CableLabs® Specifications

Web Services Recommended Practices

CL-SP-WSRP-I01-091023

ISSUED

Notice

This CableLabs specification is the result of a cooperative effort undertaken at the direction of Cable Television Laboratories, Inc. for the benefit of the cable industry and its customers. This document may contain references to other documents not owned or controlled by CableLabs. Use and understanding of this document may require access to such other documents. Designing, manufacturing, distributing, using, selling, or servicing products, or providing services, based on this document may require intellectual property licenses from third parties for technology referenced in this document.

Neither CableLabs nor any member company is responsible to any party for any liability of any nature whatsoever resulting from or arising out of use or reliance upon this document, or any document referenced herein. This document is furnished on an "AS IS" basis and neither CableLabs nor its members provides any representation or warranty, express or implied, regarding the accuracy, completeness, noninfringement, or fitness for a particular purpose of this document, or any document referenced herein.

Document Status Sheet

Document Control Number:	CL-SP-WSRP-I01-091023			
Document Title:	Web Services Recommended Practices			
Revision History:	I01 – Released 10/23/09			
Date:	October 23, 2009			
Status:	~~Work in Progress~~	~~Draft~~	**Issued**	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/ Member/ Vendor~~	**Public**

Key to Document Status Codes

Work in Progress	An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks

Contents

Figures

Tables

1 SCOPE

1.1 Purpose

The purpose of this document is to create an extensible set of recommended practices and conventions for web services and other protocols utilized in the Cable environment. This Web Services Recommended Practices (WSRP) specification should be complete enough to be applied to any messaging interface that will benefit by specifying the use of web services or other similar message exchange protocols.

The Web Services Recommended Practices is initially leveraged by a number of the Stewardship and Fulfillment Interfaces (SaFI) specifications, but should not be considered specific to these efforts. This WSRP specification should be complete enough to meet the SaFI needs while remaining abstract enough to be applied to any messaging interface either directly for interfaces that follow a similar model or indirectly through modular extensions and additional profiles.

This WSRP specification defines a framework from which all CableLabs defined web services will be derived. Other web services defined outside of CableLabs may also leverage the specification. Any particular web service definition will completely describe an interface in a way that ensures interoperability by drawing on guidelines, rules, and profiles defined within the specification framework.

The practices defined here should drive interoperability as the primary objective. Every implementation of an interface derived from the resulting specification is intended to seamlessly interoperate with all other derived implementations.

1.2 Introduction

By specifying common guidelines and best practices for protocols used for integration, the Cable industry and its suppliers can focus on implementing value added services. The protocols in this specification support interoperability between distributed systems. For example, the processes or machines may reside on the same network (e.g., within an MSO data center) or cross from one network to another (e.g., MSO communication and messaging with an external third party system).

Recommended practices for Web Services are specified in this document through a collection of predefined and extensible technology profiles. These profiles are intended to be collected together in many ways in order to address a range of implementation needs and constraints. The profiles are intended to provide options in choosing a messaging, data model, transport and security implementation that meet specific performance, security and/or other constraints. Additional profiles will be added as required to support future industry needs.

This specification is expected to be foundational to other application-specific API specifications. These additional specifications may be provided by CableLabs, other external organizations, and MSOs for both internal and external use.

1.3 Requirements

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word means that the item is an absolute requirement of this specification.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

[ITU-T X.509] ITU-T X.509, Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks, November 2008.

[RFC 2616] IETF RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, http://www.w3.org/Protocols/rfc2616/rfc2616.txt

[RFC 2617] IETF RFC 2617, HTTP Authentication: Basic and Digest Access Authentication

[SOAP 1.2] W3C Recommendation, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 27 April 2007, http://www.w3.org/TR/soap12-part1/

[WSDL] W3C Technical Report, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

[WS-I] The Web Services-Interoperability Organization (WS-I), Basic Security Profile Version 1.0, Final Material, 2007-03-30, http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

[XML Schema] W3C Recommendation, XML Schema, 28 October 2004, http://www.w3.org/XML/Schema

[XML Namespaces] W3C Recommendation, Namespaces in XML 1.0 (Second Edition), 16 August 2006, http://www.w3.org/TR/REC-xml-names/

2.2 Informative References

This specification does not use any informative references.

2.3 Reference Acquisition

- IETF Secretariat, c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20191-5434. Phone +1-703-620-8990; Fax +1-703-620-9071. http://www.ietf.org.
- ITU-T, International Telecommunication Union T Sector, Place des Nations, CH-1211, Geneva 20, Switzerland. Phone +41-22-730-51-11; Fax +41-22-733-7256. http://www.itu.int.
- W3C, World Wide Web Consortium, www.w3.org

3 TERMS AND DEFINITIONS

This specification does not define any new terms nor definitions.

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations:

HTTP	Hyper Text Transport Protocol
REST/ful	REST is a term coined by Roy Fielding in his Ph.D. dissertation [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm] to describe an architecture style of networked systems. REST is an acronym standing for Representational State Transfer.
SaFI	Stewardship and Fulfillment Interfaces
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
WSRP	Web Services Recommended Practices

5 WEB SERVICES OVERVIEW

A Web Service is a self-describing, platform and language independent software component or module designed to support computer-to-computer network communication. In a way, web services can be thought of as a distributed computing platform.

