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

Committee Draft 03 Revision 2

30 April 2009

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 (Authoratative)

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 (Authoratative)

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 (Authoratative)

Technical Committee:

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

Chair:

Bryan Aupperle, IBM

Editors:

Bryan Aupperle, IBM
David Haney
Pete Robbins, IBM

Related work:

This specification replaces or supercedes:

- OSOA SCA C Client and Implementation V1.00

This specification is related to:

- OASIS Service Component Architecture Assembly Model Version 1.1
- OASIS SCA Policy Framework Version 1.1
- OASIS Service Component Architecture Web Service Binding Specification Version 1.1

Declared XML Namespace(s):

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

Abstract:

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

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

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

Status:

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

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

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

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

Notices

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

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

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

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

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

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

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

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

Table of Contents

1 Introduction

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/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 Commmittee 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/

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

1.3 Conventions

1.3.1 Naming Conventions

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

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

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

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

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

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

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

1.3.2 Typographic Conventions

This specification follows some typographic conventions for some specific constructs

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

2 Basic Component Implementation Model

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

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

2.1 Implementing a Service

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

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

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

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

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

Service interface.

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

Implementation.

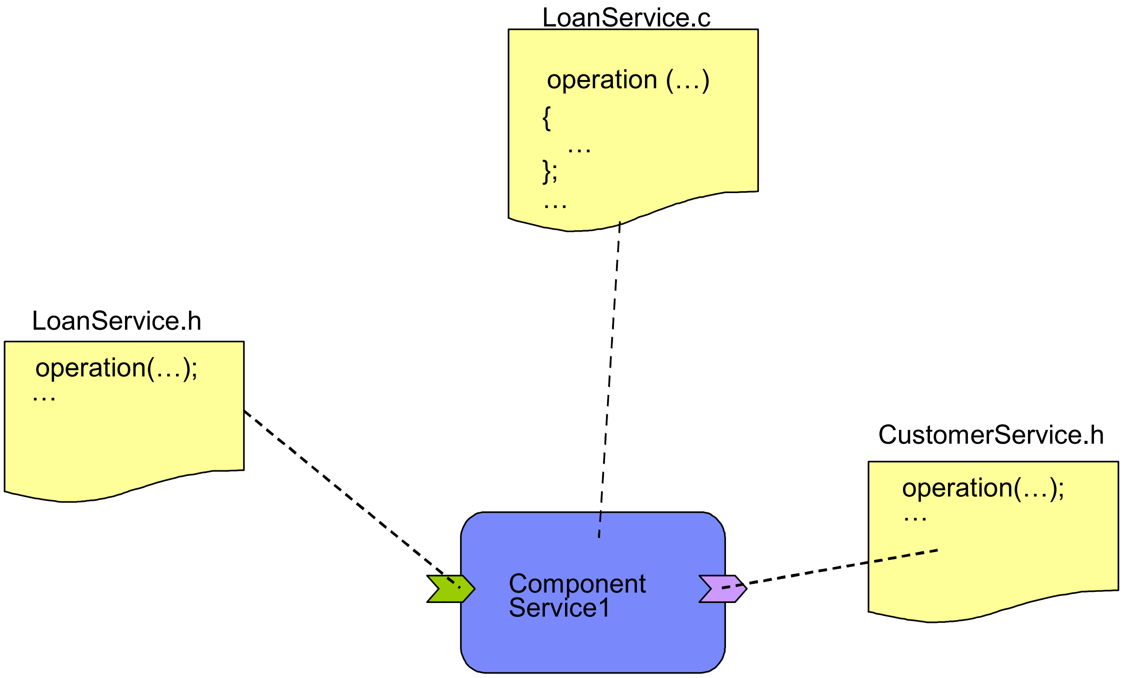
```
#include "LoanService.h"

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.



2.1.1 Implementing a Remotable Service

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

```
<?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" remotable="true"/>
   </service>
</componentType>
```

2.1.2 AllowsPassByReference

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

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

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

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

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

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

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

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

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”. [C20016]

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”. [C20017]

2.1.3 Implementing a Local Service

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

2.2 Component and Implementation Scopes

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

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

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

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

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]

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]

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..

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

```
#include "SCA.h"

void clientFunction()
{

   …

   int32_t loanRating;
   int compCode, reason;

…

   SCAPropertyInt(L"maxLoanValue", &loanRating, &compCode, &reason);

   …

}
```

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

2.4 Component Type and Component

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

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

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

```
/* 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 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.

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

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

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

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"* extention 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 sematics 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 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.

