2431

2432 **Applies to:** Variable

2433

2434 **Example:**

2435 Implementation:

```
2436  /* @Property */
2437  long loanType;
2438
```

2439 ComponentType definition:

```
2440  <componentType name="LoanService">
2441    <property name="loanType" type="xsd:int" />
2442  </componentType>
```

Deleted: Data Binding

Deleted: Specification

Deleted: 19 March

2443 **A.3.5 @Scope**

2444 Annotation on a service implementation to indicate the scope of the service.

2445

2446 **Corresponds to:** @scope attribute of an *implementation.c* element.

2447

2448 **Format:**

2449 `/* @Scope("value") */`

2450 where

- 2451 • **value : [stateless | composite] (1..1)** – specifies the scope of the implementation. The default value
2452 is stateless.

2453

2454 **Applies to:** Service

2455

2456 Example:

2457 Implementation:

2458 `/* @Scope("composite") */`

2459

2460 Component definition:

2461 `<component name="LoanService">`
2462 `<implementation.c module="loan" componentType="LoanService"`
2463 `scope="composite" />`
2464 `</component>`

2465 **A.3.6 @Init**

2466 Annotation on a service implementation to indicate a function to be called when the service is
2467 instantiated. If the service is implemented in a program, this annotation indicates the program is to be
2468 called with an initialization flag prior to the first operation.

2469

2470 **Corresponds to:** @init="true" attribute of an *implementation.c* element or a *function* child element of an
2471 *implementation.c* element.

2472

2473 **Format:**

2474 `/* @Init */`

2475 The default is **false** (the function is not to be called on service initialization).

2476

2477 **Applies to:** Function or Service

2478

2479 Example:

2480 Implementation:

2481 `/* @Init */`
2482 `void init();`

2483

2484 Component definition:

2485 `<component name="LoanService">`
2486 `<implementation.c module="loan" componentType="LoanService">`
2487 `<function name="init" init="true" />`

Deleted: Specification
Deleted: 19 March

2488

2489

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

2490

A.3.7 @Destroy

2491

2492

2493

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

2494

2495

2496

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

2497

2498

2499

Format:

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

2500

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

2501

2502

Applies to: Function or Service

2503

2504

2505

2506

2507

Example:
Implementation:

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

2508

2509

2510

2511

2512

2513

2514

Component definition:

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

2515

A.3.8 @EagerInit

2516

2517

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

2518

2519

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

2520

2521

2522

Format:

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

2523

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

2524

2525

Applies to: Service

2526

2527

2528

2529

Example:
Implementation:

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

2530

2531

Component definition:

Deleted: Specification

Deleted: 19 March

```
2532 <component name="LoanService">
2533   <implementation.c module="loan" componentType="LoanService"
2534     eagerInit="true" />
2535 </component>
```

2536 **A.3.9 @AllowsPassByReference**

2537 Annotation on service implementation or operation to indicate that a service or operation allows pass by
2538 reference semantics.

2539
2540 **Corresponds to:** `@allowsPassByReference="true"` attribute of an *implementation.c* element or a *function*
2541 child element of an *implementation.c* element.

2542
2543 **Format:**

```
2544 /* @AllowsPassByReference */
```

2545 The default is **false** (the service does not allow by reference parameters).

2546
2547 **Applies to:** Service or Function

2548
2549 **Example:**

```
2550 Implementation:
2551 /* @Service(name="LoanService")
2552  * @AllowsPassByReference
2553  */
```

2554
2555 **Component definition:**

```
2556 <component name="LoanService">
2557   <implementation.c module="loan" componentType="LoanService"
2558     allowsPassByReference="true" />
2559 </component>
```

2560 **A.4 Base Annotation Grammar**

2561 While annotations are defined using the `/* ... */` format for comments, if the `// ...` format is supported by a
2562 C compiler, the `// ...` format MAY be supported by an SCA implementation annotation processor.
2563 [CA0006]

```
2564  
2565 <annotation> ::= /* @<baseAnnotation> */
2566  
2567 <baseAnnotation> ::= <name> [( <params> )]
2568  
2569 <params> ::= <paramNameValue>[, <paramNameValue>]* |
2570             <paramValue>[, <paramValue>]*
2571  
2572 <paramNameValue> ::= <name>="<value>"
2573  
2574 <paramValue> ::= "<value>"
2575  
2576 <name> ::= NCName
2577  
2578 <value> ::= string
```

- 2579 • Adjacent string constants are concatenated
2580 • NCName is as defined by XML schema **[XSD]**

Deleted: 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 annotation processor.

- Deleted:** [XSD]
- Deleted:** Specification
- Deleted:** 19 March

- 2581
- 2582
- 2583
- Whitespace including newlines between tokens is ignored.
 - Annotations with parameters can span multiple lines within a comment, and are considered complete when the terminating “)” is reached.

Deleted: Specification

Deleted: 19 March

2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629

B C SCA Policy Annotations

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

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

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

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

Deleted: concerned first
Deleted: . Secondly, there are further specific
Deleted: that deal with particular
Deleted: for certain policy domains such as Security

B.1 General Intent Annotations

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

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

Deleted: ¶
¶

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

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

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

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

Deleted: Specification
Deleted: 19 March

2630 Notice that, by convention, qualified intents include the qualifier as part of the name of the constant,
2631 separated by an underscore. These intent constants are defined in the file that defines an annotation for
2632 the intent (annotations for intents, and the formal definition of these constants, are covered in a following
2633 section).

2634

2635 Multiple intents (qualified or not) are expressed as separate strings within an array declaration.

2636

2637 | **Corresponds to:** @requires attribute of an interface.c, implementation.c, function or callbackFunction
2638 element.

Deleted: service, reference,
operation

2639

Deleted: property

2640 **Format:**

2641 `/* @Requires("qualifiedIntent" | {"qualifiedIntent" [, "qualifiedIntent"]}) */`

2642 where

2643 `qualifiedIntent ::= QName | QName.qualifier | QName.qualifier1.qualifier2`

2644

2645 **Applies to:** Interface, Service, Function, Function Prototype

2646

2647 Examples:

2648 Attaching the intents "confidentiality.message" and "integrity.message".

2649 `/* @Requires({CONFIDENTIALITY_MESSAGE, INTEGRITY_MESSAGE}) */`

2650

2651 A reference requiring support for confidentiality:

2652 `/* @Requires(CONFIDENTIALITY)`
2653 `* @Reference(interfaceHeader="SetBar.h") */`
2654 `void setBar(struct barType *bar);`

2655

2656 Users can also choose to only use constants for the namespace part of the QName, so that they can add
2657 new intents without having to define new constants. In that case, this definition would instead look like
2658 this:

2659

2660 `/* @Requires(SCA_PREFIX "confidentiality")`
2661 `* @Reference(interfaceHeader="SetBar.h") */`
2662 `void setBar(struct barType *bar);`

2663 B.2 Specific Intent Annotations

2664 In addition to the general intent annotation supplied by the @Requires annotation described above, there
2665 are C annotations that correspond to some specific policy intents.

2666

2667 The general form of these specific intent annotations is an annotation with a name derived from the name
2668 of the intent itself. If the intent is a qualified intent, qualifiers are supplied as an attribute to the annotation
2669 in the form of a string or an array of strings.

2670

2671 For example, the SCA confidentiality intent described in General Intent Annotations using the
2672 @Requires(CONFIDENTIALITY) intent can also be specified with the specific @Confidentiality intent
2673 annotation. The specific intent annotation for the "integrity" security intent is:

2674

2675 `/* @Integrity */`

Deleted: Specification

Deleted: 19 March

2676

2677 **Corresponds to:** @requires="<Intent>" attribute of an interface.c, implementation.c, function or
2678 callbackFunction element.

Deleted: service, reference,
operation

2679

Deleted: property

2680 **Format:**

2681 `/* @<Intent>[(qualifiers)] */`

2682 where Intent is an NCName that denotes a particular type of intent.

2683 `Intent ::= NCName`
2684 `qualifiers ::= "qualifier" | {"qualifier" [, "qualifier"] }`
2685 `qualifier ::= NCName | NCName/qualifier`

2686

2687 **Applies to:** Interface, Service, Function, Function Prototype – but see specific intents for restrictions

2688

2689 Example:

2690 `/* @ClientAuthentication({"message", "transport"}) */`

2691 This annotation attaches the pair of qualified intents: *authentication.message* and *authentication.transport*
2692 (the sca: namespace is assumed in both of these cases – "http:// docs.oasis-
2693 open.org/ns/opencsa/sca/200903").

2694

2695 The Policy Framework **[POLICY]** defines a number of intents and qualifiers. The following sections
2696 define the annotations for those intents.

2697 **B.2.1 Security Interaction**

Intent	Annotation
<u>clientAuthentication</u>	@ <u>ClientAuthentication</u>
<u>serverAuthentication</u>	@ <u>ServerAuthentication</u>
<u>mutualAuthentication</u>	@ <u>MutualAuthentication</u>
confidentiality	@Confidentiality
integrity	@Integrity

Deleted: a

2698

2699 These three intents can be qualified with

- 2700 • transport
- 2701 • message

2702 **B.2.2 Security Implementation**

Intent	Annotation	<u>Qualifiers</u>
<u>authorization</u>	@ <u>Authorization</u>	<u>fine_grain</u>

Deleted: @RunAs(role"role")

Deleted: runAs

Deleted: Allow ... [3]

Formatted: Bullets and Numbering

2703

B.2.3 Reliable Messaging

Intent	Annotation
atLeastOnce	@AtLeastOnce

Deleted: Specification

Deleted: 19 March

atMostOnce	@AtMostOnce
ordered	@Ordered
exactlyOnce	@ExactlyOnce

Deleted: O

B.2.4 Transactions

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

Formatted: Font: Bold

Deleted: L

B.2.5 Miscellaneous

Intent	Annotation	Qualifiers
SOAP	@SOAP	1_1 1_2

Formatted: Font: Bold

Deleted: JMS

... [4]

Formatted: Bullets and Numbering

Deleted: service, reference, operation

Deleted: property

B.3 Policy Set Annotations

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

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

Corresponds to: @policySets attribute of an [interface.c](#), [implementation.c](#), [function](#) or [callbackFunction](#) element.

Format:

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

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

Applies to: Interface, Service, Function, Function Prototype

Example:

SCA Client and Implementation Model for C [Specification](#) Version 1.1
Copyright © OASIS® 2007, 2009. All Rights Reserved.

30 April 2009
Page 79 of 121

Deleted: Specification

Deleted: 19 March

```
2728 /* @Reference(name="helloService", interfaceHeader="helloService.h",
2729 *           required=true)
2730 * @PolicySets({ MY_NS "WS_Encryption_Policy",
2731 *           MY_NS "WS_Authentication_Policy" }) */
2732 HelloService* helloService;
2733 ...
2734 }
```

2735
2736 In this case, the Policy Sets WS_Encryption_Policy and WS_Authentication_Policy are applied, both
2737 using the namespace defined for the constant MY_NS.

