Implementation Type Documentation Requirements for SCA Assembly Model Version 1.1 Specification

Working Draft 05

15 June 2010

Specification URIs:

This Version:

http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation-**wd05**.html

http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation-**wd05**.odt

http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation-**wd05**.pdf

Previous Version:

Latest Version:

http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation.html

http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation.odt

http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation.pdf

Technical Committee:

OASIS Service Component Architecture / Assembly (SCA-Assembly) TC

Chair(s):

Martin Chapman, Oracle
Mike Edwards, IBM

Editor(s):

Dave Booz, IBM
Mike Edwards, IBM
Jeff Estefan, Jet Propulsion Laboratory

Related Work:

This document is related to:

- Service Component Architecture Assembly Specification Version 1.1

Declared XML Namespace(s):

none

Abstract:

This document defines the requirements for the documentation of an SCA implementation type that is used by a conforming SCA Runtime. The documentation describes how implementation artifacts of that implementation type relate to SCA components declared within SCA composites, as described by the SCA Assembly specification

Status:

This document was last revised or approved by the OASIS Service Component Architecture / Assembly (SCA-Assembly) 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-assembly/.

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-assembly/ipr.php.

The non-normative errata page for this specification is located at

http://www.oasis-open.org/committees/sca-assembly/

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 names "OASIS", "Service Component Architecture" are trademarks 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

[All text is normative unless otherwise indicated.]

This document defines the content of the documentation that is required to describe an SCA implementation type [SCA-Assembly], where that implementation type is supported by an SCA Runtime that claims to be conforming with the SCA Assembly specification.

The SCA Assembly specification defines an application in terms of service components that use and configure a particular implementation artifact. In order to fully define how a particular service component operates, it is necessary to describe the relationship between the configuration of the SCA component and the implementation technology used by the service component. It is the role of the Implementation Type Documentation to describe this relationship.

Some implementation types are described by formal specifications that have been created by OASIS SCA technical committees. Examples include:

- SCA WS-BPEL Client and Implementation V1.1 [SCA-BPEL]
- SCA POJO Component Implementation V1.1 [SCA-POJO]

1.1 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119].

1.2 Normative References

[RFC 2119] ——S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

[SCA-Assembly] ——OASIS Committee Draft 05, Service Component Architecture Assembly Model Specification Version 1.1, January 2010. http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd05.pdf

[SCA-BPEL] ——OASIS Committee Draft 02, Service Component Architecture WS-BPEL Client and Implementation Specification Version 1.1, March 2009. http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec-cd-02.pdf

[SCA-POJO] ——OASIS Committee Draft 02, Service Component Architecture POJO Component Implementation Specification Version 1.1, February 2010. http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-cd02.pdf

[SCA-CPP] ——OASIS Committee Draft 02, Service Component Architecture Client and Implementation Model for C++ Specification Version 1.1, March 2010. -http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd05.pdf

[SCA-JAVACAA] OASIS Committee Draft 04, Service Component Architecture SCA-J Common Annotations and APIs Specification 1.1, February 2010. http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.pdf

1.3 Non-normative References

None

2 Describing an SCA Implementation Type

This document defines the information that is needed for a particular implementation type to be used as a service component implementation within an SCA assembly. The information covers static configuration information required in order to use an implementation type and it's associated artifacts in an SCA assembly and it also covers the dynamic runtime behaviour of instances of the implementation type when the SCA assembly is executed by an SCA Runtime.

While this document gives a general description of the information that needs to be provided for an implementation type, the OASIS SCA technical committees have also produced examples of specifications that provide this same level of information for a variety of implementation technologies. These specifications can provide guidance in creating a document with the appropriate level of information for a new implementation type:

- SCA WS-BPEL Client and Implementation V1.1 [SCA-BPEL], which describes implementations built as WS-BPEL scripts
- SCA POJO Component Implementation V1.1 [SCA-POJO], which describes implementations based on simple Java classes.

2.0.1 What is an Implementation Type?

An implementation type describes how the artifacts of a concrete implementation technology are used to implement SCA components. Implementation types also describe the relationship between a technology specific implementation and the foundational aspects of SCA components, namely services, references, and properties.

Often an implementation type is defined such that it describes all SCA component implementations that use a particular implementation language, such as C++ [SCA-CPP] or BPEL [SCA-BPEL]. However, SCA is flexible and allows multiple implementation types to use the same implementation language. Examples of this occur with the Java language, where implementation types exist for POJO classes [SCA-POJO], for EJBs [SCA-JEE] and for Spring classes [SCA-SPRING]. As a result, the implementation type can represent a specialized form of an implementation technology, where the specialization may involve the use of specific APIs, frameworks or specific language extensions.