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

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

2.4.4 Implementation Function

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

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

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

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

</implementation.c>
```

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

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

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

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

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

```
#include "SCA.h"
#include "myInterface.h"
main () {
    wchar_t myService [255];
    wchar_t myOperation [255];
    int compCode, reason;
    struct FirstInputMsg myFirstIn;
    struct FirstOutputMsg myFirstOut;


    SCAService(myService, &compCode, &reason);

    SCAOperation(myOperation, &compCode, &reason);

    if (wcscmp(myOperation,L"myFirstOperation")==0){
        SCAMessageIn(myService, myOperation,
                     sizeof(struct FirstInputMsg), (void *)&myFirstIn,
                     &compCode, &reason);
        …
        SCAMessageOut(myService, myOperation,
                      sizeof(struct FirstOutputMsg),(void *)&myFirstOut,
                      &compCode, &reason);
    }
    else
       {
           …
       }
}
```

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);
   };
};
```

3.2 Accessing Services from non-SCA component implementations

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

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

```
#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);
}
```

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

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

- componentName/serviceName/bindingName

3.3 Calling Service Operations

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

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

3.3.1 Proxy Functions

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

For SCA, proxy functions have the form:

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

where:

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

- **<functionParameters>** are the parameters of the interface function
- **<functionReturn>** is the return type of the interface function

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

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

The following shows a sample of using a proxy function.

```
#include "SCA.h"

void clientFunction()
{

   SCAREF serviceToken;
   int compCode, reason;
   long custNum = 1234;
   short rating;

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

}
```

An SCA implementation MAY support proxy functions. [C30001]

3.4 Long Running Request-Response Operations

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

3.4.1 Asynchronous Invocation

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

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

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

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

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

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

the asynchronous invocation style includes a proxy function

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

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

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

```
#include "SCA.h"
#include "TravelService.h"

SCAREF serviceToken;
int compCode, reason;

void makeReservationsHandler(short rspType)
{
   struct confirmationData cd;
   wchar_t *fault, *faultDetails;

   if (rspType == SCA_REPLY_MESSAGE {
      SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
      …
   }
   else {
      SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
                         &faultDetails, &compCode, &reason);
      if (wcscmp(*fault, L”noFlight”) {
         …
      }
      else {
         …
      }
   }

   return;
}

void clientFunction()
{

   struct itineraryData id;

   …

   void (*ah)(short) = &makeReservationsHandler;

   SCAGetReference(L”customerService”, &serviceToken, &compCode, &reason);

   SCAInvokeAsync(serviceToken, L“makeReservations”, sizeof(itineraryData),
                 ah, &compCode, &reason);

   return;
}
```

3.4.2 Polling Invocation

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()`.

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

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

the polling invocation style includes a proxy function

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

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

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

```
#include "SCA.h"
#include "TravelService.h"

void pollingClientFunction()
{
   SCAREF serviceToken;
   int compCode, reason;
   short rspType;

   struct itineraryData id;
   struct confirmationData cd;
   wchar_t *fault, *faultDetails;

   …

   SCAGetReference(L”customerService”, &serviceToken, &compCode, &reason);

   SCAInvokePoll(serviceToken, L“makeReservations”, sizeof(itineraryData),
                 &compCode, &reason);

   SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
   while (!rspType) {
      // do something, then wait for some time…
      SCACheckResponse(serviceToken, &rspType, &compCode, &reason);
   }
   if (rspType == SCA_REPLY_MESSAGE {
      SCAGetReplyMessage(serviceToken, sizeof(cd), &cd, &compCode, &reason);
      …
   }
   else {
      SCAGetFaultMessage(serviceToken, sizeof(faultDetails), &fault,
                         &faultDetails, &compCode, &reason);
      if (wcscmp(*fault, L”noFlight”) {
         …
      }
      else {
         …
      }
   }

   return;
}
```

3.4.3 Synchronous Invocation

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

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

```
#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 (wcscmp(*fault, L”noFlight”) {
         …
      }
      else {
         …
      }
   }

   return;
}
```

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]

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>
```

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.

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

char state[3] = “TX”, zipCode[6] = “78746”, creditRating[3] = “AA”;

ClientFunction() {
   SCAREF serviceToken;
   int compCode, reason;

   SCAGetReference(L”quotationService”, &serviceToken, &compCode, &reason);

   SCA_requestQuotation(serviceToken, “AB123”, 2000);
}

char *getState() {
   return state;
}
char *getZipCode() {
   return zipCode;
}
char *getCreditRating() {
   return creditRating;
}
```

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

4.2.2 Callback Instance Management

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

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

4.2.3 Implementing Multiple Bidirectional Interfaces

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

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
```

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

6 C API

6.1 SCA Programming Interface

The following shows the C interface declarations for synchronous programming.

```
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);
```

```

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

void SCAPropertyShort(wchar_t *propertyName,
                      int16_t *value,
                      int *compCode,
                      int *reason);

void SCAPropertyInt(wchar_t *propertyName,
                    int32_t *value,
                    int *compCode,
                    int *reason);

void SCAPropertyLong(wchar_t *propertyName,
                     int64_t *value,
                     int *compCode,
                     int *reason);

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

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

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

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

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

void SCAGetReplyMessage(SCAREF token,
                        int *bufferLen,
                        char *buffer,
                        int *compCode,
                        int *reason);

void SCAGetFaultMessage(SCAREF token,
                        int *bufferLen,
                        wchar_t **faultName,
                        char *buffer,
                        int *compCode,
                        int *reason);

void SCASetFaultMessage(wchar_t *serviceName,
                        wchar_t *operationName,
                        wchar_t *faultName,
                        int bufferLen,
                        char *buffer,
                        int *compCode,
                        int *reason);

void SCASelf(wchar_t *serviceName,
             SCAREF *serviceToken,
             int *compCode,
             int *reason);

void SCAGetCallback(wchar_t *serviceName,
                    SCAREF *serviceToken,
                    int *compCode,
                    int *reason);