2738
2739 PolicySets satisfy intents expressed for the implementation when both are present, according to the rules
2740 defined in [POLICY].

Formatted: Bullets and Numbering

2741 **B.4 Policy Annotation Grammar Additions**

```
2742 <annotation> ::= /* @<baseAnnotation> | @<requiresAnnotation> |
2743                @<intentAnnotation> | @<policySetAnnotation> */
2744
2745 <requiresAnnotation> ::= Requires(<intents>)
2746
2747 <intents> ::= "<qualifiedIntent>" |
2748             {"<qualifiedIntent>"[, "<qualifiedIntent>"]*}
2749
2750 <qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |
2751                    <intentName>.<qualifier>.<qualifier>
2752
2753 <intentName> ::= {anyURI}NCName
2754
2755 <intentAnnotation> ::= <intent>[(<qualifiers>)]
2756
2757 <intent> ::= NCName[(param)]
2758
2759 <qualifiers> ::= "<qualifier>" | {"<qualifier>"[, "<qualifier>"]*}
2760
2761 <qualifier> ::= NCName | NCName/<qualifier>
2762
2763 <policySetAnnotation> ::= policySets(<policysets>)
2764
2765 <policySets> ::= "<policySetName>" | {"<policySetName>"[, "<policySetName>"]*}
2766
2767 <policySetName> ::= {anyURI}NCName
```

- 2768 • anyURI is as defined by XML schema [\[XSD\]](#)

Deleted: [XSD]

Formatted: Bullets and Numbering

2769 **B.5 Annotation Constants**

```
2770 <annotationConstant> ::= /* @Define <identifier> <token string> */
2771
2772 <identifier> ::= token
2773
2774 <token string> ::= "string" | "string"[ ## <token string>]
```

- 2775 • Constants are immediately expanded

Deleted: Specification

Deleted: 19 March

2776

C C WSDL Annotations

2777 To allow developers to control the mapping of C to WSDL, a set of annotations is defined. If WSDL
2778 mapping annotations are supported by an implementation, the annotations defined here MUST be
2779 supported and MUST be mapped to WSDL as described, [CC0005]

Deleted: If WSDL mapping annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to WSDL as described.

2780

C.1 Interface Header Annotations

2781

C.1.1 @WebService

2782 Annotation on a C header file indicating that it represents a web service. A second or subsequent
2783 instance of this annotation in a file, or a first instance after any function declarations indicates the start of
2784 a new service and has to contain a name value. An SCA implementation MUST treat any instance of a
2785 @Interface annotation and without an explicit @WebService annotation as if a @WebService annotation
2786 with a name value equal to the name value of the @Interface annotation and no other parameters was
2787 specified. [CC0001]

2788
2789 **Corresponds to:** javax.jws.WebService annotation in the JAX-WS specification (7.11.1)

2790
2791 **Format:**

2792

```
/* @WebService(name="portTypeName", targetNamespace="namespaceURI",  
2793 *      serviceName="WSDLServiceName", portName="WSDLPortName") */
```

2794 where

- 2795 • **name : NCName (0..1)** – specifies the name of the web service portType. The default is the root
2796 name of the header file containing the annotation.
- 2797 • **targetNamespace : anyURI (0..1)** – specifies the target namespace for the web service. The default
2798 namespace is determined by the implementation.
- 2799 • **serviceName : NCName (0..1)** – specifies the name for the associated WSDL service. The default
2800 service name is the name of the header file containing the annotation suffixed with “Service”. The
2801 name of the associated binding is also determined by the serviceName. In the case of a SOAP
2802 binding, the binding name is the name of the service suffixed with “SoapBinding”.
- 2803 • **portName : NCName (0..1)** – specifies the name for the associated WSDL port for the service. If a
2804 @WebService does not have a portName element, an SCA implementation MUST use the value
2805 associated with the name element, suffixed with “Port”, [CC0008]

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Deleted: If a @WebService does not have a portName element, an SCA implementation MUST use the value associated with the name element, suffixed with “Port”.

2806
2807 **Applies to:** Header file

2808
2809 **Example:**

2810 Input C header file (stockQuote.h):

2811

```
/* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
2812 *      serviceName="StockQuoteService") */  
2813 ...  
2814
```

2815
2816 **Generated WSDL file:**

2817

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
2818     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
2819     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
2820     xmlns:tns="http://www.example.org/"
```

Deleted: Specification

Deleted: 19 March

```
2821     targetNamespace="http://www.example.org/">
2822
2823     <portType name="StockQuote">
2824       <sca-c:bindings>
2825         <sca-c:prefix name="stockQuote"/>
2826       </sca-c:bindings>
2827     </portType>
2828
2829     <binding name="StockQuoteServiceSoapBinding">
2830       <soap:binding style="document"
2831         transport="http://schemas.xmlsoap.org/soap/http"/>
2832     </binding>
2833
2834     <service name="StockQuoteService">
2835       <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
2836         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
2837       </port>
2838     </service>
2839 </definitions>
```

2840 **C.1.2 @WebFunction**

2841 Annotation on a C function indicating that it represents a web service operation. An SCA implementation
2842 MUST treat a function annotated with an @Operation annotation and without an explicit @WebFunction
2843 annotation as if a @WebFunction annotation with with an operationName value equal to the name value
2844 of the @Operation annotation and no other parameters was specified. [CC0002]

2845

2846 **Corresponds to:** javax.jws.WebMethod annotation in the JAX-WS specification (7.11.2)

2847

2848 **Format:**

```
2849 /* @WebFunction(operationName="operation", action="SOAPAction",
2850 *               exclude="false") */
```

2851 where:

- 2852 • **operationName : NCName (0..1)** – specifies the name of the WSDL operation to associate with this
2853 function. The default is the name of the C function the annotation is applied to omitting any preceding
2854 namespace prefix and portType name.
- 2855 • **action : string (0..1)** – specifies the value associated with the soap:operation/@soapAction attribute
2856 in the resulting code. The default value is an empty string.
- 2857 • **exclude : boolean (0..1)** – specifies whether this function is included in the web service interface.
2858 The default value is “false”.

2859

2860 **Applies to:** Function.

2861

2862 **Example:**

2863 Input C header file:

```
2864 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
2865 *             serviceName="StockQuoteService") */
2866
2867 /* @WebFunction(operationName="GetLastTradePrice",
2868 *             action="urn:GetLastTradePrice") */
2869 float getLastTradePrice(const char *tickerSymbol);
2870
2871 /* @WebFunction(exclude="true") */
2872 void setLastTradePrice(const char *tickerSymbol, float value);
```

2873

Deleted: Specification

Deleted: 19 March

2874

Generated WSDL file:

2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937

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

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

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

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

  <portType name="StockQuote">
    <sca-c:bindings>
      <sca-c:prefix name="stockQuote"/>
    </sca-c:bindings>
    <operation name="GetLastTradePrice">
      <sca-c:bindings>
        <sca-c:function name="getLastTradePrice"/>
      </sca-c:bindings>
      <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
      </input>
      <output name="GetLastTradePriceResponse"
        message="tns:GetLastTradePriceResponse">
      </output>
    </operation>
  </portType>

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

Deleted: Specification

Deleted: 19 March

```

    </wsdl:operation>
  </binding>

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

```

C.1.3 @WebOperation

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

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

Format:

```

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

```

where:

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

Applies to: Struct.

Example:

Input C header file:

```

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

/* @WebOperation(operationName="GetLastTradePrice",
 *               response="getLastTradePriceResponseMsg"
 *               action="urn:GetLastTradePrice") */
struct getLastTradePriceMsg {
  char tickerSymbol[10];
} getLastTradePrice;

struct getLastTradePriceResponseMsg {
  float return;
} getLastTradePriceResponse;

```

Deleted: Specification

Deleted: 19 March

2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052

Generated WSDL file:

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

  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.example.org/"
    attributeFormDefault="unqualified"
    elementFormDefault="unqualified"
    targetNamespace="http://www.example.org/">
    <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
    <xs:element name="GetLastTradePriceResponse"
      type="tns:GetLastTradePriceResponse"/>
    <xs:simpleType name="TickerSymbolType">
      <xs:restriction base="xs:string">
        <xsd:maxLength value="9"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="GetLastTradePrice">
      <xs:sequence>
        <xs:element name="tickerSymbol" type="TickerSymbolType"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="GetLastTradePriceResponse">
      <xs:sequence>
        <xs:element name="return" type="xs:float"/>
      </xs:sequence>
    </xs:complexType>
  </xs:schema>

  < message name="GetLastTradePrice">
    <sca-c:bindings>
      <sca-c:struct name="getLastTradePrice"/>
    </sca-c:bindings>
    <part name="parameters" element="tns:GetLastTradePrice">
    </part>
  </message>

  < message name="GetLastTradePriceResponse">
    <sca-c:bindings>
      <sca-c:struct name="getLastTradePriceResponse"/>
    </sca-c:bindings>
    <part name="parameters" element="tns:GetLastTradePriceResponse">
    </part>
  </ message>

  <portType name="StockQuote">
    <sca-c:bindings>
      <sca-c:prefix name="stockQuote"/>
    </sca-c:bindings>
    <operation name="GetLastTradePrice">
      <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
      </input>
      <output name="GetLastTradePriceResponse"
        message="tns:GetLastTradePriceResponse">
      </output>
    </operation>
  </portType>

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

Deleted: Specification
Deleted: 19 March

```

3053     <wsdl:operation name="GetLastTradePrice">
3054         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3055         <wsdl:input name="GetLastTradePrice">
3056             <soap:body use="literal"/>
3057         </wsdl:input>
3058         <wsdl:output name="GetLastTradePriceResponse">
3059             <soap:body use="literal"/>
3060         </wsdl:output>
3061     </wsdl:operation>
3062 </binding>
3063
3064     <service name="StockQuoteService">
3065         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3066             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3067         </port>
3068     </service>
3069 </definitions>

```

C.1.4 @OneWay

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

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

Format:

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

Applies to: Function.

Example:

Input C header file:

```

3083 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3084  *             serviceName="StockQuoteService") */
3085
3086 /* @WebFunction(operationName="SetTradePrice",
3087  *             action="urn:SetTradePrice")
3088  * @OneWay */
3089 void setTradePrice(const char *tickerSymbol, float price);

```

Generated WSDL file:

```

3092 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3093             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3094             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3095             xmlns:tns="http://www.example.org/"
3096             targetNamespace="http://www.example.org/">
3097
3098     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3099               xmlns:tns="http://www.example.org/"
3100               attributeFormDefault="unqualified"
3101               elementFormDefault="unqualified"
3102               targetNamespace="http://www.example.org/">
3103         <xs:element name="SetTradePrice" type="tns:SetTradePrice"/>
3104         <xs:complexType name="SetTradePrice">
3105             <xs:sequence>
3106                 <xs:element name="tickerSymbol" type="xs:string"/>
3107                 <xs:element name="price" type="xs:float"/>

```

Deleted: Specification

Deleted: 19 March