2.0.2 How an Implementation is used in SCA

SCA describes applications in terms of assemblies of service components. Service components are declared within SCA composites. Every component ~~must~~ uses an implementation - which is expressed as a reference to an artifact that provides a runtime implementation of the service component contract.

A typical SCA component is shown in Listing 1:

```
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
           targetNamespace=
              "http://docs.oasis-open.org/ns/opencsa/scatests/200903"
           name="TestComposite4">

    <component name="ComponentA">
       <implementation.java class="org.oasisopen.sca.Service1Impl"/>
         <service name="Service1">
           <interface.java interface="org.oasisopen.sca.Service1"/>
         </service>
         <property name="serviceName" value="AService"/>
         <reference name="reference1"/>
    </component>

</composite>
```

Listing 1: Example SCA component

The component "ComponentA" has an implementation, which in this example is a Java POJO implementation, declared using the <implementation.java/> element. The implementation.java element contains a reference to the implementation artifact, which in this example is a Java class with the name "Service1Impl" in the package "org.oasisopen.sca".

The remainder of the contents of the component declaration is configuration that is applied to the implementation at runtime. The component can declare all the services, references and properties of the implementation and apply configuration information to each of them. This can include things such as bindings for services and references and property values for properties.

Note that the configurable aspects of an SCA component implementation are called the componentType of the implementation - basically, it is the set of services, references and properties that the implementation has - for details of the componentType see the section "The ComponentType of an Implementation Artifact"

2.1 Describing the Implementation extension element

The implementation type documentation MUST describe the XML element that is used when declaring implementations of that type in an SCA component. [IMP10001] It is highly recommended that the element is defined in a namespace that is owned by the same entity that owns the definition of the implementation type.

The name used for the implementation element MUST to be unique - it MUST NOT use the same name as any other implementation type. [IMP10002] The name can be derived from the programming language used for the implementation type (e.g. "python" or "ruby") or it can be derived from the technology used in the implementation (e.g. "spring"). By convention, the OASIS SCA technical committees have adopted a naming convention that forms an implementation extension element name by concatenating the string “implementation.” with the informal name of the implementation type. For example, <implementation.java/> represents the SCA POJO [SCA-POJO] implementation type. It is highly recommended that the name used for the element follows this convention.

Formally, the implementation extension element is an element that is in the substitution group of the <sca:implementation/> element defined in the sca-core.xsd defined by the SCA Assembly specification [SCA-Assembly]. The declaration of the sca:implementation element is~~as~~ shown in Listing 2:

```
<!-- Implementation -->
<element name="implementation" type="sca:Implementation" abstract="true"/>
<complexType name="Implementation" abstract="true">
   <complexContent>
      <extension base="sca:CommonExtensionBase">
        <choice minOccurs="0" maxOccurs="unbounded">
           <element ref="sca:requires"/>
           <element ref="sca:policySetAttachment"/>
        </choice>
         <attribute name="requires" type="sca:listOfQNames"
                    use="optional"/>
         <attribute name="policySets" type="sca:listOfQNames"
                    use="optional"/>
      </extension>
   </complexContent>
</complexType>
```

Listing 2: Declaration of base <implementation/> element and Implementation type.

The implementation extension element MUST be declared as an element in the substitution group of the sca:implementation element. [IMP10003] The implementation extension element MUST be declared to be of a type which is an extension of the sca:Implementation type. [IMP10004]

The <implementation.java/> element declaration can serve as a useful model, as shown in Listing 3:

```
<!-- Java Implementation -->
<element name="implementation.java" type="sca:JavaImplementation"
         substitutionGroup="sca:implementation"/>
<complexType name="JavaImplementation">
   <complexContent>
      <extension base="sca:Implementation">
         <sequence>
            <any namespace="##other" processContents="lax"
                 minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
         <attribute name="class" type="NCName" use="required"/>
      </extension>
   </complexContent>
</complexType>
```

Listing 3: Declaration of <implementation.java/> element

It is recommended that tThe implementation extension element element ~~should~~ allows for attributes and/or subelements which describe the implementation artifact to be used as the SCA component implementation, such as the @class attribute of <implementation.java/>. If necessary, one or more attributes and subelements can be used to describe the implementation artifact (other than the configuration of services, references and properties supplied by the component).