void SCAReleaseCallback(SCAREF serviceToken,
                    int *compCode,
                    int *reason);

void SCAInvokeAsync(SCAREF token,
                    wchar_t *operationName,
                    int inputMsgLen,
                    void *inputMsg,
                    void (*handler)(short),
                    int *compCode,
                    int *reason);

void SCAInvokePoll(SCAREF token,
                   wchar_t *operationName,
                   int inputMsgLen,
                   void *inputMsg,
                   int *compCode,
                   int *reason);

void SCACheckResponse(SCAREF token,
                      short *responseType,
                      int *compCode,
                      int *reason);

void SCACancelInvoke(SCAREF token,
                     int *compCode,
                     int *reason);

void SCAEntryPoint(wchar_t *serviceURI,
                   wchar_t *domainURI,
                   SCAREF *serviceToken,
                   int *compCode,
                   int *reason);
```

The C synchronous programming interface has the following functions:

6.1.1 SCAGetReference

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

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

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

6.1.2 SCAGetReferences

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

Precondition	C component instance is running	
Input Parameter	referenceName	Name of the Reference to initialize
Output Parameters	referenceTokens	Array of tokens to be used in subsequent SCAInvoke() calls. These will all be NULL if referenceName is not defined for the component. Operations need to be invoked on each token in the array.
	num_targets	Number of tokens returned in the array.
	compCode	SCACC_OK, if the call is successful SCACC_ERROR, otherwise – see reason for details
	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.	

6.1.3 SCAInvoke

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

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

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

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

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.	

6.1.5 SCAGetReplyMessage

A C component implementation uses `SCAGetReplyMessage()` to retrieve the reply message of an 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.	

6.1.6 SCAGetFaultMessage

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

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

6.1.7 SCASetFaultMessage

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

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

6.1.8 SCASelf

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.	

6.1.9 SCAGetCallback

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

Precondition	C component instance is running	
Input Parameter	`serviceName`	Name of the Service to initialize. If a component only provides one service, this string can be empty.
Output Parameters	`serviceToken`	Token to be used in subsequent `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.	

6.1.10 SCAReleaseCallback

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

Precondition	C component instance is running and has a valid serviceToken	
Input Parameter	`serviceToken`	Token returned by prior `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

Post Condition	The token becomes invalid for subsequent calls.

6.1.11 SCAInvokeAsync

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

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

6.1.12 SCAInvokePoll

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

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

		SCA_PARAMETER_ERROR if the operationName is not defined for the interface SCA_SERVICE_UNAVAILABLE if provider of the interface is no longer operational
Post Condition	Unless a SCA_SERVICE_UNAVAILABLE reason is returned, the token remains valid for subsequent calls.	

6.1.13 SCACheckResponse

A C component implementation uses SCACheckResponse() to determine if a response to a long 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	

6.1.14 SCACancelInvoke

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.	

6.1.15 SCAEntryPoint

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

Precondition	None	
Input Parameter	serviceURI	URI of the Service to access

	domainURI	URI of the SCA domain
Output Parameters	serviceToken	Token to be used in subsequent SCAInvoke() 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 Implemenation Support

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

```
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.

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.	

6.2.3 SCAMessageIn

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 `SCAService()`.
	`operationName`	Name returned by `SCAOperation()`.
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.	

6.2.4 SCAMessageOut

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

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

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"
   <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>
```

7.2 componentType files

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

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

The SCDL for the AutoInsuranceService component is:

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

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

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

7.3 C Contribution Extensions

7.3.1 Export.c

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

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

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

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

7.3.2 Import.c

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

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

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

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

8 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]

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]

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

- Macros [C90002]

10 WSDL to C and C to WSDL Mapping

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

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

The following general 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 annorations 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.

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

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

Header file naming is implementation dependent.

10.1.3 Operations

Asynchronous mapping is not supported.

10.1.3.1 Operation Names

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

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. [C100002]

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

For program-based service implementations:

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

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 stuct name. Additionally an SCA implementation MUST append *"Response"* to the request struct name to form the response struct name. [C100005]

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

10.1.3.2 Message and Part

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. [C100003]

In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped sytle, 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. [C100004]

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

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. [C100019]

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. [C100020]

10.1.4 Types

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

MTOM/XOP content processing is left to the application.

10.1.5 Fault

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

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. [C100006]

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

10.1.6 Service and Port

This mapping does not define generation of client side code.

10.1.7 XML Names

See comments in Operations

10.2 Interpretations for C to WSDL Mapping

10.2.1 Package

Not relevant.

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

10.2.2 Class

Not relevant since mapping is only based on declarations.

10.2.3 Interface

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

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

or (for components implemented in programs):

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

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. [C100008]

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

10.2.4 Method

For components implemented in libraries, functions map to operations.

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. [C100009]

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

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

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

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

10.2.5 Method Parameters and Return Type

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

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. [C100011]

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

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

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

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. [C100017]

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

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

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

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

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

10.2.6 Service Specific Exception

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

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

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

10.2.7 Generics

Not relevant.

10.2.8 Service and Ports

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

10.3 Data Binding

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

10.3.1 Simple Content Binding

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

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

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

Table 1: XSD simple type to C type mapping

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

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

Table 2: C type to XSD type mapping

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

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

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

10.3.1.1 WSDL to C Mapping Details

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

Example:

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

maps to:

```
wchar_t *myString;
/* this points to a dymically allocated buffer with the data */

<xsd:simpleType name=”boundedString25”>
  <xsd:restriction base="xsd:string">
    <xsd:length value="25"/>
  </xsd:restriction>
</xsd:simpletype>
…
<xsd:element name=”myString” type=”boundedString25”/>
```

maps to:

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

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

Examples:

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

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;
```

```
/* this points to a dynamically allocated array of struct tm’s */
```

```
<xsd:simpleType name=”lineNumList”>
  <xsd:list itemType=”xsd:int”/>
</xsd:simpleType>
…
<xsd:element name=”lineNums” type=”lineNumList”/>
```

maps to:

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

- Union Types are not supported.

