WSDK

WSDK: Web Services Interoperability.

Version 1.0 November 17, 2005

Table of Contents

1 Introduction

1.1 Web Services

Web services are governed by the SOAP specification. They are defined by WSDL files, following the WSDL specification. WSDL files use XML Schema for type and element definitions.

XML Schema (XS) is a broad, complex specification, written to address the full spectrum of XML use cases. Many of its advanced features are not often used in web service applications. Different tools and services use varying conventions.

It is considered “good practice” for web service interoperability to start with a definition of message content, using XSD and WSDL, which employs a subset “profile” of standard XS features. Then server and client implementations (using various programming languages and tools) may be developed based on that definition. However, many tools offer the reverse method: i.e. deriving the WSDL and XS definitions from the implementation in the particular programming language. This is convenient when all endpoints of the web service use the same conventions, but this approach is more likely to encounter interoperability problems arising from the mapping of XML Schema types between implementations.

1.2 WS-I Standards

The WSDK attempts to implement the WS-I Basic Profile (http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html), though it is possible that some features are not fully implemented.

Conformance with standard is intended to improve interoperability between endpoints of a web service; the standard makes the following note:

> “It is impossible to completely guarantee the interoperability of a particular service. However, the Profile does address the most common problems that implementation experience has revealed to date. “

The WS-I web site provides source for sample applications, and some test tools for evaluating a service against the Basic Profile.

Many interoperability issues arise from handling the types specified by XML Schema - this report summarizes how the Curl WSDK 1.0 WSDL package represents XML Schema types in Curl.

2 XML Schema Type correspondence

2.1 Simple Types

All primitive simple types are supported. XMLSimpleValue is used for some infrequently used types that do not have direct equivalents as standard Curl types,. XMLSimpleValue gives access to the underlying String value, and the ascribed XML Schema datatype.

XML Type	Curl Type
xs:string	String
xs:boolean	bool
xs:int	int
xs:double	double
xs:float	float
xs:long	int64
xs:short	int16
xs:byte	int8
xs:base64Binary	{Array-of uint8}
xs:hexBinary	{Array-of uint8}
xs:unsignedShort	uint16
xs:unsignedByte	uint8
xs:integer	int64
xs:positiveinteger	int64
xs:negativeinteger	int64
xs:nonPositiveinteger	int64
xs:nonNegativeinteger	int64
xs:unsignedInt	int64
xs:unsignedLong	int64
xs:decimal	double
xs:dateTime	DateTime
xs:anyURI	Url
xs:QName	XDMName
xs:date	XMLSimpleValue
xs:time	XMLSimpleValue
xs:duration	XMLSimpleValue
xs:gYear	XMLSimpleValue
xs:gYearMonth	XMLSimpleValue
xs:gMonth	XMLSimpleValue
xs:gMonthDay	XMLSimpleValue
xs:Day	XMLSimpleValue
xs:normalizedString	XMLSimpleValue
xs:token	XMLSimpleValue
xs:language	XMLSimpleValue
xs:Name	XMLSimpleValue
xs:NCName	XMLSimpleValue
xs:ENTITY	XMLSimpleValue
xs:ENTITIES	XMLSimpleValue
xs:NMTOKEN	XMLSimpleValue
xs:NMTOKENS	XMLSimpleValue
xs:ID	XMLSimpleValue
xs:IDREF	XMLSimpleValue
xs:IDREFS	XMLSimpleValue

Derived simple types are supported as follows:

- Types derived by enumeration are represented as the equivalent Curl 'define-enum'
- Types derived by restriction are represented as an alias to the underlying primitive type.
- Types derived by list are represented by arrays parameterized by the underlying base type.
- Types derived by union are represented as an alias to 'any'.

2.2 Complex Types

In general, complex types are represented as classes, with fields corresponding to the specified elements and attributes. The type of each field is based on the member (element or attribute) type. The treatment of members depends on the cardinality and content model.

In some cases, however, when the specified structure does not have a direct equivalent as a class, a different representation is used (XDMElement or HashTable). These cases are noted below.