Regarding the location of the implementation artifact, the location SHOULD always be taken as relative to the SCA contribution which contains the composite holding the component declaration. [IMP10005]

2.2 The ComponentType of an Implementation Artifact

For a implementation of any type, its features that relate to SCA component concepts are declared in the implementation artifact's componentType [SCA-Assembly].

The implementation type documentation MUST define how the componentType is defined for any given implementation artifact that is used with the implementation type. -[IMP10006]

There are two general approaches to defining the componentType:

1. calculate the componentType by introspecting the implementation artifact itself
2. provide a separate componentType file which contains a full declaration of the componentType for the given implementation artifact

An example of the introspection approach is shown in the componentType section of the Java POJO implementation type specification [SCA-POJO]. An example of the approach using a separate componentType file is shown in the SCA Client and Implementation Model for C++ [SCA-CPP].

In either case, the implementation type documentation MUST describe how the componentType is related to the content of the implementation artifact itself, both in terms of the base content of the artifact and also the impact of any SCA-specific language extensions and customizations that are available for use with an implementation of this type. [IMP10007]

The <sca:componentType/> element is declaration is shown in Listing 4:

```
<!-- Component Type -->
<element name="componentType" type="sca:ComponentType"/>
<complexType name="ComponentType">
   <complexContent>
      <extension base="sca:CommonExtensionBase">
         <sequence>
            <element ref="sca:implementation" minOccurs="0"/>
            <choice minOccurs="0" maxOccurs="unbounded">
               <element name="service" type="sca:ComponentService"/>
               <element name="reference"
```

```
                    type="sca:ComponentTypeReference"/>
                <element name="property" type="sca:Property"/>
            </choice>
            <any namespace="##other" processContents="lax" minOccurs="0"
                 maxOccurs="unbounded"/>
        </sequence>
    </extension>
  </complexContent>
</complexType>
```

Listing 4: ComponentType declaration.

In essence, the componentType of an implementation declares the services, the references and the properties of the implementation artifact, which are customized by a component that uses the implementation.

2.2.1 Support for Bidirectional Interfaces and for Long Running Request/Response operations

An important feature of the SCA model is its capability of defining service interactions between components that are asynchronous in nature - where the timing and/or the type of a response to a request can vary. There are two main aspects of SCA which support this

- Bidirectional interfaces
- Long-Running Request/Response operations

The implementation type documentation MUST describe how the SCA Aspects of Bidirectional Interfaces and Long Running Request/Response Operations are handled both for a component which is a service client and also for a component which is a service provider. [IMP10008]

2.3 SCA Extensions and Customizations for Implementation Artifacts

An implementation type can either simply use the existing features of a particular implementation language (e.g. the Java language, the C++ language), or it may provide some SCA-specific extensions or customizations that can be useful to the programmer when creating implementation artifacts that are designed for use with SCA. These extensions and customizations may affect the componentType of an implementation artifact and/or affect the runtime behavior of the artifact. Examples of extensions and customizations include SCA-specific annotations and SCA-related APIs.

If SCA-specific extensions or customizations are available for an implementation type, the implementation type documentation MUST describe all of the available extensions and customizations. [IMP10009] The implementation type documentation MUST describe the impact of any extensions and customizations on the componentType of the implementation artifact. [IMP10010] The implementation type documentation MUST describe the impact of any extensions and customizations on the runtime behaviour of the implementation artifact.**[IMP10011]**

An example of an extension can be seen in the Java POJO specification [SCA-POJO] with the @Reference annotation, which allows a programmer to mark a field, a constructor parameter or a setter method as an SCA reference.

2.4 Describing the Runtime Behaviour of an Implementation Artifact

The implementation type documentation MUST describe the runtime behaviour of instances of SCA components which use implementation artifacts described by the implementation type. [IMP10012]

In particular, the documentation MUST describe how the SCA component configuration affects the configuration of a component instance at runtime - how services are invoked, how references are obtained and how they are invoked, how property values are mapped to types in the implementation's instance and how the values are obtained by the component implementation. [IMP10013]

The lifecycle of runtime instances MUST be described - when implementation instances are created, how long they live and when they are destroyed, in relation to the containing SCA component and in relation to service invocations related to the component. [IMP10014] The number of instances belonging to a single component MUST be described along with any serialization and multi-threading considerations. [IMP10015]

If there are runtime exceptions or faults that apply to implementation type artifacts, these MUST be described by the implementation type documentation. [IMP10016]

2.5 Describing an Interface Type associated with an Implementation Type

An implementation type might have an associated interface type which it uses when describing the interfaces of services and references. If the implementation type is able to use an existing interface type, e.g., interface.wsdl or interface.java, then the implementation type documentation can simply reference the documentation for that interface type.

if the implementation type uses an interface type that is not described in the documentation for some existing implementation type, then the implementation type documentation MUST describe the interface type. [IMP10017]

For some new interface type, there are essentially two pieces of information to provide:

- a definition of the interface extension element
- a definition of the mapping of the interface type to interface.wsdl.