10.3.1.2 C to WSDL Mapping Details

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

Example:

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

maps to:

```
<xsd:element name="myFunction">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="idList" type="xsd:short"
                   minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="value" type="xsd:double"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- Multi-dimensional arrays map into nested elements.

Example:

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

maps to:

```
<xsd:element name="myFunction">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="multiIdArray"
                   minOccurs="0" maxOccurs="unbounded"/>
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="multiIdArray"
                         minOccurs="4" maxOccurs="4"/>
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="multiIdArray" type="xsd:short"
```

```
                               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 `struct`s. The mapping to and from `struct`s MUST follow the rules defined in WSDL to C Mapping Details. [C100016]

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;
  /* this points to a dynamically allocated array of longs */
  double value;
};
```

- 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:

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

  maps to:

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

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

  Example:

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

  maps to:

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

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

10.3.2.2 C to WSDL Mapping Details

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

  Example:

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

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

```
struct DataStruct {
  char *name;
  double value;
};
```

maps to:

```
<xsd:element name="myFunction">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="data" type="DataStruct" />
      <xsd:element name="id" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="DataStruct">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="value" type="xsd:double"/>
  </xsd:sequence>
</xsd:complexType>
```

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

Example:

```
struct DataStruct {
  char name[256];
  double value;
};
```

maps to:

```
<xsd:element name="myFunction">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="data" type="DataStruct" />
      <xsd:element name="id" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="DataStruct">
  <xsd:sequence>
    <xsd:element name="name">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:maxLength value="255"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="value" type="xsd:double"/>
  </xsd:sequence>
</xsd:complexType>
```

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

Example:

```
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 `struct`s or `enum`s, the mapping rules are applied recursively.

Example:

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

with types defined as follows:

```
struct DataStruct {
  char name[30];
  double values[20];
  ParameterType type;
};

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

maps to:

```
<xsd:element name="myFunction">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="data" type="DataStruct"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="DataStruct">
  <xsd:sequence>
    <xsd:element name="name">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:maxLength value="29"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="values" type="xsd:double" minOccurs=20
maxOccurs=20/>
    <xsd:element name="type" type=" ParameterType"/>
  </xsd:sequence>
</xsd:complexType>

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

- Mapping of C `union`s is not supported by this specification.
- `Typedef`s are resolved when evaluating parameter and return types. `Typedef`s are resolved before the mapping to Schema is done.

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-wsdlext-c-1.1.xsd. [C110004]

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.

6. It MUST support <export.c/> and <import.c/> elements as defined in C Contributions in contribution documents.
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.
8. It MAY support WSDL extentsions as defined in C WSDL Mapping Extensions. If WSDL extentsionsare 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.

11.3 SCA Documents

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

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

   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.

   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.

11.4 C Files

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

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.

11.5 WSDL Files

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

1. It is a valid WSDL 1.1 **[WSDL11]** document.
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.

A C SCA Annotations

To allow developers to define SCA related information directly in source files, without having to separately author SCDL files, a set of annotations is defined. 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. [CA0001]

A.1 Application of Annotations to C Program Elements

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

- Function or Function Prototype

  The annotation immediately precedes the function definition or declaration.

  Example:

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

- Variable

  The annotation immediately precedes the variable definition.

  Example:

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

- Set of Functions Implementing a Service

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

  Example:

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

- Set of Function Prototypes Defining an Interface

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

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

A.2 Interface Header Annotations

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

A.2.1 @Interface

Annotation that indicates the start of a new interface definition. 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. [CA0003]

Corresponds to: *interface.c* element

Format:

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

where

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

Applies to: Set of functions defining an interface.

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

Example:

Interface header:

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

Service definition:

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

A.2.2 @Operation

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

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

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

Format:

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

where

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

Applies to (library based implementations): Function declaration

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

Example:

Interface header (loans.h):

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

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

Interface definition:

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

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

Format:

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

where

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

Applies to (program based implementations): stuct declarations

Example:

Interface header (loans.h):

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

Interface definition:

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

A.2.3 @Remotable

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

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

Format:

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

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

Applies to: Interface

Example:

Interface header (LoanService.h):

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

Service definition:

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

A.2.4 @Callback

Annotation on a service interface to specify the callback interface.

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

Format:

```
/* @Callback(header=”headerName”) */
```

where

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

Applies to: Interface

Example:

Interface header (MyService.h):

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

Service definition:

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

A.2.5 @OneWay

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

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

Format:

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

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

Applies to: Function Prototype

Example:

Interface header:

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

Service definition:

```
<service name="LoanService">
   <interface.c header="LoanService.h">
      <function name="reportEvent" oneWay="true" />
   </interface.c>
</service>
```

A.3 Implementation Annotations

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

A.3.1 @ComponentType

Annotation used to indicate the start of a new componentType.