2.2.1 Cardinality

A required element (minOccurs = 1 maxOccurs = 1) is represented as a non-nullable field.

An optional element (minOccurs = 0 maxOccurs = 1) is represented as a nullable field.

A repeating element (maxOccurs > 1) is represented as a parameterized array.

An element specified as 'nillable' is always represented as a nullable field.

2.2.2 Complex Content

Elements specified in model groups 'all' and 'sequence' are represented as fields in the class.

Elements specified in 'choice' model groups result in the use of 'HashTable' (rather than a specific class) to represent the container.

Elements specified indirectly, by reference to 'group' definitions, are expanded into the constituent elements.

Arbitrary elements, specified using the 'any' wildcard result in the use of 'XDMElement' (rather than a specific class) to represent the container.

Attributes are also represented as fields.

Attributes specified indirectly, by reference to 'attributeGroup' definitions, are expanded into the constituent attributes.

Arbitrary attributes, specified using the 'anyAttribute' wildcard result in the use of 'XDMElement' (rather than a specific class) to represent the container.

Mixed content results in the use of 'XDMElement' (rather than a specific class).

Types derived by extension result in a class definition which inherits from the class corresponding to the specified base type.

Types derived by restriction result, in general, in the use of 'XDMElement' (rather than a specific class). When the definition matches the pattern conventionally used to describe an encoded array, it is represented as a parameterized array.

2.2.3 Simple Content

This is used to describe "atomic" elements, which contain only a text element (and possibly attributes). The text value is represented using a field named 'content'.

2.3 Global Definitions

XML schema allows structure to be specified using globally named elements. These are treated similarly to type definitions.

Globally named attributes are treated similarly to attribute groups with a single member.

2.4 Local Definitions

XML Schema allows structure to be specified locally, without any global type. When the representation of such a structure would correspond to a class definition, a name based on the element name is used.

2.5 Schema Composition

A schema or WSDL file may incorporate definitions from other schema files by use of 'import' and 'include'. These are processed similarly to the top level schema definition. Certain "well known" SOAP/WSDL schema definitions are not explicitly imported.

- "http://www.w3.org/2001/XMLSchema"
- "http://schemas.xmlsoap.org/wsdl/"
- "http://schemas.xmlsoap.org/soap/encoding/"
- "http://www.w3.org/XML/1998/namespace"

2.6 Names

A legal XML name might not correspond to legal Curl name. (The allowed character sets are different; XML names include the namespace; naming conventions vary between programming languages and styles).

When web service definitions are internalized into Curl, legal Curl names, following standard Curl conventions, are used. Name conflicts are checked, and resolved by addition of namespace prefixes or numeric suffixes.

2.7 SOAP encoding

SOAP allows operation inputs and outputs to be interpreted without use of explicit XS types according to an encoding. This 'encoded' usage is discouraged by the WS-I interoperability guidelines, in favor of the more explicit 'literal' usage described above.

The conventions used for 'encoded' services are as follows. The SOAP spec allows for arbitrary encoding styles, identified by the value of the 'encodingStyle' attribute. In practice, the default encoding style that is always used is defined in the specification: http://schemas.xmlsoap.org/soap/encoding/.

Message elements use the 'xsi:type' attribute to describe their type. Message elements marked with 'xsi:nil' true are treated as nulls. Array elements use 'soapenc:Array' for 'xsi:type' value and 'soapenc:arrayType' with a value containing the element type concatenated with dimensions. (The spec also has provisions for sparse arrays and nonstandard offsets, but these are rarely used, and are not supported by the WSDK.)

Message elements may be shared ("multiref"): they are specified once, with an unique 'id' attribute, and reference elsewhere, via a 'ref' attribute. The original element may be "in place" in the message, or "freestanding" in the message envelope outside the message body. Some services and tools use “multiref” encoding even for primitive values, or for values which appear only once.