```

3108     </xs:sequence>
3109   </xs:complexType>
3110 </xs:schema>
3111
3112   < message name="SetTradePrice">
3113     <part name="parameters" element="tns:SetTradePrice">
3114       </part>
3115   </message>
3116
3117   <portType name="StockQuote">
3118     <sca-c:bindings>
3119       <sca-c:prefix name="stockQuote"/>
3120     </sca-c:bindings>
3121     <operation name="SetTradePrice">
3122       <sca-c:bindings>
3123         <sca-c:function name="setTradePrice"/>
3124       </sca-c:bindings>
3125       <input name="SetTradePrice" message="tns:SetTradePrice">
3126         </input>
3127     </operation>
3128   </portType>
3129
3130   <binding name="StockQuoteServiceSoapBinding">
3131     <soap:binding style="document"
3132       transport="http://schemas.xmlsoap.org/soap/http"/>
3133     <wsdl:operation name="SetTradePrice">
3134       <soap:operation soapAction="urn:SetTradePrice" style="document"/>
3135       <wsdl:input name="SetTradePrice">
3136         <soap:body use="literal"/>
3137       </wsdl:input>
3138     </wsdl:operation>
3139   </binding>
3140
3141   <service name="StockQuoteService">
3142     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3143       <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3144     </port>
3145   </service>
3146 </definitions>

```

3147 C.1.5 @WebParam

3148 Annotation on a C function indicating the mapping of a parameter to the associated input and output
 3149 WSDL messages. Or on a C struct indicating the mapping of a member to the associated WSDL
 3150 message.

3151

3152 **Corresponds to:** javax.jws.WebParam annotation in the JAX-WS specification (7.11.4)

3153

3154 **Format:**

```

3155   /* @WebParam(paramName="parameter", name="WSDLElement",
3156      *         targetNamespace="namespaceURI", mode="IN"|"OUT"|"INOUT",
3157      *         header="false", partName="WSDLPart", type="xsdType") */

```

3158 where:

- 3159 • **paramName : NCName (1..1)** – specifies the name of the parameter that this annotation applies to.
 3160 Only named parameters MAY be referenced by a @WebParam annotation. [CC0009]
- 3161 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default
 3162 value is the name of the parameter. If an @WebParam annotation is not present, and the parameter
 3163 is unnamed, then a name of "argN", where N is an incrementing value from 1 indicating the position of
 3164 he parameter in the argument list, will be used.

Deleted: Specification

Deleted: 19 March

- 3165 • **targetNamespace** : *string (0..1)* – specifies the target namespace for the part. The default
3166 namespace is the namespace of the associated @WebService. The targetNamespace attribute is
3167 ignored unless the binding style is document, and the binding parameterStyle is bare. See
3168 [@SOAPBinding](#).
- 3169 • **mode** : *token (0..1)* – specifies whether the parameter is associated with the input message, output
3170 message, or both. The default value is determined by the passing mechanism for the parameter. See
3171 [Method Parameters and Return Type](#).
- 3172 • **header** : *boolean (0..1)* – specifies whether this parameter is associated with a SOAP header
3173 element. The default value is "false".
- 3174 • **partName** : *NCName (0..1)* – specifies the name of the WSDL part associated with this item. The
3175 default value is the value of name.
- 3176 • **type** : *QName (0..1)* – specifies the XML Schema type of the WSDL part or element associated with
3177 this parameter. [The value of the type property of a @WebParam annotation MUST be either one of](#)
3178 [the simpleTypes defined in namespace](#)
3179 [http://www.w3.org/2001/XMLSchema](#) [http://www.w3.org/2001/XMLSchema](#) or, if the type of the
3180 [parameter is a struct, the QName of a XSD complex type following the mapping specified in Complex](#)
3181 [Content Binding](#), [CC0006] The default type is determined by the mapping defined in [Data Binding](#).

Deleted: @SOAPBinding

Deleted: Method Parameters and
Return Type

Deleted: NCName

Deleted: The value of the type
property of a @WebParam annotation
MUST be one of the simpleTypes
defined in namespace
[http://www.w3.org/2001/XMLSchema](#).

Deleted: Simple Content Binding

3183 **Applies to:** Function parameter or struct member.

3184 **Example:**

3185 Input C header file:

```
3187 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
3188 *      serviceName="StockQuoteService") */  
3189  
3190 /* @WebFunction(operationName="GetLastTradePrice",  
3191 *      action="urn:GetLastTradePrice")  
3192 * @WebParam(paramName="tickerSymbol", name="symbol", mode="IN") */  
3193 float getLastTradePrice(char *tickerSymbol);
```

3195 Generated WSDL file:

```
3196 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3197     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3198     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
3199     xmlns:tns="http://www.example.org/"  
3200     targetNamespace="http://www.example.org/">  
3201  
3202     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
3203         xmlns:tns="http://www.example.org/"  
3204         attributeFormDefault="unqualified"  
3205         elementFormDefault="unqualified"  
3206         targetNamespace="http://www.example.org/">  
3207         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
3208         <xs:element name="GetLastTradePriceResponse"  
3209             type="tns:GetLastTradePriceResponse"/>  
3210         <xs:complexType name="GetLastTradePrice">  
3211             <xs:sequence>  
3212                 <xs:element name="symbol" type="xs:string"/>  
3213             </xs:sequence>  
3214         </xs:complexType>  
3215         <xs:complexType name="GetLastTradePriceResponse">  
3216             <xs:sequence>  
3217                 <xs:element name="return" type="xs:float"/>  
3218             </xs:sequence>  
3219         </xs:complexType>  
3220     </xs:schema>
```

Deleted: Specification

Deleted: 19 March

```
3221 < message name="GetLastTradePrice">
3222   <part name="parameters" element="tns:GetLastTradePrice">
3223   </part>
3224 </message>
3225
3226
3227 < message name="GetLastTradePriceResponse">
3228   <part name="parameters" element="tns:GetLastTradePriceResponse">
3229   </part>
3230 </ message>
3231
3232 <portType name="StockQuote">
3233   <sca-c:bindings>
3234     <sca-c:prefix name="stockQuote"/>
3235   </sca-c:bindings>
3236   <operation name="GetLastTradePrice">
3237     <sca-c:bindings>
3238       <sca-c:function name="getLastTradePrice"/>
3239       <sca-c:parameter name="tickerSymbol"
3240         part="tns:GetLastTradePrice/parameter"
3241         childElementName="symbol"/>
3242     </sca-c:bindings>
3243     <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3244     </input>
3245     <output name="GetLastTradePriceResponse"
3246       message="tns:GetLastTradePriceResponse">
3247     </output>
3248   </operation>
3249 </portType>
3250
3251 <binding name="StockQuoteServiceSoapBinding">
3252   <soap:binding style="document"
3253     transport="http://schemas.xmlsoap.org/soap/http"/>
3254   <wsdl:operation name="GetLastTradePrice">
3255     <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3256     <wsdl:input name="GetLastTradePrice">
3257       <soap:body use="literal"/>
3258     </wsdl:input>
3259     <wsdl:output name="GetLastTradePriceResponse">
3260       <soap:body use="literal"/>
3261     </wsdl:output>
3262   </wsdl:operation>
3263 </binding>
3264
3265 <service name="StockQuoteService">
3266   <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3267     <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3268   </port>
3269 </service>
3270 </definitions>
```

3271 C.1.6 @WebResult

3272 Annotation on a C function indicating the mapping of the function's return type to the associated output
3273 WSDL message.

3274

3275 **Corresponds to:** javax.jws.WebResult annotation in the JAX-WS specification (7.11.5)

3276

3277 **Format:**

```
3278 /* @WebResult(name="WSDLElement", targetNamespace="namespaceURI",
3279 *             header="false", partName="WSDLPart", type="xsdType") */
```

Deleted: Specification

Deleted: 19 March

3280 where:

- 3281 • **name : NCName (0..1)** – specifies the name of the associated WSDL part or element. The default
3282 value is "return".
- 3283 • **targetNamespace : string (0..1)** – specifies the target namespace for the part. The default
3284 namespace is the namespace of the associated @WebService. The targetNamespace attribute is
3285 ignored unless the binding style is document, and the binding parameterStyle is bare. (See
3286 @SOAPBinding).
- 3287 • **header : boolean (0..1)** – specifies whether the result is associated with a SOAP header element.
3288 The default value is "false".
- 3289 • **partName : NCName (0..1)** – specifies the name of the WSDL part associated with this item. The
3290 default value is the value of name.
- 3291 • **type : NCName (0..1)** – specifies the XML Schema type of the WSDL part or element associated with
3292 this parameter. The value of the type property of a @WebResult annotation MUST be one of the
3293 simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema, [CC0007] The default type
3294 is determined by the mapping defined in 11.3.1.

Deleted: The value of the type property of a @WebResult annotation MUST be one of the simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema.

3296 **Applies to:** Function.

3297

3298 **Example:**

3299 **Input C header file:**

```
3300 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
3301 *      serviceName="StockQuoteService") */  
3302  
3303 /* @WebFunction(operationName="GetLastTradePrice",  
3304 *      action="urn:GetLastTradePrice")  
3305 * @WebResult(name="price") */  
3306 float getLastTradePrice(const char *tickerSymbol);
```

3307

3308 **Generated WSDL file:**

```
3309 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3310     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3311     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
3312     xmlns:tns="http://www.example.org/"  
3313     targetNamespace="http://www.example.org/">  
3314  
3315     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
3316         xmlns:tns="http://www.example.org/"  
3317         attributeFormDefault="unqualified"  
3318         elementFormDefault="unqualified"  
3319         targetNamespace="http://www.example.org/">  
3320         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>  
3321         <xs:element name="GetLastTradePriceResponse"  
3322             type="tns:GetLastTradePriceResponse"/>  
3323         <xs:complexType name="GetLastTradePrice">  
3324             <xs:sequence>  
3325                 <xs:element name="tickerSymbol" type="xs:string"/>  
3326             </xs:sequence>  
3327         </xs:complexType>  
3328         <xs:complexType name="GetLastTradePriceResponse">  
3329             <xs:sequence>  
3330                 <xs:element name="price" type="xs:float"/>  
3331             </xs:sequence>  
3332         </xs:complexType>  
3333     </xs:schema>  
3334  
3335     < message name="GetLastTradePrice">
```

Deleted: Specification

Deleted: 19 March