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

Format:

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

Applies to: Set of services, references and properties

Example:

Implementation:

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

Component definition:

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

A.3.2 @Service

Annotation that indicates the start of a new service implementation.

Corresponds to: *implementation.*c element

Format:

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

where

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

Applies to: Set of functions implementing a service

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

Example:

Implementation:

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

ComponentType definition:

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

A.3.3 @Reference

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

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

Format:

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

where

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

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

Applies to: Service

Example:

Implementation:

```
/* @Reference(name="getRate", interfaceHeader="rates.h") */

/* @Reference(name="publishRate", interfaceHeader="myRates.h",
              required="false", multiple="yes")
 */
```

ComponentType definition:

```
<componentType name="LoanService">
   <reference name="getRate">
      <interface.c header="rates.h">
   </reference>
   <reference name="publishRate" multiplicity="0..n">
      <interface.c header="myRates.h">
   </reference>
</componentType>
```

A.3.4 @Property

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

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

Format:

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

where

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

Applies to: Variable

Example:

Implementation:

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

ComponentType definition:

```
<componentType name="LoanService">
   <property name=”loanType” type=”xsd:int” />
</componentType>
```

A.3.5 @Scope

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

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

Format:

```
/* @Scope(“value”) */
```

where

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

Applies to: Service

Example:

Implementation:

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

Component definition:

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

A.3.6 @Init

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

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

Format:

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

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

Applies to: Function or Service

Example:

Implementation:

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

Component definition:

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

A.3.7 @Destroy

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.

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

Format:

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

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

Applies to: Function or Service

Example:

Implementation:

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

Component definition:

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

A.3.8 @EagerInit

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

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

Format:

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

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

Applies to: Service

Example:

Implementation:

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

Component definition:

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

A.3.9 @AllowsPassByReference

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

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

Format:

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

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

Applies to: Service or Function

Example:

Implementation:

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

Component definition:

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

A.4 Base Annotation Grammar

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

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

<baseAnnotation> ::= <name> [(<params>)]

<params> ::= <paramNameValue>[, <paramNameValue>]* |
             <paramValue>[, <paramValue>]*

<paramNameValue> ::= <name>=”<value>”

<paramValue> ::= “<value>”

<name> ::= NCName

<value> ::= string
```

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

- 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.

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.

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:

```
"{" + 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" */
```

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

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

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

Format:

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

where

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

Applies to: Interface, Service, Function, Function Prototype

Examples:

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

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

A reference requiring support for confidentiality:

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

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

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

B.2 Specific Intent Annotations

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

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

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

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

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

Format:

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

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

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

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

Example:

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

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

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

B.2.1 Security Interaction

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

These three intents can be qualified with

- transport
- message

B.2.2 Security Implementation

Intent	**Annotation**	**Qualifiers**
authorization	@Authorization	fine_grain

B.2.3 Reliable Messaging

Intent	**Annotation**
atLeastOnce	@AtLeastOnce

atMostOnce	@AtMostOnce
ordered	@Ordered
exactlyOnce	@ExactlyOnce

B.2.4 Transactions

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

B.2.5 Miscellaneous

Intent	Annotation	Qualifiers
SOAP	@SOAP	1_1 1_2

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:

```
/* @Reference(name="helloService", interfaceHeader="helloService.h",
 *             required=true)
 * @PolicySets({ MY_NS "WS_Encryption_Policy",
 *                MY_NS "WS_Authentication_Policy" }) */
   HelloService* helloService;
  …
}
```

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

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

B.4 Policy Annotation Grammar Additions

```
<annotation> ::= /* @<baseAnnotation> | @<requiresAnnotation> |
                     @<intentAnnotation> | @<policySetAnnotation> */

<requiresAnnotation> ::= Requires(<intents>)

<intents> ::= “<qualifiedIntent>” |
              {“<qualifiedIntent>”[, “<qualifiedIntent>”]*})

<qualifiedIntent> ::= <intentName> | <intentName>.<qualifier> |
                      <intentName>.<qualifier>.qualifier>

<intentName> ::= {anyURI}NCName

<intentAnnotation> ::= <intent>[(<qualifiers>)]

<intent> ::= NCName[(param)]

<qualifiers> ::= “<qualifier>” | {“<qualifier>”[, “<qualifier>”]*}

<qualifier> ::= NCName | NCName/<qualifier>

<policySetAnnotation> ::= policySets(<policysets>)

<policySets> ::= “<policySetName>” | {“<policySetName>”[, “<policySetName>”]*}

<policySetName> ::= {anyURI}NCName
```

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

B.5 Annotation Constants

```
<annotationConstant> ::= /* @Define <identifier> <token string> */

<identifier> ::= token

<token string> ::= “string” | “string”[ ## <token string>]
```

- Constants are immediately expanded

C C WSDL Annotations

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

C.1 Interface Header Annotations

C.1.1 @WebService

Annotation on a C header file indicating that it represents a web service. A second or subsequent instance of this annotation in a file, or a first instance after any function declarations indicates the start of a new service and has to contain a name value. 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. [CC0001]

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

Format:

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

where

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

Applies to: Header file

Example:

Input C header file (stockQuote.h):

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

…
```

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/">

   <portType name="StockQuote">
      <sca-c:bindings>
         <sca-c:prefix name="stockQuote"/>
      </sca-c:bindings>
   </portType>

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

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

C.1.2 @WebFunction

Annotation on a C function indicating that it represents a web service operation. 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. [CC0002]

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

Format:

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

where:

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

Applies to: Function.