2.8 Service Method Signatures

The Curl service class corresponding to a WSDL service port defines methods for each SOAP operation (there are two, for synchronous and asynchronous requests). The signatures of these methods (arguments and return values) are derived from ‘input’ and ‘output’ message definitions in the WSDL file. The conventions for the messages vary, depending on whether the operation style is “rpc” or “document”. For “document” style operations, the input and output messages are expected to contain a single ‘part’, which refers to a defined ‘element’ in the types. For “rpc” style operations, the messages may contain multiple parts: each ‘part’ of the input message describes an argument; each ‘part’ in an output message describes a return value.

SOAP messages can contain headers. When headers are specified, input headers are represented as keyword arguments, and output headers as additional return values.

3 WSDK Interoperability

3.1 Known Issues

The following issues have been identified:

- Repeating elements must be nested in their own container.

This is a WSDK1.0 limitation. XML schema has no limitation of the cardinality of elements in a model group. However, it is considered good practice for web service interoperability to nest repeated elements in their own container.

- The names used for elements and types must be different.

This is a WSDK 1.0 limitation. XML Schema distinguishes element and type names. However, it is common practice to use a naming convention that distinguishes them.

3.2 XS Coverage

The following features are rarely used in web service definitions, and may be more likely to encounter interoperability issues and unexpected behavior from the WSDK

- xs:union
- xs:choice
- xs:list
- xs:attributeGroup
- xs:anyAttribute
- xs:any

Similarly, the use of SOAP headers is not frequent, so services employing headers may also encounter interoperability issues.

3.3 Diagnostic Information

When investigating and reporting web service interoperability issues or other problems, the following information is useful to help identify the problem

1. The WSDL file, and any referenced files (includes or imports).
2. Sample request / response messages. These can be obtained in Curl using the HTTP Monitor tool.

4 Examples

4.1 XS patterns with Curl representation

4.1.1 structure

4.1.1.1 XS

```
<xs:complexType name="ResultElement">
 <xs:all>
   <xs:element name="summary" type="xs:string"/>
   <xs:element name="URL" type="xs:string"/>
   <xs:element name="snippet" type="xs:string"/>
   <xs:element name="title" type="xs:string"/>
   <xs:element name="cachedSize" type="xs:string"/>
   <xs:element name="relatedInformationPresent" type="xs:boolean"/>
   <xs:element name="hostName" type="xs:string"/>
   <xs:element name="directoryCategory" type="typens:DirectoryCategory"/>
   <xs:element name="directoryTitle" type="xs:string"/>
 </xs:all>
</xs:complexType>
```

4.1.1.2 Curl

```
{define-class public open ResultElement
  field public summary:String = ""
  field public url:String = ""
  field public snippet:String = ""
  field public title:String = ""
  field public cached-size:String = ""
  field public related-information-present:bool
  field public host-name:String = ""
  field public directory-category:DirectoryCategory = {uninitialized-value-for-
type DirectoryCategory}
  field public directory-title:String = ""
}
```

4.1.2 array (literal)

4.1.2.1 XS

```
<xs:complexType name="ArrayOfPlaceFacts">
 <xs:sequence>
   <xs:element minOccurs="0" maxOccurs="unbounded" name="PlaceFacts"
type="tnxs:PlaceFacts" />
 </xs:sequence>
</xs:complexType>
```

4.1.2.2 Curl

```
{let public constant ArrayOfPlaceFacts:ClassType =
     {Array-of PlaceFacts}
}
```

4.1.3 array (encoded)

4.1.3.1 XS

```
<xs:complexType name="ResultElementArray">
 <xs:complexContent>
   <xs:restriction base="soapenc:Array">
      <xs:attribute ref="soapenc:arrayType"
wsdl:arrayType="typens:ResultElement[]"/>
   </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

4.1.3.2 Curl

```
{let public constant ResultElementArray:ClassType =
     {Array-of ResultElement}
}
```

4.1.4 enumeration

4.1.4.1 XS