All remotable interfaces MUST be mappable to interface.wsdl. [IMP10030]

It is highly recommended that the interface extension element is defined in a namespace that is owned by the same entity that owns the definition of the interface type.

The name used for the interface extension element needs to be unique - it MUST NOTuse the same name as any other interface type. [IMP10031] The name can be derived from the programming language used for the interface type (e.g. "java") or it can be derived by any other means that makes sense in the context of the interface type.

Describing the interface extension element is similar in nature to describing an implementation extension element. The interface extension element must be declared as an element in the substitution group of the <sca:interface/> element. [IMP10018] The interface extension element must be declared to be of a type which is an extension of the sca:Interface type. [IMP10019] The base <sca:interface/> element and sca:Interface type are defined in sca-core.xsd by the SCA Assembly specification [SCA-Assembly] and are shown in Listing 5:

```
<!-- Interface -->
<element name="interface" type="sca:Interface" abstract="true"/>
<complexType name="Interface" abstract="true">
   <complexContent>
      <extension base="sca:CommonExtensionBase">
         <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="sca:requires"/>
            <element ref="sca:policySetAttachment"/>
         </choice>
         <attribute name="remotable" type="boolean" use="optional"/>
        <attribute name="requires" type="sca:listOfQNames"
```

```
                use="optional"/>
            <attribute name="policySets" type="sca:listOfQNames"
                use="optional"/>
        </extension>
    </complexContent>
</complexType>
```

Listing 5: Declaration of base interface element and Interface type.

By convention, the OASIS SCA technical committees have adopted a naming convention that forms an interface extension element name by concatenating the string “interface.” with the informal name of the interface type. For example, the <interface.java/> element declaration from the SCA Common Annotations and APIs specification [SCA-JAVACAA] can serve as a useful model, as shown in Listing 6:

```
<!-- Java Interface -->
<element name="interface.java" type="sca:JavaInterface"
         substitutionGroup="sca:interface"/>
<complexType name="JavaInterface">
   <complexContent>
      <extension base="sca:Interface">
         <sequence>
            <any namespace="##other" processContents="lax" minOccurs="0"
                 maxOccurs="unbounded"/>
         </sequence>
         <attribute name="interface" type="NCName" use="required"/>
         <attribute name="callbackInterface" type="NCName"
                    use="optional"/>
      </extension>
   </complexContent>
</complexType>
```

Listing 6: Declaration of the interface.java element.

Note that the <interface.java/> element is in the substitution group of <sca:interface/> and its type is an extension of the sca:Interface type.

The interface extension element MUST allow for attributes and/or subelements which describe the interface artifact. [IMP10020] Examples of interface extension attributes include the @interface and @callbackInterface attributes of <interface.java/>. If necessary, one or more attributes and subelements can be used to configure the interface artifact.

Regarding the location of the interface artifact, the location SHOULD always be taken as relative to the SCA contribution which contains the composite holding the component declaration. [IMP10021]

2.5.1 Support of Local and Remotable Interfaces

The SCA Assembly specification [SCA-Assembly] defines the concepts of ***local*** and ***remotable*** interfaces. Where a new interface type is defined, the implementation type documentation MUST define how the concepts of local and remotable interfaces apply to the interface type. [IMP10022]

2.5.2 Interface Compatibility rules

The compatibility of two interface declarations is an important part of the SCA model. This is discussed in detail in the SCA Assembly specification [SCA-Assembly]. Where a new interface type is defined, the implementation type documentation MUST define the compatibility rules for the interface type, including superset interfaces, subset interfaces and equal interfaces. [IMP10023]

2.6 Describing the behavior of Implementation artifacts within Contributions

Artifacts of all types are made available for use in an SCA application by means of ***contributions*** which are deployed into the SCA Domain used by the SCA Runtime, Contributions are defined in the SCA Assembly specification [SCA-Assembly]. Essentially, a contribution is a collection of artifacts that are organized into a hierarchy based off a single root.