Example:

Input C header file:

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

/* @WebFunction(operationName="GetLastTradePrice",
 *              action="urn:GetLastTradePrice") */
float getLastTradePrice(const char *tickerSymbol);

/* @WebFunction(exclude="true") */
void setLastTradePrice(const char *tickerSymbol, float value);
```

Generated WSDL file:

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

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

   <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 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="getLastTradePriseResponseMsg"
 *               action="urn:GetLastTradePrice") */
struct getLastTradePriceMsg {
   char tickerSymbol[10];
} getLastTradePrice;

struct getLastTradePriceResponseMsg {
   float return;
} getLastTradePriceResponse;
```

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"/>
      <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>

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

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

/* @WebFunction(operationName="SetTradePrice",
 *               action="urn:SetTradePrice")
 * @OneWay */
void setTradePrice(const char *tickerSymbol, float price);
```

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="SetTradePrice" type="tns:SetTradePrice"/>
      <xs:complexType name="SetTradePrice">
         <xs:sequence>
            <xs:element name="tickerSymbol" type="xs:string"/>
            <xs:element name="price" type="xs:float"/>
         </xs:sequence>
      </xs:complexType>
   </xs:schema>

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

   <portType name="StockQuote">
      <sca-c:bindings>
         <sca-c:prefix name="stockQuote"/>
      </sca-c:bindings>
      <operation name="SettTradePrice">
         <sca-c:bindings>
            <sca-c:function name="setTradePrice"/>
         </sca-c:bindings>
         <input name="SetTradePrice" message="tns:SetTradePrice">
         </input>
      </operation>
   </portType>

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

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

C.1.5 @WebParam

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

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

Format:

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

where:

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

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

Applies to: Function parameter or struct member.

Example:

Input C header file:

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

/* @WebFunction(operationName="GetLastTradePrice",
 *              action="urn:GetLastTradePrice")
 * @WebParam(paramName="tickerSymbol", name="symbol", mode="IN") */
float getLastTradePrice(char *tickerSymbol);
```

Generated WSDL file:

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

   <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>
         <wsdl:output name="GetLastTradePriceResponse">
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </binding>

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

C.1.6 @WebResult

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

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

Format:

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

where:

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

Applies to: Function.

Example:

Input C header file:

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

/* @WebFunction(operationName="GetLastTradePrice",
 *              action="urn:GetLastTradePrice")
 * @WebResult(name="price") */
float getLastTradePrice(const char *tickerSymbol);
```

Generated WSDL file:

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

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

   < message name="GetLastTradePrice">
```

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

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

   <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>

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

C.1.7 @SOAPBinding

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

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

Format:

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

where:

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

- ***parameterStyle : token (0..1) –*** specifies the WSDL parameter style. The default value is "*WRAPPED*".

Applies to: WebService, Function.

Example:

Input C header file:

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

…
```

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/">

   <portType name="StockQuote">
      <sca-c:bindings>
         <sca-c:prefix name="stockQuote"/>
      </sca-c:bindings>
   </portType>

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

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

C.1.8 @WebFault

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

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

Format:

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

where:

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

Applies to: struct.

Example:

Input C header file:

```
/* @WebFault(name="UnknownSymbolFault",
 *           targetNamespace="http://www.example.org/")
struct UnkSymMsg {
   char faultInfo[10];
} unkSymInfo;

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

/* @WebFunction(operationName="GetLastTradePrice",
 *              action="urn:GetLastTradePrice")
 * @WebThrows(faults="unkSymMsg") */
float getLastTradePrice(const char *tickerSymbol);
```

Generated WSDL file:

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

   <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:simpleType name="UnknownSymbolFaultType">
         <xs:restriction base="xs:string">
            <xsd:maxlength value="9"/>
         </xs:restriction>
      </xs:simpleType>
      <xs:element name="UnknownSymbolFault" type="UnknownSymbolFaultType"/>
   </xs:schema>

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

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

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

   <portType name="StockQuote">
      <sca-c:bindings>
         <sca-c:prefix name="stockQuote"/>
      </sca-c:bindings>
      <operation name="GetLastTradePrice">
         <sca-c:bindings>
            <sca-c:function name="getLastTradePrice"/>
         </sca-c:bindings>
         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
         </input>
         <output name="GetLastTradePriceResponse"
               message="tns:GetLastTradePriceResponse">
         </output>
         <fault name="UnknownSymbol” message="tns:UnknownSymbol">
         </fault>
      </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:fault>
            <soap:fault name="UnknownSymbol" use="literal"/>
         </wsdl:fault>
      </wsdl:operation>
   </binding>

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

C.1.9 @WebThrows

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

Corresponds to: No equivalent in JAX-WS.

Format:

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

where:

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

Applies to: Function or Operation

Example:

See @WebFault.

D C WSDL Mapping Extensions

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

D.1 <sca-c:bindings>

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

D.2 <sca-c:prefix>

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

Format:

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

where:

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

Applicable WSDL element(s):

- wsdl:portType

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

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>

   <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"/>
      <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>

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

Generated C header file:

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

/* @WebFunction(operationName="GetLastTradePrice",
 *              action="urn:GetLastTradePrice") */
float stockQuoteGetLastTradePrice(const char *tickerSymbol);
```