```
<xs:simpleType name="PlaceType">
 <xs:restriction base="xs:string">
   <xs:enumeration value="UnknownPlaceType" />
   <xs:enumeration value="AirRailStation" />
   <xs:enumeration value="BayGulf" />
   <xs:enumeration value="CapePeninsula" />
   <xs:enumeration value="CityTown" />
   <xs:enumeration value="HillMountain" />
   <xs:enumeration value="Island" />
   <xs:enumeration value="Lake" />
   <xs:enumeration value="OtherLandFeature" />
   <xs:enumeration value="OtherWaterFeature" />
   <xs:enumeration value="ParkBeach" />
   <xs:enumeration value="PointOfInterest" />
   <xs:enumeration value="River" />
 </xs:restriction>
</xs:simpleType>
```

4.1.4.2 Curl

```
{define-enum public PlaceType
    unknown-place-type = "UnknownPlaceType",
    air-rail-station = "AirRailStation",
    bay-gulf = "BayGulf",
    cape-peninsula = "CapePeninsula",
    city-town = "CityTown",
    hill-mountain = "HillMountain",
    island = "Island",
    lake = "Lake",
    other-land-feature = "OtherLandFeature",
    other-water-feature = "OtherWaterFeature",
    park-beach = "ParkBeach",
    point-of-interest = "PointOfInterest",
    river = "River"
}
```

4.1.5 list of anonymous enumeration

4.1.5.1 XS

```
<xs:simpleType name="Themes">
 <xs:list>
   <xs:simpleType>
     <xs:restriction base="xs:string">
       <xs:enumeration value="Photo" />
       <xs:enumeration value="Topo" />
       <xs:enumeration value="Relief" />
     </xs:restriction>
   </xs:simpleType>
 </xs:list>
</xs:simpleType>
```

4.1.5.2 Curl

```
{let public constant Themes:ClassType = {Array-of String}}
```

4.1.6 simpleContent

4.1.6.1 XS

```
      <s:complexType name="category-type">
        <s:simpleContent>
          <s:extension base="s:string">
            <s:attribute name="id" type="s1:category-id-type" />
          </s:extension>
        </s:simpleContent>
      </s:complexType>
```

4.1.6.2 Curl

```
  field public content:String = ""
  field public id:Category-id-type
```

4.2 Method Signatures

4.2.1 “document” style

```
<s:element name="GetPlaceList">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="placeName" type="s:string"
/>
      <s:element minOccurs="1" maxOccurs="1" name="MaxItems" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="imagePresence"
type="s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetPlaceListResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetPlaceListResult"
                 type="tns:ArrayOfPlaceFacts" />
    </s:sequence>
  </s:complexType>
</s:element>

<wsdl:message name="GetPlaceListSoapIn">
```

```
  <wsdl:part name="parameters" element="tns:GetPlaceList" />
</wsdl:message>
<wsdl:message name="GetPlaceListSoapOut">
  <wsdl:part name="parameters" element="tns:GetPlaceListResponse" />
</wsdl:message>

<wsdl:operation name="GetPlaceListInRect">
  <wsdl:input message="tns:GetPlaceListInRectSoapIn" />
  <wsdl:output message="tns:GetPlaceListInRectSoapOut" />
</wsdl:operation>

<wsdl:operation name="GetPlaceList">
  <soap:operation style="document"
    soapAction="http://terraserver-usa.com/terraserver/GetPlaceList" />
  <wsdl:input> <soap:body use="literal" /> </wsdl:input>
  <wsdl:output> <soap:body use="literal" /> </wsdl:output>
```

4.2.2 “rpc” style

```
<message name="doSpellingSuggestion">
  <part name="key" type="xsd:string"/>
  <part name="phrase" type="xsd:string"/>
</message>

<message name="doSpellingSuggestionResponse">
  <part name="return" type="xsd:string"/>
</message>

<operation name="doSpellingSuggestion">
  <input message="typens:doSpellingSuggestion"/>
  <output message="typens:doSpellingSuggestionResponse"/>
</operation>

<operation name="doSpellingSuggestion">
  <soap:operation soapAction="urn:GoogleSearchAction"/>
  <input>
    <soap:body use="encoded" namespace="urn:GoogleSearch"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
    <soap:body use="encoded" namespace="urn:GoogleSearch"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
</operation>
```