Whenever a reference is made to an artifact of a particular implementation type, for example a reference within an implementation type element, that artifact MUST be found within the contributions deployed into the domain. [IMP10024]

The default location for an artifact is within the SCA contribution where the reference is made - i.e. where the implementation type element appears in a composite file within a particular contribution, that same contribution is searched. It is expected that the implementation type element contains configuration that identifies the artifact. This identification can take the form of a filename or package name, which can include the hierarchy path for the artifact (eg directory path or Java package name). Alternatively, the identification may involve a namspace, where the assumption is that all artifacts of a given type are searched to find a matching namespace and element name, as occurs for XML artifacts (e.g. BPEL processes).

The implementation type documentation MUST describe the way in which the artifact reference information is used to locate a specific artifact. [IMP10025] The implementation type documentation must describe the permitted organization of the implementation type artifacts within a contribution. [IMP10026]

Some implementation types can also allow for implementation artifacts to be imported into one contribution from a second (exporting) contribution, as described in the "SCA Artifact Resolution" section of the SCA Assembly specification [SCA-Assembly]. Where this is supported, the implementation type documentation MUST describe how the import works. [IMP10027] Import and Export of artifacts can either follow the base mechanism described in the SCA Assembly specification, which is based on the use of namespaces, or it may follow an implementation-type specific mechanism.

The base mechanism involves the declaration of <sca:export/> and <sca:import/> elements with an sca-contribution.xml file that is in the META-INF directory of the contribution. It is recomended that an extension of the base mechanism is used.

2.6.1 Implementation-Type specific forms of Import and Export

Where an implementation type requires the use of a specific form of import and export mechanism for the resolution of artifacts between contributions, the implementation type documentation is required to define how this works.

An example of such a mechanism exists for the Java POJO implementation type [SCA-POJO].There are base importBase and exportBase elements and types defined in the SCA Assembly specification [SCA-Assembly]. For the Java POJO implementation, <import.java/> and <export.java/> elements are defined as shown in Listing 7:

```
   <!-- Import.java -->
<element name="import.java" type="sca:JavaImportType"
substitutionGroup="sca:importBase" />
   <complexType name="JavaImportType">
      <complexContent>
         <extension base="sca:Import">
            <attribute name="package" type="string" use="required"/>
            <attribute name="location" type="anyURI" use="optional"/>
         </extension>
      </complexContent>
   </complexType>

   <!-- Export.java -->
```

```
<element name="export.java" type="sca:JavaExportType"
 substitutionGroup="sca:exportBase" />
<complexType name="JavaExportType">
   <complexContent>
      <extension base="sca:Export">
         <attribute name="package" type="string" use="required"/>
      </extension>
   </complexContent>
</complexType>
```

Listing 7: Definition of the <import.java/> and <export.java/> elements

If using an extension of the base SCA mechanism for imports and exports, the implementation type documentation must define import and export elements that extend the base Import and Export types. [IMP10028]

By convention, the OASIS SCA technical committees have adopted a naming convention that forms import and export extension element names by concatenating the strings “import.” and "export." with the informal name of the implementation type.

2.6.2 Implementation-Type specific forms of Contribution

One format of contribution packaging is mandatory - the ZIP file contribution format. However, SCA allows for many other contribution formats. If an implementation type has a specialized contribution format, then the implementation type documentation MUST provide a definition of that format. [IMP10029]

3 Conformance

Implementation Type Documentation that claims to conform to the requirements of this specification MUST meet the following conditions

1. The Implementation Type Documentation MUST comply with all the mandatory statements listed in in the table Mandatory Items in the appendix "Conformance Items"

A. Conformance Items

This section contains a list of conformance items for the SCA Assembly Implementation Type Documentation specification.