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>

   <portType name="StockQuote">
      <sca-c:bindings>
         <sca-c:prefix name="stockQuote"/>
         <sca-c:enableWrapperStyle>false</sca-c:enableWrapperStyle>
      </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>
</definitions>
```

Generated C header file:

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

/* @WebFunction(operationName="GetLastTradePrice",
 *               action="urn:GetLastTradePrice") */
   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:

```
 <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="getTradePrice"/>
         </sca-c:bindings>
         <input name="GetLastTradePrice" message="tns:GetLastTradePrice">
         </input>
         <output name="GetLastTradePriceResponse"
               message="tns:GetLastTradePriceResponse">
         </output>
      </operation>
   </portType>
</definitions>
```

Generated C header file:

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

/* @WebFunction(operationName="GetLastTradePrice",
 *                action="urn:GetLastTradePrice") */
   float getTradePrice(const wchar_t *tickerSymbol);
```

D.5 <sca-c:struct>

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

Format:

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

where:

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

Applicable WSDL element(s):

- wsdl:message

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

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

   < message name="GetLastTradePriceResponse">
      <sca-c:bindings>
         <sca-c:struct name="getTradePriceResponse"/>
      </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>
</definitions>
```

Generated C header file:

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

/* @WebOperation(operationName="GetLastTradePrice",
 *               response="getLastTradePriseResponse"
 *               action="urn:GetLastTradePrice") */
struct getLastTradePrice {
   wchar_t *tickerSymbol;  /* Since the length of the element is not
                            * restricted, a pointer is returned with the
                            * actual value held by the SCA runtime. */
};

struct getLastTradePriceResponse {
   float return;
};
```

D.6 <sca-c:parameter>

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

Format:

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

where:

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

Applicable WSDL element(s):

- wsdl:portType/wsdl:operation

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="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>
         <wsdl:output name="GetLastTradePriceResponse">
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </binding>

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

Generated C header file:

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

/* @WebFunction(operationName="GetLastTradePrice",
 *              action="urn:GetLastTradePrice")
 * @WebParam(paramName="tickerSymbol", name="symbol") */
   float getLastTradePrice(const wchar_t *tickerSymbol);
```

D.7 JAX-WS WSDL Extensions

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. [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

Table 3: Allowed JAX-WS Extensions

D.8 WSDL Extensions Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
   xmlns:sca-c="http://docs.oasis-open.org/ns/opencsa/sca-c-cpp/c/200901"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">

   <element name="bindings" type="sca-c:BindingsType" />
   <complexType name="BindingsType">
      <choice minOccurs="0" maxOccurs="unbounded">
         <element ref="sca-c:prefix" />
         <element ref="sca-c:enableWrapperStyle" />
         <element ref="sca-c:function" />
         <element ref="sca-c:struct" />
         <element ref="sca-c:parameter" />
      </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>
```

E XML Schemas

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

```
<?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>
```

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

```
<?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">
```

```
            <sequence>
               <element name="operation" type="sca:CImplementationFunction"
                     minOccurs="0" maxOccurs="unbounded" />
               <any namespace="##other" processContents="lax"
                     minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="module" type="NCName" use="required"/>
            <attribute name="path" type="string" use="optional"/>
            <attribute name="library" type="boolean" use="optional"/>
            <attribute name="componentType" type="string" use="required"/>
            <attribute name="scope" type="sca:CImplementationScope"
                  use="optional"/>
            <attribute name="eagerInit" type="boolean" use="optional"/>
            <attribute name="init" type="boolean" use="optional"/>
            <attribute name="destoy" type="boolean" use="optional"/>
            <attribute name="allowsPassByReference" type="boolean"
                  use="optional"/>
            <anyAttribute namespace="##other" processContents="lax"/>
         </extension>
      </complexContent>
   </complexType>

   <simpleType name="CImplementationScope">
      <restriction base="string">
         <enumeration value="stateless"/>
         <enumeration value="composite"/>
      </restriction>
   </simpleType>

   <complexType name="CImplementationFunction">
      <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="allowsPassByReference" type="boolean"
            use="optional"/>
      <attribute name="init" type="boolean" use="optional"/>
      <attribute name="destoy" type="boolean" use="optional"/>
      <anyAttribute namespace="##other" processContents="lax"/>
   </complexType>

</schema>
```

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

```
<?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-contributions.xsd"/>

   <element name="export.c" type="sca:CExport"
         substitutionGroup="sca:Export"/>

   <complexType name="CExport">
      <complexContent>
         <attribute name="name" type="QName" use="required"/>
         <attribute name="path" type="string" use="optional"/>
      </complexContent>
   </complexType>

   <element name="import.c" type="sca:CImport"
         substitutionGroup="sca:Import"/>

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

</schema>
```

F Conformance Points

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.