```
3336 <part name="parameters" element="tns:GetLastTradePrice">
3337 </part>
3338 </message>
3339
3340 < message name="GetLastTradePriceResponse">
3341 <part name="parameters" element="tns:GetLastTradePriceResponse">
3342 </part>
3343 </ message>
3344
3345 <portType name="StockQuote">
3346 <sca-c:bindings>
3347 <sca-c:prefix name="stockQuote"/>
3348 </sca-c:bindings>
3349 <operation name="GetLastTradePrice">
3350 <sca-c:bindings>
3351 <sca-c:function name="getLastTradePrice"/>
3352 </sca-c:bindings>
3353 <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3354 </input>
3355 <output name="GetLastTradePriceResponse"
3356 message="tns:GetLastTradePriceResponse">
3357 </output>
3358 </operation>
3359 </portType>
3360
3361 <binding name="StockQuoteServiceSoapBinding">
3362 <soap:binding style="document"
3363 transport="http://schemas.xmlsoap.org/soap/http"/>
3364 <wsdl:operation name="GetLastTradePrice">
3365 <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3366 <wsdl:input name="GetLastTradePrice">
3367 <soap:body use="literal"/>
3368 </wsdl:input>
3369 <wsdl:output name="GetLastTradePriceResponse">
3370 <soap:body use="literal"/>
3371 </wsdl:output>
3372 </wsdl:operation>
3373 </binding>
3374
3375 <service name="StockQuoteService">
3376 <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3377 <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3378 </port>
3379 </service>
3380 </definitions>
```

3381 **C.1.7 @SOAPBinding**

3382 Annotation on a C WebService or function specifying the mapping of the web service onto the SOAP
3383 message protocol.

3385 **Corresponds to:** javax.jws.SOAPBinding annotation in the JAX-WS specification (7.11.6)

3387 **Format:**

```
3388 /* @SOAPBinding(style="DOCUMENT"|"RPC", use="LITERAL"|"ENCODED",
3389 * parameterStyle="BARE"|"WRAPPED") */
```

3390 where:

- 3391 • **style : token (0..1)** – specifies the WSDL binding style. The default value is “DOCUMENT”.
- 3392 • **use : token (0..1)** – specifies the WSDL binding use. The default value is “LITERAL”.

Deleted: Specification

Deleted: 19 March

3393 • **parameterStyle : token (0..1)** – specifies the WSDL parameter style. The default value is
3394 “WRAPPED”.

3396 **Applies to:** WebService, Function.

3398 **Example:**

3399 **Input C header file:**

```
3400 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",  
3401 *           serviceName="StockQuoteService") */  
3402 * @SOAPBinding(style="RPC") */  
3403 ...  
3404
```

3406 **Generated WSDL file:**

```
3407 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"  
3408             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
3409             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"  
3410             xmlns:tns="http://www.example.org/"  
3411             targetNamespace="http://www.example.org/">  
3412  
3413     <portType name="StockQuote">  
3414         <sca-c:bindings>  
3415             <sca-c:prefix name="stockQuote"/>  
3416         </sca-c:bindings>  
3417     </portType>  
3418  
3419     <binding name="StockQuoteServiceSoapBinding">  
3420         <soap:binding style="rpc"  
3421             transport="http://schemas.xmlsoap.org/soap/http"/>  
3422     </binding>  
3423  
3424     <service name="StockQuoteService">  
3425         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">  
3426             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>  
3427         </port>  
3428     </service>  
3429 </definitions>
```

3430 C.1.8 @WebFault

3431 Annotation on a C struct indicating that it format of a fault message.

3433 **Corresponds to:** javax.xml.ws.WebFault annotation in the JAX-WS specification (7.2)

3435 **Format:**

```
3436 /* @WebFault(name="WSDLElement", targetNamespace="namespaceURI") */
```

3437 **where:**

- 3438 • **name : NCName (1..1)** – specifies the local name of the global element mapped to this fault.
- 3439 • **targetNamespace : string (0..1)** – specifies the namespace of the global element mapped to this
3440 fault. The default namespace is determined by the implementation.

3442 **Applies to:** struct.

3443

Deleted: Specification

Deleted: 19 March

3444 Example:

3445 Input C header file:

```
3446 /* @WebFault (name="UnknownSymbolFault",
3447 *      targetNamespace="http://www.example.org/")
3448 struct UnkSymMsg {
3449     char faultInfo[10];
3450 } unkSymInfo;
3451
3452 /* @WebService (name="StockQuote", targetNamespace="http://www.example.org/",
3453 *      serviceName="StockQuoteService") */
3454
3455 /* @WebFunction (operationName="GetLastTradePrice",
3456 *      action="urn:GetLastTradePrice")
3457 * @WebThrows (faults="unkSymMsg") */
3458 float getLastTradePrice(const char *tickerSymbol);
```

Deleted: Info

3459

3460 Generated WSDL file:

```
3461 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3462     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3463     xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3464     xmlns:tns="http://www.example.org/"
3465     targetNamespace="http://www.example.org/"
3466
3467     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3468         xmlns:tns="http://www.example.org/"
3469         attributeFormDefault="unqualified"
3470         elementFormDefault="unqualified"
3471         targetNamespace="http://www.example.org/"
3472         <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3473         <xs:element name="GetLastTradePriceResponse"
3474             type="tns:GetLastTradePriceResponse"/>
3475         <xs:complexType name="GetLastTradePrice">
3476             <xs:sequence>
3477                 <xs:element name="tickerSymbol" type="xs:string"/>
3478             </xs:sequence>
3479         </xs:complexType>
3480         <xs:complexType name="GetLastTradePriceResponse">
3481             <xs:sequence>
3482                 <xs:element name="return" type="xs:float"/>
3483             </xs:sequence>
3484         </xs:complexType>
3485         <xs:simpleType name="UnknownSymbolFaultType">
3486             <xs:restriction base="xs:string">
3487                 <xsd:maxLength value="9"/>
3488             </xs:restriction>
3489         </xs:simpleType>
3490         <xs:element name="UnknownSymbolFault" type="UnknownSymbolFaultType"/>
3491     </xs:schema>
3492
3493     <message name="GetLastTradePrice">
3494         <part name="parameters" element="tns:GetLastTradePrice">
3495         </part>
3496     </message>
3497
3498     <message name="GetLastTradePriceResponse">
3499         <part name="parameters" element="tns:GetLastTradePriceResponse">
3500         </part>
3501     </message>
3502
3503     <message name="UnknownSymbol">
3504         <sca-c:bindings>
3505             <sca-c:struct name="unkSymMsg"/>
```

Deleted: Specification

Deleted: 19 March

Deleted: Info

```
3506     </sca-c:bindings>
3507     <part name="parameters" element="tns:UnknownSymbolFault">
3508     </part>
3509 </message>
3510
3511 <portType name="StockQuote">
3512     <sca-c:bindings>
3513         <sca-c:prefix name="stockQuote"/>
3514     </sca-c:bindings>
3515     <operation name="GetLastTradePrice">
3516         <sca-c:bindings>
3517             <sca-c:function name="getLastTradePrice"/>
3518         </sca-c:bindings>
3519         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3520         </input>
3521         <output name="GetLastTradePriceResponse"
3522             message="tns:GetLastTradePriceResponse">
3523         </output>
3524         <fault name="UnknownSymbol" message="tns:UnknownSymbol">
3525         </fault>
3526     </operation>
3527 </portType>
3528
3529 <binding name="StockQuoteServiceSoapBinding">
3530     <soap:binding style="document"
3531         transport="http://schemas.xmlsoap.org/soap/http"/>
3532     <wsdl:operation name="GetLastTradePrice">
3533         <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3534         <wsdl:input name="GetLastTradePrice">
3535             <soap:body use="literal"/>
3536         </wsdl:input>
3537         <wsdl:output name="GetLastTradePriceResponse">
3538             <soap:body use="literal"/>
3539         </wsdl:output>
3540         <wsdl:fault>
3541             <soap:fault name="UnknownSymbol" use="literal"/>
3542         </wsdl:fault>
3543     </wsdl:operation>
3544 </binding>
3545
3546 <service name="StockQuoteService">
3547     <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3548         <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3549     </port>
3550 </service>
3551 </definitions>
```

3552 **C.1.9 @WebThrows**

3553 Annotation on a C function or operation indicating which faults might be thrown by this function or
3554 operation.

3555

3556 **Corresponds to:** No equivalent in JAX-WS.

3557

3558 **Format:**

3559 `/* @WebThrows (faults="faultMsg1" [, "faultMsgn"]*) */`

3560 where:

- 3561 • **faults : NMTOKEN (1..n)** – specifies the names of all faults that might be thrown by this function or
3562 operation. The name of the fault is the name of its associated C struct name. A C struct that is listed
3563 in a @WebThrows annotation **MUST** itself have a @WebFault annotation. **[CC0004]**

Deleted: Specification

Deleted: 19 March

3564
3565 **Applies to:** Function or Operation
3566
3567 Example:
3568 | See [@WebFault](#).

Deleted: @WebFault

Deleted: Specification
Deleted: 19 March

3569

D C WSDL Mapping Extensions

3570 The following WSDL extensions are used to augment the conversion process from WSDL to C. All of
3571 these extensions are defined in the namespace `http://docs.oasis-open.org/ns/opencsa/sca-c-`
3572 `cpp/c/200901`. For brevity, all definitions of these extensions will be fully qualified, and all references to
3573 the “sca-c” prefix are associated with the namespace above. If WSDL extensions are supported by an
3574 implementation, all the extensions defined here MUST be supported and MUST be mapped to C as
3575 described, [CD0001]

Deleted: If WSDL extensions are supported by an implementation, all the extensions defined here MUST be supported and MUST be mapped to C as described.

3576

D.1 <sca-c:bindings>

3577 <sca-c:bindings> is a container type which can be used as a WSDL extension. All other SCA wsdl
3578 extensions will be specified as children of a <sca-c:bindings> element. An <sca-c:bindings> element can
3579 be used as an extension to any WSDL type that accepts extensions.

3580

D.2 <sca-c:prefix>

3581 <sca-c:prefix> provides a mechanism for defining an alternate prefix for the functions or structs
3582 implementing the operations of a portType.

3584

Format:

3585 `<sca-c:prefix name="portTypePrefix"/>`

3586 where:

- 3587 • **prefix/@name : string (1..1)** – specifies the string to prepend to an operation name when generating
3588 a C function or structure name.

3590

Applicable WSDL element(s):

- 3591 • wsdl:portType

3593 A <sca-c:bindings/> element MUST NOT have more than one < sca-c:prefix/> child element. [CD0003]

3595 Example:

3596 Input WSDL file:

3597 `<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"`
3598 `xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"`
3599 `xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"`
3600 `xmlns:tns="http://www.example.org/"`
3601 `targetNamespace="http://www.example.org/">`
3602
3603 `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"`
3604 `xmlns:tns="http://www.example.org/"`
3605 `attributeFormDefault="unqualified"`
3606 `elementFormDefault="unqualified"`
3607 `targetNamespace="http://www.example.org/">`
3608 `<xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>`
3609 `<xs:element name="GetLastTradePriceResponse"`
3610 `type="tns:GetLastTradePriceResponse"/>`
3611 `<xs:complexType name="GetLastTradePrice">`
3612 `<xs:sequence>`
3613 `<xs:element name="tickerSymbol" type="xs:string"/>`
3614 `</xs:sequence>`
3615 `</xs:complexType>`

Deleted: Specification

Deleted: 19 March