A.1. Mandatory Items

Conformance ID	Description
[IMP10001]	The implementation type documentation MUST describe the XML element that is used when declaring implementations of that type in an SCA component.
[IMP10002]	The name used for the implementation element MUST to be unique - it MUST NOT use the same name as any other implementation type.
[IMP10003]	The implementation extension element MUST be declared as an element in the substitution group of the sca:implementation element.
[IMP10004]	The implementation extension element MUST be declared to be of a type which is an extension of the sca:Implementation type.
[IMP10005]	Regarding the location of the implementation artifact, the location SHOULD always be taken as relative to the SCA contribution which contains the composite holding the component declaration.
[IMP10006]	The implementation type documentation MUST define how the componentType is defined for any given implementation artifact that is used with the implementation type.
[IMP10007]	the implementation type documentation MUST describe how the componentType is related to the content of the implementation artifact itself, both in terms of the base content of the artifact and also the impact of any SCA-specific language extensions and customizations that are available for use with an implementation of this type.
[IMP10008]	The implementation type documentation MUST describe how the SCA Aspects of Bidirectional Interfaces and Long Running Request/Response Operations are handled both for a component which is a service client and also for a component which is a service provider.
[IMP10009]	If SCA-specific extensions or customizations are available for an implementation type, the implementation type documentation MUST describe all of the available extensions and customizations.
[IMP10010]	The implementation type documentation MUST describe the impact of any extensions and customizations on the componentType of the implementation artifact.
[IMP10011]	The implementation type documentation MUST describe the impact of any extensions and customizations on the runtime behaviour of the implementation artifact.
[IMP10012]	The implementation type documentation MUST describe the runtime behaviour of instances of SCA components which use implementation artifacts described by the implementation type.

[IMP10013]	In particular, the documentation MUST describe how the SCA component configuration affects the configuration of a component instance at runtime - how services are invoked, how references are obtained and how they are invoked, how property values are mapped to types in the implementation's instance and how the values are obtained by the component implementation.
[IMP10014]	The lifecycle of runtime instances MUST be described - when implementation instances are created, how long they live and when they are destroyed, in relation to the containing SCA component and in relation to service invocations related to the component.
[IMP10015]	The number of instances belonging to a single component MUST be described along with any serialization and multi-threading considerations.
[IMP10016]	If there are runtime exceptions or faults that apply to implementation type artifacts, these MUST be described by the implementation type documentation.
[IMP10017]	if the implementation type uses an interface type that is not described in the documentation for some existing implementation type, then the implementation type documentation MUST describe the interface type.
[IMP10018]	The interface extension element must be declared as an element in the substitution group of the <sca:interface/> element.
[IMP10019]	The interface extension element must be declared to be of a type which is an extension of the sca:Interface type.
[IMP10020]	The interface extension element MUST allow for attributes and/or subelements which describe the interface artifact.
[IMP10021]	Regarding the location of the interface artifact, the location SHOULD always be taken as relative to the SCA contribution which contains the composite holding the component declaration.
[IMP10022]	Where a new interface type is defined, the implementation type documentation MUST define how the concepts of local and remotable interfaces apply to the interface type.
[IMP10023]	Where a new interface type is defined, the implementation type documentation MUST define the compatibility rules for the interface type, including superset interfaces, subset interfaces and equal interfaces.
[IMP10024]	Whenever a reference is made to an artifact of a particular implementation type, for example a reference within an implementation type element, that artifact MUST be found within the contributions deployed into the domain.
[IMP10025]	The implementation type documentation MUST describe the way in which the artifact reference information is used to locate a specific artifact.
[IMP10026]	The implementation type documentation must describe the permitted organization of the implementation type artifacts within a contribution.

[IMP10027]	Some implementation types can also allow for implementation artifacts to be imported into one contribution from a second (exporting) contribution, as described in the "SCA Artifact Resolution" section of the SCA Assembly specification [SCA-Assembly]. Where this is supported, the implementation type documentation MUST describe how the import works.
[IMP10028]	If using an extension of the base SCA mechanism for imports and exports, the implementation type documentation must define import and export elements that extend the base Import and Export types.
[IMP10029]	If an implementation type has a specialized contribution format, then the implementation type documentation MUST provide a definition of that format.
[IMP10030]	All remotable interfaces MUST be mappable to interface.wsdl
[IMP10031]	The name used for the interface extension element needs to be unique - it MUST NOTuse the same name as any other interface type.

B. Acknowledgments

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

Participants:

- Mike Edwards, IBM
- Dave Booz, IBM
- Jeff Estefan, Jet Propulsion Laboratory

C. Non-Normative Text

D. Revision History

Revision	Date	Editor	Changes Made
1	19/04/10	Mike Edwards	Initial version created
2	23/04/10	Dave Booz Mike Edwards	Extensive revisions and updates. Completion of the section on contributions Additional sections on Interfaces and Implementations
3	13/05/10	Jeff Estefan	Clean-up and consistency
4	17/05/10	Mike Edwards	Formal normative statements. Added conformance items appendix.