Conformance ID	Description
[C20016]	The SCA runtime MAY use by-reference semantics when passing input parameters, return values or exceptions on calls to remotable services within the same system address space if both the service function implementation and the client are marked “allows pass by reference”.
[C20017]	The SCA runtime MUST use by-value semantics when passing input parameters, return values and exceptions on calls to remotable services within the same system address space if the service function implementation is not marked “allows pass by reference” or the client is not marked “allows pass by reference”.
[C30001]	An SCA implementation MAY support proxy functions.
[C40001]	An operation marked as oneWay is considered non-blocking and the SCA runtime MAY use a binding that buffers the requests to the function and sends them at some time after they are made.
[C50001]	Vendor defined reason codes SHOULD start at 101.
[C60002]	An SCA runtime MAY additionally provide a DataObject variant of this API for handling properties with complex XML types. The type of the value parameter in this variant is DATAOBJECT.
[C60003]	A SCA runtime MAY provide the functions `SCAService()`, `SCAOperation()`, `SCAMessageIn()` and `SCAMessageOut()` to support C implementations in programs.
[C70001]	The @*name* attribute of a <*export.c/*> element MUST be unique amongst the <*export.c/*> elements in a domain.
[C70002]	The @*name* attribute of a <*import.c/*> child element of a <*contribution/*> MUST be unique amongst the <*import.c/*> elements in of that contribution.
[C80001]	The return type and types of the parameters of a function of a local service interface MUST be one of: • Any fundamental or compound types as defined by C.
[C80002]	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.
[C90001]	A C header file used to define an interface MUST: • Declare at least one function or message format struct
[C90002]	A C header file used to define an interface MUST NOT use the following constructs: • Macros

Conformance ID	Description
[C100001]	In the absence of customizations, an SCA implementation SHOULD map each portType to separate header file. An SCA implementation MAY use any sca-c:prefix binding declarations to control this mapping.
[C100002]	For components implemented in libraries, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the function.
[C100003]	In the absence of any customizations for a WSDL operation that does not meet the requirements for the wrapped style, the name of a mapped function parameter or struct member MUST be the value of the name attribute of the wsdl:part element with the first character converted to lower case.
[C100004]	In the absence of any customizations for a WSDL operation that meets the requirements for the wrapped sytle, the name of a mapped function parameter or struct member MUST be the value of the local name of the wrapper child with the first character converted to lower case.
[C100005]	For components implemented in a program, in the absence of customizations, an SCA implementation MUST concatenate the portType name, with the first character converted to lower case, and the operation name, with the first character converted to upper case, to form the request stuct name. Additionally an SCA implementation MUST append *"Response"* to the request struct name to form the response struct name.
[C100006]	In the absence of customizations, an SCA implementation MUST map the name of the message element referred to by a fault element to name of the struct describing the fault message content. If necessary, to avoid name collisions, an implementation MAY append "*Fault*" to the name of the message element when mapping to the struct name.
[C100007]	An SCA implementation SHOULD provide a default namespace mapping and this mapping SHOULD be configurable.
[C100008]	In the absence of customizations, an SCA implementation MUST map the header file name to the portType name. An implementation MAY append *"PortType"* to the header file name in the mapping to the portType name.
[C100009]	In the absence of customizations, an SCA implementation MUST map the function name to the operation name, stripping the portType name, if present and any namespace prefix from the front of function name before mapping it to the operation name.
[C100010]	In the absence of customizations, a struct with a name that does not end in *"Response"* or *"Fault"* is considered to be a request message struct and an SCA implementation MUST map the struct name to the operation name, stripping the portType name, if present, and any namespace prefix from the front of the struct name before mapping it to the operation name.
[C100011]	In the absence of customizations, an SCA implementation MUST map the parameter name, if present, to the part or global element component name. If the parameter does not have a name the SCA implementation MUST use argN as the part or global element child name.
[C100012]	In the absence of customizations, an SCA implementation MUST map the return type to a part or global element child named "*return*".

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 `struct`s. The mapping to and from `struct`s MUST follow the rules defined in WSDL to C Mapping Details.
[C100017]	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.
[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-wsdlext-c-1.1.xsd.

4217 F.1 Annotation Conformance Points

4218 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.

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 stuct 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/XMLSchemahttp://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"*.

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

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.

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 is treated as a reference to the C WSDL extensions defined in C WSDL Mapping Extensions	
2	Annotations on generated classes		[CF0002]
2.1	WSDL and XML Schema import directives		[CF0003]
2.1.1	Optional WSDL extensions		[CF0004]
2.2	SEI naming	[C100001]	

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]

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]	

Section	Conformance Point	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]

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

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

F.3.1 Ignored Conformance Points

Section	Conformance Point
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	Asychronous 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

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

G Migration

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.

G.1 Implementation.c attributes

@location has been replaced with *@path*.

G.2 SCALocate and SCALocateMultiple

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

H Acknowledgements

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

Participants:

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

I Revision History

[optional; should not be included in OASIS Standards]

Revision	Date	Editor	Changes Made
2		Bryan Aupperle	• Apply Changes for CCPP-75 and CCPP-76
1	30 April 2009	Bryan Aupperle	• Apply Changes for CCPP-62, CCPP-64, CCPP-66, CCPP-68 and CCPP-71