```
3616 <xs:complexType name="GetLastTradePriceResponse">
3617 <xs:sequence>
3618 <xs:element name="return" type="xs:float"/>
3619 </xs:sequence>
3620 </xs:complexType>
3621 </xs:schema>
3622
3623 < message name="GetLastTradePrice">
3624 <part name="parameters" element="tns:GetLastTradePrice">
3625 </part>
3626 </message>
3627
3628 < message name="GetLastTradePriceResponse">
3629 <part name="parameters" element="tns:GetLastTradePriceResponse">
3630 </part>
3631 </ message>
3632
3633 <portType name="StockQuote">
3634 <sca-c:bindings>
3635 <sca-c:prefix name="stockQuote"/>
3636 </sca-c:bindings>
3637 <operation name="GetLastTradePrice">
3638 <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3639 </input>
3640 <output name="GetLastTradePriceResponse"
3641 message="tns:GetLastTradePriceResponse">
3642 </output>
3643 </operation>
3644 </portType>
3645
3646 <binding name="StockQuoteServiceSoapBinding">
3647 <soap:binding style="document"
3648 transport="http://schemas.xmlsoap.org/soap/http"/>
3649 <wsdl:operation name="GetLastTradePrice">
3650 <soap:operation soapAction="urn:GetLastTradePrice" style="document"/>
3651 <wsdl:input name="GetLastTradePrice">
3652 <soap:body use="literal"/>
3653 </wsdl:input>
3654 <wsdl:output name="GetLastTradePriceResponse">
3655 <soap:body use="literal"/>
3656 </wsdl:output>
3657 </wsdl:operation>
3658 </binding>
3659
3660 <service name="StockQuoteService">
3661 <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
3662 <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
3663 </port>
3664 </service>
3665 </definitions>
```

```
3666
3667 Generated C header file:
3668 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
3669 * serviceName="StockQuoteService") */
3670
3671 /* @WebFunction(operationName="GetLastTradePrice",
3672 * action="urn:GetLastTradePrice") */
3673 float stockQuoteGetLastTradePrice(const char *tickerSymbol);
```

Deleted: Specification
Deleted: 19 March

D.3 <sca-c:enableWrapperStyle>

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

Format:

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

where:

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

Applicable WSDL element(s):

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

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

Example:

Input WSDL file:

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

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

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

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

Deleted: Specification

Deleted: 19 March

```

3728     </part>
3729   </ message>
3730
3731   <portType name="StockQuote">
3732     <sca-c:bindings>
3733       <sca-c:prefix name="stockQuote"/>
3734       <sca-c:enableWrapperStyle>false</sca-c:enableWrapperStyle>
3735     </sca-c:bindings>
3736     <operation name="GetLastTradePrice">
3737       <sca-c:bindings>
3738         <sca-c:function name="getLastTradePrice"/>
3739       </sca-c:bindings>
3740       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3741         </input>
3742       <output name="GetLastTradePriceResponse"
3743         message="tns:GetLastTradePriceResponse">
3744         </output>
3745     </operation>
3746   </portType>
3747 </definitions>

```

Generated C header file:

```

3750 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3751  *           serviceName="StockQuoteService") */
3752
3753 /* @WebFunction(operationName="GetLastTradePrice",
3754  *           action="urn:GetLastTradePrice") */
3755 DATAOBJECT getLastTradePrice(DATAOBJECT parameters);

```

D.4 <sca-c:function>

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

Format:

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

where:

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

Applicable WSDL element(s):

- wsdl:portType/wsdl:operation

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

Example:

Input WSDL file:

```

3774 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3775   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3776   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3777   xmlns:tns="http://www.example.org/"
3778   targetNamespace="http://www.example.org/"
3779
3780   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

```

Deleted: Specification

Deleted: 19 March

```
3781      xmlns:tns="http://www.example.org/"
3782      attributeFormDefault="unqualified"
3783      elementFormDefault="unqualified"
3784      targetNamespace="http://www.example.org/">
3785      <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3786      <xs:element name="GetLastTradePriceResponse"
3787        type="tns:GetLastTradePriceResponse"/>
3788      <xs:complexType name="GetLastTradePrice">
3789        <xs:sequence>
3790          <xs:element name="tickerSymbol" type="xs:string"/>
3791        </xs:sequence>
3792      </xs:complexType>
3793      <xs:complexType name="GetLastTradePriceResponse">
3794        <xs:sequence>
3795          <xs:element name="return" type="xs:float"/>
3796        </xs:sequence>
3797      </xs:complexType>
3798    </xs:schema>
3799
3800    < message name="GetLastTradePrice">
3801      <part name="parameters" element="tns:GetLastTradePrice">
3802      </part>
3803    </message>
3804
3805    < message name="GetLastTradePriceResponse">
3806      <part name="parameters" element="tns:GetLastTradePriceResponse">
3807      </part>
3808    </ message>
3809
3810    <portType name="StockQuote">
3811      <sca-c:bindings>
3812        <sca-c:prefix name="stockQuote"/>
3813      </sca-c:bindings>
3814      <operation name="GetLastTradePrice">
3815        <sca-c:bindings>
3816          <sca-c:function name="getTradePrice"/>
3817        </sca-c:bindings>
3818        <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3819        </input>
3820        <output name="GetLastTradePriceResponse"
3821          message="tns:GetLastTradePriceResponse">
3822        </output>
3823      </operation>
3824    </portType>
3825  </definitions>
```

3826
3827 Generated C header file:

```
3828 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3829 *      serviceName="StockQuoteService") */
3830
3831 /* @WebFunction(operationName="GetLastTradePrice",
3832 *      action="urn:GetLastTradePrice") */
3833 float getTradePrice(const wchar_t *tickerSymbol);
```

3834 **D.5 <sca-c:struct>**

3835 <sca-c:struct> specifies the name of the C struct that the associated WSDL message is associated with. If
3836 <sca-c:struct> is used for an operation request or response message, the portType prefix, either default
3837 or a specified with <sca-c:prefix> is not prepended to the struct name.

3838
3839

Deleted: Specification
Deleted: 19 March

3840 **Format:**

3841 `<sca-c:struct name="myStruct"/>`

3842 where:

- 3843 • **struct@name : NCName (1..1)** – specifies the name of the C struct associated with this WSDL
3844 message.

3845

3846 **Applicable WSDL element(s):**

- 3847 • wsdl:message

3848

3849 A `<sca-c:bindings/>` element **MUST NOT** have more than one `< sca-c:struct/>` child element. [CD0006]

3850

3851 **Example:**

3852 **Input WSDL file:**

```
3853 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3854             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
3855             xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
3856             xmlns:tns="http://www.example.org/"
3857             targetNamespace="http://www.example.org/"
3858
3859     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3860               xmlns:tns="http://www.example.org/"
3861               attributeFormDefault="unqualified"
3862               elementFormDefault="unqualified"
3863               targetNamespace="http://www.example.org/"
3864               >
3865       <xs:element name="GetLastTradePrice" type="tns:GetLastTradePrice"/>
3866       <xs:element name="GetLastTradePriceResponse"
3867                 type="tns:GetLastTradePriceResponse"/>
3868       <xs:complexType name="GetLastTradePrice">
3869         <xs:sequence>
3870           <xs:element name="tickerSymbol" type="xs:string"/>
3871         </xs:sequence>
3872       </xs:complexType>
3873       <xs:complexType name="GetLastTradePriceResponse">
3874         <xs:sequence>
3875           <xs:element name="return" type="xs:float"/>
3876         </xs:sequence>
3877       </xs:complexType>
3878     </xs:schema>
3879
3880     < message name="GetLastTradePrice">
3881       <sca-c:bindings>
3882         <sca-c:struct name="getTradePrice"/>
3883       </sca-c:bindings>
3884       <part name="parameters" element="tns:GetLastTradePrice">
3885       </part>
3886     </message>
3887
3888     < message name="GetLastTradePriceResponse">
3889       <sca-c:bindings>
3890         <sca-c:struct name="getTradePriceResponse"/>
3891       </sca-c:bindings>
3892       <part name="parameters" element="tns:GetLastTradePriceResponse">
3893       </part>
3894     </ message>
3895
3896     <portType name="StockQuote">
3897       <sca-c:bindings>
3898         <sca-c:prefix name="stockQuote"/>
```

Deleted: Specification

Deleted: 19 March

```
3898     </sca-c:bindings>
3899     <operation name="GetLastTradePrice">
3900       <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
3901       </input>
3902       <output name="GetLastTradePriceResponse"
3903         message="tns:GetLastTradePriceResponse">
3904       </output>
3905     </operation>
3906   </portType>
3907 </definitions>
```

```
3908
3909 Generated C header file:
3910 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/"
3911    *      serviceName="StockQuoteService") */
3912
3913 /* @WebOperation(operationName="GetLastTradePrice",
3914    *      response="getLastTradePriceResponse"
3915    *      action="urn:GetLastTradePrice") */
3916 struct getLastTradePrice {
3917     wchar_t *tickerSymbol; /* Since the length of the element is not
3918                            * restricted, a pointer is returned with the
3919                            * actual value held by the SCA runtime. */
3920 };
3921
3922 struct getLastTradePriceResponse {
3923     float return;
3924 };
```

3925 **D.6 <sca-c:parameter>**

3926 <sca-c:parameter> specifies the name of the C function parameter or struct member associated with a
3927 specific WSDL message part or wrapper child element.

3928
3929 **Format:**

```
3930 <sca-c:parameter name="CParameter" part="WSDLPart"
3931   childElementName="WSDLElement" type="CType"/>
```

3932 where:

- 3933 • **parameter/@name : NCName (1..1)** – specifies the name of the C function parameter or struct
3934 member associated with this WSDL operation part or wrapper child element. “return” is used to
3935 denote the return value.
- 3936 • **parameter/@part : string (1..1)** - an XPath expression identifying the wsdl:part of a wsdl:message.
- 3937 • **parameter/@childElementName : QName (1..1)** – specifies the qualified name of a child element of
3938 the global element identified by parameter/@part.
- 3939 • **parameter/@type : string (0..1)** – specifies the type of the parameter or struct member or return
3940 type. The @type attribute of a <parameter/> element MUST be either a C type specified in Simple
3941 Content Binding or, if the message part has complex content, a struct following the mapping specified
3942 in Complex Content Binding.[CD0002] The default type is determined by the mapping defined in
3943 Data Binding.

Deleted: NCName

Deleted: The @type attribute of a
<parameter/> element MUST be a
valid C type.

Deleted: Data Binding

3945 **Applicable WSDL element(s):**

- 3946 • wsdl:portType/wsdl:operation

3947
3948
3949 **Example:**

Deleted: Specification

Deleted: 19 March

Input WSDL file:

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

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

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

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

  <portType name="StockQuote">
    <sca-c:bindings>
      <sca-c:prefix name="stockQuote"/>
    </sca-c:bindings>
    <operation name="GetLastTradePrice">
      <sca-c:bindings>
        <sca-c:function name="getLastTradePrice"/>
        <sca-c:parameter name="tickerSymbol"
          part="tns:GetLastTradePrice/parameter"
          childElementName="symbol"/>
      </sca-c:bindings>
      <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
      </input>
      <output name="GetLastTradePriceResponse"
        message="tns:GetLastTradePriceResponse">
      </output>
    </operation>
  </portType>

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

Deleted: Specification

Deleted: 19 March

```
4014         <wsdl:output name="GetLastTradePriceResponse">
4015             <soap:body use="literal"/>
4016         </wsdl:output>
4017     </wsdl:operation>
4018 </binding>
4019
4020     <service name="StockQuoteService">
4021         <port name="StockQuotePort" binding="tns:StockQuoteServiceSoapBinding">
4022             <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
4023         </port>
4024     </service>
4025 </definitions>
```

```
4026
4027 Generated C header file:
4028 /* @WebService(name="StockQuote", targetNamespace="http://www.example.org/",
4029  *             serviceName="StockQuoteService") */
4030
4031 /* @WebFunction(operationName="GetLastTradePrice",
4032  *              action="urn:GetLastTradePrice")
4033  * @WebParam(paramName="tickerSymbol", name="symbol") */
4034 float getLastTradePrice(const wchar_t *tickerSymbol);
```

4035 **D.7 JAX-WS WSDL Extensions**

4036 An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL
4037 extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.
4038 Table 3 is a list of JAX-WS WSDL extensions that MAY be interpreted, and their corresponding SCA
4039 WSDL extension. [CD0007]

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

Deleted: 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.

Deleted: The following is a list of JAX-WS WSDL extensions that MAY be recognized, and their corresponding SCA WSDL extension.

4041 Table 3: Allowed JAX-WS Extensions

Formatted: Caption

4042 **D.8 WSDL Extensions Schema**

```
4043 <?xml version="1.0" encoding="UTF-8"?>
4044 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4045         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
4046         xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
4047         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4048         elementFormDefault="qualified">
4049
4050     <element name="bindings" type="sca-c:BindingsType" />
4051     <complexType name="BindingsType">
4052         <choice minOccurs="0" maxOccurs="unbounded">
4053             <element ref="sca-c:prefix" />
4054             <element ref="sca-c:enableWrapperStyle" />
4055             <element ref="sca-c:function" />
4056             <element ref="sca-c:struct" />
4057             <element ref="sca-c:parameter" />
```

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

Deleted: Specification

Deleted: 19 March

4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086

```
</choice>
</complexType>

<element name="prefix" type="sca-c:PrefixType" />
<complexType name="PrefixType">
  <attribute name="name" type="xsd:string" use="required" />
</complexType>

<element name="function" type="sca-c:FunctionType" />
<complexType name="FunctionType">
  <attribute name="name" type="xsd:NCName" use="required" />
</complexType>

<element name="struct" type="sca-c:StructType" />
<complexType name="StructType">
  <attribute name="name" type="xsd:NCName" use="required" />
</complexType>

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

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

Deleted: Specification
Deleted: 19 March

4087

E XML Schemas

4088

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

4089

4090

4091

4092

4093

4094

4095

4096

4097

4098

4099

4100

4101

4102

4103

4104

4105

4106

4107

4108

4109

4110

4111

4112

4113

4114

4115

4116

4117

4118

4119

4120

4121

4122

4123

4124

4125

4126

4127

4128

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  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">
    <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="input" type="NCName" use="optional"/>
    <attribute name="output" type="NCName" use="optional"/>
    <anyAttribute namespace="##other" processContents="lax"/>
  </complexType>
</schema>
```

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

Formatted: Bullets and Numbering

4129

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

4130

4131

4132

4133

4134

4135

4136

4137

4138

4139

4140

4141

4142

4143

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

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

  <element name="implementation.c" type="sca:CImplementation"
    substitutionGroup="sca:implementation" />

  <complexType name="CImplementation">
    <complexContent>
      <extension base="sca:Implementation">
```

Deleted: Specification

Deleted: 19 March

```

4144     <sequence>
4145         <element name="operation" type="sca:CImplementationFunction"
4146             minOccurs="0" maxOccurs="unbounded" />
4147         <any namespace="##other" processContents="lax"
4148             minOccurs="0" maxOccurs="unbounded"/>
4149     </sequence>
4150     <attribute name="module" type="NCName" use="required"/>
4151     <attribute name="path" type="string" use="optional"/>
4152     <attribute name="library" type="boolean" use="optional"/>
4153     <attribute name="componentType" type="string" use="required"/>
4154     <attribute name="scope" type="sca:CImplementationScope"
4155         use="optional"/>
4156     <attribute name="eagerInit" type="boolean" use="optional"/>
4157     <attribute name="init" type="boolean" use="optional"/>
4158     <attribute name="destroy" type="boolean" use="optional"/>
4159     <attribute name="allowsPassByReference" type="boolean"
4160         use="optional"/>
4161     <anyAttribute namespace="##other" processContents="lax"/>
4162 </extension>
4163 </complexContent>
4164 </complexType>
4165
4166 <simpleType name="CImplementationScope">
4167     <restriction base="string">
4168         <enumeration value="stateless"/>
4169         <enumeration value="composite"/>
4170     </restriction>
4171 </simpleType>
4172
4173 <complexType name="CImplementationFunction">
4174     <attribute name="name" type="NCName" use="required"/>
4175     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4176     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4177     <attribute name="allowsPassByReference" type="boolean"
4178         use="optional"/>
4179     <attribute name="init" type="boolean" use="optional"/>
4180     <attribute name="destroy" type="boolean" use="optional"/>
4181     <anyAttribute namespace="##other" processContents="lax"/>
4182 </complexType>
4183
4184 </schema>

```

4185 E.3 sca-contribution-c-1.1.xsd

```

4186 <?xml version="1.0" encoding="UTF-8"?>
4187 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4188     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4189     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4190     elementFormDefault="qualified">
4191
4192     <include schemaLocation="sca-contributions.xsd"/>
4193
4194     <element name="export.c" type="sca:CExport"
4195         substitutionGroup="sca:Export"/>
4196
4197     <complexType name="CExport">
4198         <complexContent>
4199             <attribute name="name" type="QName" use="required"/>
4200             <attribute name="path" type="string" use="optional"/>
4201         </complexContent>
4202     </complexType>
4203
4204     <element name="import.c" type="sca:CImport"
4205         substitutionGroup="sca:Import"/>

```

Deleted: Specification

Deleted: 19 March

4206
4207
4208
4209
4210
4211
4212
4213
4214

```
<complexType name="CImport">  
  <complexContent>  
    <attribute name="name" type="QName" use="required"/>  
    <attribute name="location" type="string" use="required"/>  
  </complexContent>  
</complexType>  
</schema>
```

Deleted: Specification
Deleted: 19 March

4215

F Conformance Points

4216

This section contains a list of conformance items 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]	An SCA runtime MUST support these scopes; stateless and composite . Additional scopes MAY be provided by SCA runtimes.
[C20004]	A C implementation MUST only designate functions with no arguments and a void return type as lifecycle functions.
[C20006]	If the header file identified by the @header attribute of an <interface.c/> element contains function declarations that are not operations of the interface, then the functions that define operations of the interface MUST be identified using <function/> child elements of the <interface.c/> element.
[C20007]	If the header file identified by the @callbackHeader attribute of an <interface.c/> element contains function declarations that are not operations of the callback interface, then the functions that define operations of the callback interface MUST be identified using <callbackFunction/> child elements of the <interface.c/> element.
[C20008]	If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using message formats, then all functions of the interface (callback interface) MUST be identified using <function/> (<callbackFunction/>) child elements of the <interface.c/> element.
[C20009]	The @name attribute of a <function/> child element of a <interface.c/> MUST be unique amongst the <function/> elements of that <interface.c/>.
[C20010]	The @name attribute of a <callbackFunction/> child element of a <interface.c/> MUST be unique amongst the <callbackFunction/> elements of that <interface.c/>.
[C20011]	If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using message formats, then the struct defining the input message format MUST be identified using an @input attribute.
[C20012]	If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using message formats, then the struct defining the output message format MUST be identified using an @output attribute.
[C20013]	The @name attribute of a <function/> child element of a <implementation.c/> MUST be unique amongst the <function/> elements of that <implementation.c/>.
[C20014]	An SCA runtime MUST ensure that a stateless scoped implementation instance object is only ever dispatched on one thread at any one time. In addition, within the SCA lifecycle of an instance, an SCA runtime MUST only make a single invocation of one business function.
[C20015]	An SCA runtime MAY run multiple threads in a single composite scoped implementation instance object and it MUST NOT perform any synchronization.

Deleted: Items

Deleted: the SCA C Client and Implementation Model

Formatted Table

Formatted: Pattern: Clear (Gray-20%)

Formatted: Pattern: Clear (Yellow)

Formatted: Pattern: Clear (Light Yellow)

Formatted: Pattern: Clear (Turquoise)

Formatted: Pattern: Clear (Pale Blue)

Formatted: Pattern: Clear (Light Green)

Formatted: Pattern: Clear (Tan)

Formatted: Pattern: Clear (Yellow)

Deleted: input

Formatted: Pattern: Clear (Gray-10%)

Formatted: Pattern: Clear (Light Yellow)

Deleted: method

Deleted: Specification

Deleted: 19 March

Conformance ID	Description	Formatted Table
[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".	Formatted Table Deleted: [C20016]
[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".	Deleted: [C20017]
[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.	Deleted: [C40001] Formatted Table
[C50001]	Vendor defined reason codes SHOULD start at 101.	
[C60002]	An SCA runtime MAY additionally provide a DataObject variant of this API for handling properties with complex XML types. The type of the value parameter in this variant is DATAOBJECT.	
[C60003]	A SCA runtime MAY provide the functions <code>SCAService()</code> , <code>SCAOperation()</code> , <code>SCAMessageIn()</code> and <code>SCAMessageOut()</code> to support C implementations in programs.	Formatted Table Deleted: [C60003]
[C70001]	The <code>@name</code> attribute of a <code><export.c/></code> element MUST be unique amongst the <code><export.c/></code> elements in a domain.	Deleted: [C70001]
[C70002]	The <code>@name</code> attribute of a <code><import.c/></code> child element of a <code><contribution/></code> MUST be unique amongst the <code><import.c/></code> elements in of that contribution.	Deleted: [C70002]
[C80001]	The return type and types of the parameters of a function of a local service interface MUST be one of: <ul style="list-style-type: none"> Any fundamental or compound types as defined by C. 	Formatted: Pattern: Clear (Turquoise) Deleted: Any of the C primitive types (for example, int, short, char). In this case the data will be passed by value as is normal for C.¶ <#>Pointers to any of the C primitive types (for example, int *, short *, char *).¶ DATAOBJECT. An SDO handle
[C80002]	The return type and types of the parameters of a function of a remotable service interface MUST be one of: <ul style="list-style-type: none"> Any of the C types specified in Simple Content Binding and Complex Content Binding. These types may be passed by-value or by-pointer. Unless the function and client indicate that they allow by-reference semantics (see AllowsPassByReference), a copy will be explicitly created by the runtime for any parameters passed by-pointer. An SDO DATAOBJECT. This type may be passed by-value or by-pointer. Unless the function and client indicate that they allow by-reference semantics (see AllowsPassByReference), a deep-copy of the DATAOBJECT will be created by the runtime for any parameters passed by-value or by-pointer. When by-reference semantics are allowed, the DATAOBJECT handle will be passed. 	Formatted: Pattern: Clear (Pale Blue) Formatted: Pattern: Clear (Pale Blue) Deleted: Any of the C primitive types (for example, int, short, char). This will be copied Formatted: Pattern: Clear (Pale Blue)
[C90001]	A C header file used to define an interface MUST: <ul style="list-style-type: none"> Declare at least one function or message format struct 	Deleted: An SDO handle. The SDO will be copied and passed to the destination Formatted Table
[C90002]	A C header file used to define an interface MUST NOT use the following constructs: <ul style="list-style-type: none"> Macros 	Deleted: [C90001] Deleted: [C90002] Deleted: Specification Deleted: 19 March

Conformance ID	Description	Formatted Table
[C100001]	In the absence of customizations, an SCA implementation SHOULD map each portType to separate header file. An SCA implementation MAY use any sca-c:prefix binding declarations to control this mapping.	Deleted: [C100001]
[C100002]	For components implemented in libraries, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the function.	Deleted: [C100002]
[C100003]	In the absence of any customizations for a WSDL operation that does not meet the requirements for the wrapped style, the name of a mapped function parameter or struct member MUST be the value of the name attribute of the wsdl:part element with the first character converted to lower case.	Deleted: [C100003]
[C100004]	In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped style, the name of a mapped function parameter or struct member MUST be the value of the local name of the wrapper child with the first character converted to lower case.	Deleted: [C100004]
[C100005]	For components implemented in a program, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the request struct name. Additionally an SCA implementation MUST append "Response" to the request struct name to form the response struct name.	Deleted: [C100005]
[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.	Deleted: [C100006]
[C100007]	An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be configurable.	Deleted: [C100007]
[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.	Deleted: [C100008]
[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 front of function name before mapping it to the operation name.	Deleted: [C100009] Formatted: Pattern: Clear (Light Green) Deleted: present, Deleted: from the function 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.	Deleted: [C100010]
[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.	Deleted: [C100011]
[C100012]	In the absence of customizations, an SCA implementation MUST map the return type to a part or global element child named "return".	Deleted: [C100012] Deleted: Specification Deleted: 19 March

Conformance ID	Description
[C100013]	Program based implementation SHOULD use the Document-Literal style and encoding.
[C100014]	In the absence of customizations, an SCA implementation MUST map the struct member name to the part or global element child name.
[C100015]	An SCA implementation MUST ensure that in/out parameters have the same type in the request and response structs.
[C100016]	An SCA implementation MUST support mapping message parts or global elements with complex types and parameters , return types, and struct members with a type defined by a struct . The mapping from WSDL MAY be to DataObjects and/or structs . The mapping to and from structs MUST follow the rules defined in WSDL to C Mapping Details .
[C100017]	An SCA implementation MUST map: <ul style="list-style-type: none"> a function's return value as an out parameter. by-value and const parameters as in parameters. in the absence of customizations, pointer parameters as in/out parameters.
[C100019]	For library-based service implementations, an SCA implementation MUST map In parameters as pass by-value and In/Out and Out parameters as pass via pointers.
[C100020]	For program-based service implementations, an SCA implementation MUST map all values in the input message as pass by-value and the updated values for In/Out parameters and all Out parameters in the response message as pass by-value.
[C100021]	An SCA implementation MUST map simple types as defined in Table 1 and Table 2 by default.
[C100022]	An SCA implementation MAY map boolean to _Bool by default.
[C110001]	An SCA implementation MUST reject a composite file that does not conform to http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd or http://docs.oasis-open.org/opencsa/sca/200903/sca-implementation-c-1.1.xsd .
[C110002]	An SCA implementation MUST reject a componentType or constraining type file that does not conform to http://docs.oasis-open.org/opencsa/sca/200903/sca-interface-c-1.1.xsd .
[C110003]	An SCA implementation MUST reject a contribution file that does not conform to http://docs.oasis-open.org/opencsa/sca/200903/sca-contribution-c-1.1.xsd .
[C110004]	An SCA implementation MUST reject a WSDL file that does not conform to http://docs.oasis-open.org/opencsa/sca-c-cpp/c/200901/sca-wsdlex-c-1.1.xsd .

Formatted Table

Deleted: [C100013]

Deleted: [C100014]

Deleted: [C100015]

Deleted: [C100016]

Formatted: Pattern: Clear (Tan)

Deleted: parameters,

Deleted: ,

Deleted: WSDL to C Mapping Details

Deleted: [C100017]

Deleted: [C100019]

Formatted Table

Deleted: [C100022]

Deleted: [CA0001]

... [5]

Formatted: Bullets and Numbering

Formatted Table

Formatted: Pattern: Clear (Gray-20%)

F.1 Annotation Conformance Points

This section contains a list of conformance points related to source file annotations for this specification.

Conformance ID	Description
[CA0001]	If SCA annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to SCDL as described. The SCA runtime MUST only process the SCDL files and not the annotations.

Deleted: Specification

Deleted: 19 March

Conformance ID	Description
[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 @Interface was 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, unless the exclude value of the @WebFunction annotation is true, as if @Operation was specified with the operationName value of the @WebFunction annotation used as the name value of the @Operation annotation.
[CA0005]	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.
[CA0006]	While annotations are defined using the /* ... */ format for comments, if the // ... format is supported by a C compiler, the // ... format MAY be supported by an SCA implementation annotation processor.
[CC0001]	An SCA implementation MUST treat any instance of a @Interface annotation and without an explicit @WebService annotation as if a @WebService annotation with a name value equal to the name value of the @Interface annotation and no other parameters was specified.
[CC0002]	An SCA implementation MUST treat a function annotated with an @Operation annotation and without an explicit @WebFunction annotation as if a @WebFunction annotation with with an operationName value equal to the name value of the @Operation annotation and no other parameters was specified.
[CC0003]	An SCA implementation MUST treat an @Operation annotation without an explicit @WebOperation annotation as if a @WebOperation annotation with with an operationName value equal to the name value of the @Operation annotation, a response value equal to the output value of the @Operation annotation and no other parameters was specified is applied to the struct identified as the input value of the @Operation annotation.
[CC0004]	A C struct that is listed in a @WebThrows annotation MUST itself have a @WebFault annotation.
[CC0005]	If WSDL mapping annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to WSDL as described.
[CC0006]	The value of the type property of a @WebParam annotation MUST be either one of the simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema or, if the type of the parameter is a struct, the QName of a XSD complex type following the mapping specified in Complex Content Binding.
[CC0007]	The value of the type property of a @WebResult annotation MUST be one of the simpleTypes defined in namespace http://www.w3.org/2001/XMLSchema .
[CC0008]	If a @WebService does not have a portName element, an SCA implementation MUST use the value associated with the name element, suffixed with "Port".

Formatted Table

Formatted: Pattern: Clear (Gray-10%)

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Deleted: Specification

Deleted: 19 March

Conformance ID	Description
[CC0009]	Only named parameters MAY be referenced by a @WebParam annotation.

Formatted Table

Formatted: Bullets and Numbering

F.2 WSDL Extension Conformance Points

This section contains a list of conformance points related to WSDL extensions for this specification.

Conformance ID	Description
[CD0001]	If WSDL extensions are supported by an implementation, all the extensions defined here MUST be supported and MUST be mapped to C as described.
[CD0002]	The @type attribute of a <parameter/> element MUST be either a C type specified in Simple Content Binding or, if the message part has complex content, a struct following the mapping specified in Complex Content Binding.
[CD0003]	A <sca-c:bindings/> element MUST NOT have more than one <sca-c:prefix/> child element.
[CD0004]	A <sca-c:bindings/> element MUST NOT have more than one <sca-c:enableWrapperStyle/> child element.
[CD0005]	A <sca-c:bindings/> element MUST NOT have more than one <sca-c:function/> child element.
[CD0006]	A <sca-c:bindings/> element MUST NOT have more than one <sca-c:struct/> child element.
[CD0007]	An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present. Table 3 is a list of JAX-WS WSDL extensions that MAY be interpreted, and their corresponding SCA WSDL extension.

Formatted Table

Formatted: Bullets and Numbering

F.3 JAX-WS Conformance Points

The JAX-WS 2.1 specification [JAXWS21] defines conformance points for various requirements defined by that specification. The following table outlines those conformance points, which apply to the WSDL mapping described in this specification.

Section	Conformance Point	Notes	Conformance ID
2	WSDL 1.1 support	[A]	[CF0001]
2	Customization required	[CD0001] The reference to the JAX-WS binding language <u>is</u> treated as a reference to the C WSDL extensions defined in <u>C WSDL Mapping Extensions</u> .	
2	Annotations on generated classes		[CF0002]
2.1	WSDL and XML Schema import directives		[CF0003]
2.1.1	Optional WSDL extensions		[CF0004]
2.2	SEI naming	[C100001]	

Deleted: statements

Deleted: statements

Deleted: and describes whether the conformance statement

Deleted: ies

Deleted: binding

Formatted Table

Deleted: Statement

Deleted: are

Deleted: C WSDL Mapping Extensions

Formatted Table

Deleted: [C100001]

Deleted: Specification

Deleted: 19 March

Section	Conformance Point	Notes	Conformance ID
2.2	javax.jws.WebService required	[B] References to javax.jws.WebService in the conformance statement are treated as the C annotation @WebService.	[CF0005]
2.3	Method naming	[C100002] and [C100005]	
2.3	javax.jws.WebMethod required	[A], [B] References to javax.jws.WebMethod in the conformance statement are treated as the C annotation @WebFunction or @WebOperation.	[CF0006]
2.3	Transmission primitive support		[CF0007]
2.3	Using javax.jws.OneWay	[A], [B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0008]
2.3.1	Using javax.jws.SOAPBinding	[A], [B] References to javax.jws.SOAPBinding in the conformance statement are treated as the C annotation @SOAPBinding.	[CF0009]
2.3.1	Using javax.jws.WebParam	[A], [B] References to javax.jws.WebParam in the conformance statement are treated as the C annotation @WebParam.	[CF0010]
2.3.1	Using javax.jws.WebResult	[A], [B] References to javax.jws.WebResult in the conformance statement are treated as the C annotation @WebResult.	[CF0011]
2.3.1.1	Non-wrapped parameter naming	[C100003]	
2.3.1.2	Default mapping mode		[CF0012]
2.3.1.2	Disabling wrapper style	[B] References to jaxws:enableWrapperStyle in the conformance statement are treated as the C annotation sca-c:enableWrapperStyle.	[CF0013]
2.3.1.2	Wrapped parameter naming	[C100004]	
2.3.1.2	Parameter name clash	[A]	[CF0014]

Formatted Table

Deleted: Statement

Deleted: [C100002]

Deleted: [C100005]

Deleted: [C100003]

Deleted: [C100004]

Deleted: Specification

Deleted: 19 March

Section	Conformance Point	Notes	Conformance ID
2.5	javax.xml.ws.WebFault required	[B] References to javax.jws.WebFault in the conformance statement are treated as the C annotation @WebFault.	[CF0015]
2.5	Exception naming	[C100006]	
2.5	Fault equivalence	[A] References to fault exception classes are treated as references to fault message structs.	[CF0016]
2.6	Required WSDL extensions	MIME Binding not necessary	[CF0018]
2.6.1	Unbound message parts	[A]	[CF0019]
2.6.2.1	Duplicate headers in binding		[CF0020]
2.6.2.1	Duplicate headers in message		[CF0021]
3	WSDL 1.1 support	[A]	[CF0022]
3	Standard annotations	[A] [CC0005]	
3.1	Java identifier mapping	[A]	[CF0023]
3.2	WSDL and XML Schema import directives		[CF0024]
3.4	portType naming	[C100008]	
3.5	Operation naming	[C100009] and [C100010]	
3.5.1	One-way mapping	[B] References to javax.jws.OneWay in the conformance statement are treated as the C annotation @OneWay.	[CF0025]
3.5.1	One-way mapping errors		[CF0026]
3.6.1	Parameter classification	[C100017]	
3.6.1	Parameter naming	[C100011] and [C100014]	
3.6.1	Result naming	[C100012]	
3.6.1	Header mapping of parameters and results	References to javax.jws.WebParam in the conformance statement are treated as the C annotation @WebParam. References to javax.jws.WebResult in the conformance statement are treated as the C annotation @WebResult.	[CF0027]
3.7	Exception naming	[CC0004]	

Formatted Table

Deleted: Statement

Deleted: [C100006]

Formatted Table

Deleted: [C100008]

Deleted: [C100009]

Deleted: [C100010]

Deleted: [C100017]

Deleted: [C100011]

Deleted: [C100014]

Deleted: [C100012]

Deleted: Specification

Deleted: 19 March

Section	Conformance <u>Point</u>	Notes	Conformance ID
3.8	Binding selection	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY] .	[CF0029]
3.10	SOAP binding support	[A]	[CF0030]
3.10.1	SOAP binding style required		[CF0031]
3.11	Port selection		[CF0032]
3.11	Port binding	References to the BindingType annotation are treated as references to SOAP related intents defined by [POLICY] .	[CF0033]

Formatted Table

Deleted: Statement

Formatted Table

- 4225 [A] All references to Java in the conformance point are treated as references to C.
- 4226 [B] Annotation generation is only necessary if annotations are supported by an SCA implementation.

Deleted: statement

4227 **F.3.1 Ignored Conformance Points**

Deleted: Statements

Formatted: Bullets and Numbering

Formatted Table

Deleted: Statement

Section	Conformance <u>Point</u>
2.1	Definitions mapping
2.2	javax.xml.bind.XmlSeeAlso required
2.3.1	use of JAXB annotations
2.3.1.2	Using javax.xml.ws.RequestWrapper
2.3.1.2	Using javax.xml.ws.ResponseWrapper
2.3.3	Use of Holder
2.3.4	Asynchronous mapping required
2.3.4	Asynchronous mapping option
2.3.4.2	Asynchronous method naming
2.3.4.2	Asynchronous parameter naming
2.3.4.2	Failed method invocation
2.3.4.4	Response bean naming
2.3.4.5	Asynchronous fault reporting
2.3.4.5	Asynchronous fault cause
2.4	JAXB class mapping
2.4	JAXB customization use
2.4	JAXB customization clash
2.4.1	javax.xml.ws.wsaddressing.W3CEndpointReference
2.5	Fault Equivalence

Deleted: Specification

Deleted: 19 March

Section	Conformance Point
2.6.3.1	Use of MIME type information
2.6.3.1	MIME type mismatch
2.6.3.1	MIME part identification
2.7	Service superclass required
2.7	Service class naming
2.7	javax.xml.ws.WebServiceClient required
2.7	Default constructor required
2.7	2 argument constructor required
2.7	Failed getPort Method
2.7	javax.xml.ws.WebEndpoint required
3.1.1	Method name disambiguation
3.2	Package name mapping
3.3	Class mapping
3.4.1	Inheritance flattening
3.4.1	Inherited interface mapping
3.6	use of JAXB annotations
3.6.2.1	Default wrapper bean names
3.6.2.1	Default wrapper bean package
3.6.2.3	Null Values in rpc/literal
3.7	java.lang.RuntimeExceptions and java.rmi.RemoteExceptions
3.7	Fault bean name clash
3.11	Service creation

Formatted Table

Deleted: Statement

Deleted: Specification

Deleted: 19 March

4228

G Migration

4229
4230
4231

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.

4232

G.1 Implementation.c attributes

4233

@location has been replaced with *@path*.

4234

G.2 SCALocate and SCALocateMultiple

4235
4236

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

Deleted: Specification
Deleted: 19 March

4237

H Acknowledgements

4238

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

4239

4240

Participants:

4241

Participant Name	Affiliation
Bryan Aupperle	IBM
Andrew Borley	IBM
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
David Haney	Individual
Mark Little	Red Hat
Jeff Mischkinsky	Oracle Corporation
Peter Robbins	IBM

Deleted: Specification

Deleted: 19 March

4242
4243
4244

4245

I Revision History

[optional; should not be included in OASIS Standards]

Revision	Date	Editor	Changes Made
<u>2</u>		<u>Bryan Aupperle</u>	<ul style="list-style-type: none"><u>Apply Changes for CCPP-75 and CCPP-76</u>
<u>1</u>	<u>30 April 2009</u>	<u>Bryan Aupperle</u>	<ul style="list-style-type: none"><u>Apply Changes for CCPP-62, CCPP-64, CCPP-66, CCPP-68 and CCPP-71</u>

Deleted: Specification
Deleted: 19 March

Page 16: [1] Deleted	Bryan Aupperle	6/24/2009 2:48 PM
----------------------	----------------	-------------------

The @name attribute of a <callbackFunction/> child element of a <interface.c/> MUST be unique amongst the <callbackFunction/> elements of that <interface.c/>.

Page 16: [2] Deleted	Bryan Aupperle	6/24/2009 2:47 PM
----------------------	----------------	-------------------

If the header file identified by the @header or @callbackHeader attribute of an <interface.c/> element defines the operations of the interface (callback interface) using message formats, then the struct defining the input message format MUST be identified using an @input attribute.

Page 78: [3] Deleted	Bryan Aupperle	3/27/2009 2:44 PM
----------------------	----------------	-------------------

Allow	@Allow(roles="<comma separated list of roles>")
permitAll	@PermitAll
denyAll	@DenyAll

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

Page 79: [4] Deleted	Bryan Aupperle	3/27/2009 2:42 PM
----------------------	----------------	-------------------

JMS	@JMS
Application of Intent Annotations	
Policy Annotation Scope	
Relationship of Declarative And Annotated Intents	

Page 112: [5] Deleted	Bryan Aupperle	4/30/2009 2:20 PM
-----------------------	----------------	-------------------

[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 @Interface was 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, unless the exclude value of the @WebFunction annotation is true, as if @Operation was specified with the operationName value of the @WebFunction annotation used as the name value of the @Operation annotation.
[CA0005]	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.

- [CC0001] An SCA implementation MUST treat any instance of a `@Interface` annotation and without an explicit `@WebService` annotation as if a `@WebService` annotation with a name value equal to the name value of the `@Interface` annotation and no other parameters was specified.
- [CC0002] An SCA implementation MUST treat a function annotated with an `@Operation` annotation and without an explicit `@WebFunction` annotation as if a `@WebFunction` annotation with an operationName value equal to the name value of the `@Operation` annotation and no other parameters was specified.
- [CC0003] An SCA implementation MUST treat an `@Operation` annotation without an explicit `@WebOperation` annotation as if a `@WebOperation` annotation with an operationName value equal to the name value of the `@Operation` annotation, a response value equal to the output value of the `@Operation` annotation and no other parameters was specified is applied to the struct identified as the input value of the `@Operation` annotation.
- [CC0004] A C struct that is listed in a `@WebThrows` annotation MUST itself have a `@WebFault` annotation.
- [CC0005] If WSDL mapping annotations are supported by an implementation, the annotations defined here MUST be supported and MUST be mapped to WSDL as described.
- [CC0006] The value of the type property of a `@WebParam` annotation MUST be one of the simpleTypes defined in namespace <http://www.w3.org/2001/XMLSchema>.
- [CC0007] The value of the type property of a `@WebResult` annotation MUST be one of the simpleTypes defined in namespace <http://www.w3.org/2001/XMLSchema>.
- [CC0008] If a `@WebService` does not have a **portName** element, an SCA implementation MUST use the value associated with the **name** element, suffixed with “Port”.
- [CC0009] Only named parameters MAY be referenced by a `@WebParam` annotation.
- [CD0001] If WSDL extensions are supported by an implementation, all the extensions defined here MUST be supported and MUST be mapped to C as described.
- [CD0002] The `@type` attribute of a `<parameter/>` element MUST be a valid C type.
- [CD0003] A `<sca-c:bindings/>` element MUST NOT have more than one `<sca-c:prefix/>` child element.
- [CD0004] A `<sca-c:bindings/>` element MUST NOT have more than one `<sca-c:enableWrapperStyle/>` child element.
- [CD0005] A `<sca-c:bindings/>` element MUST NOT have more than one `<sca-c:function/>` child element.
- [CD0006] A `<sca-c:bindings/>` element MUST NOT have more than one `<sca-c:struct/>` child element.
- [CD0007] An SCA implementation MAY support the reading and interpretation of JAX-WS defined WSDL extensions; however it MUST give precedence to the corresponding SCA WSDL extension if present.