Web Services can take many forms but generally fall into two categories, SOAP based or RESTful. Some key characteristics of SOAP and RESTful services are identified in the table below.

Table 1 - Key Characteristics of SOAP and RESTful Services

Web Service	Messaging	Message Payload	RPC mapping	Standards	Tool Support
SOAP	Support for many messaging models including JMS, SMTP, and HTTP	XML	Modeled within HTTP header, SOAP envelope and message	Many standard bodies, standards, best practices, etc.	Many new tools both open source and commercial. Very few tools that support complete standard.
RESTful	Uses HTTP	Any HTTP message payload including XML, JSON, ATOM	Modeled within HTTP action and message payload	Leverages existing web standards. No new standards required.	No new tools required.

Initially the Web Services Recommended Practices specification addresses SOAP based web services but may subsequently be enhanced to include RESTful communications and other similar models.

At the highest level, a web service is comprised of several layers. In the Figure 1 below, a complete end-to-end web service from input/output messages through service logic is shown. It's important to note that this effort is only concerned with the outward facing service contract and messages. However, some of the recommended practices are put in place to facilitate efficient message processing by commonly used web services stacks. It is out of scope to try to address any practices around internal application business or service logic.

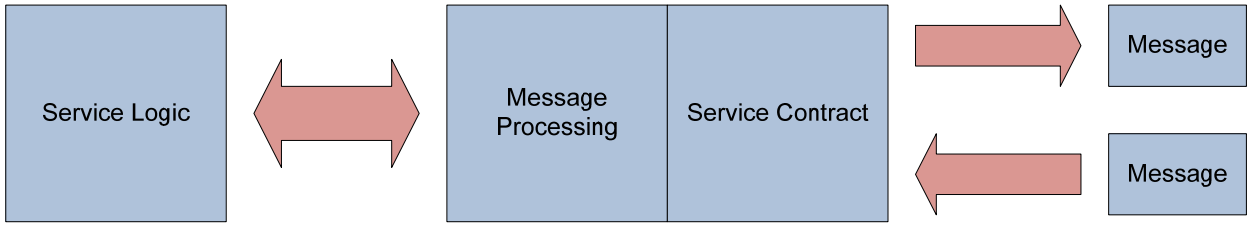

Figure 1 - Web Services Overview

6 DATA MODEL

6.1 Profile XML Schema

[XML Schema] is a W3C standard used to define the structure and constraints of an XML document. It allows a clean separation between the declaration of elements and their types. It generally follows an object-oriented model where types are similar to classes and elements are instances of a type.

It should also be noted that the lower case "XML schema" is used to refer to a particular schema document, not to the specification.

6.1.1 XML Namespaces

XML Namespaces are used to package or group XML Schema components including elements, attributes and types. By packaging XML Schemas into a namespace identified by a Uniform Resource Identifier (URI), naming collisions or ambiguities across systems can be avoided.

6.1.1.1 XML Namespace Usage

All XML schemas SHOULD be defined within a valid XML Namespace [XML Namespaces]. Each XML schema is not required to have a unique namespace so that related components that might be physically separated by files may still share a common namespace. However, components that make up an entire solution but which are functionally distinct are not required to share a namespace. All new namespaces MUST adopt a common format using all lowercase. The format to be used for CableLabs specifications MUST follow the following pattern:

`http://www.cablelabs.com/namespaces/<ProjectName>/<xsd|wsdl>/<ComponentSchemaName>/<MajorVersion>`

Namespace Variable	Definition	Example
<ProjectName>	Name or abbreviation of the Project in which this namespace is defined.	`http://www.cablelabs.com/namespaces/advads`
<xml/wsdl>	Provides additional specifics around if the namespace applies to a data model (xsd) or to a service contract (wsdl).	`http://www.cablelabs.com/namespaces/advads/xsd`
<ComponetSchemaName>	Provides the specific component name within Project.	`http://www.cablelabs.com/namespaces/advads/xsd/sms`

6.1.1.2 Namespace Hiding Versus Exposing

All but the simplest XML schemas are often made up of several individual schema definitions. A major design decision that must be made up front and which has impacts on other schema design decisions is if the various namespaces used within a schema should be exposed within XML instance documents. Exposing these namespaces allows common component names to be used and easily identified back to their source but creates a more verbose instance document. While hiding the namespace allows clean and simple instance documents, hiding the namespace is only an option for local elements as all global elements must be namespace qualified. The main way in which the namespace qualification is determined is by the value of *elementFormDefault,* as denoted below, where qualified means that local elements must be namespace qualified just like global elements and unqualified means namespace qualification is not required.

`elementFormDefault="qualified|unqualified"`

The same capability is available for attributes in the following form:

`attributeFormDefault="qualified|unqualified"`

In order to disambiguate elements while maintaining a readable format it is recommended that all but the simplest schemas expose element namespaces and hide attribute namespaces by setting the following values:

```
<xs:schema elementFormDefault="qualified"attributeFormDefault="unqualified"/>
```

In addition all schemas imported or included within a single solution SHOULD adopt the same value for both *elementFormDefault* and *attributeFormDefault* and SHOULD limit the number of globally defined elements so that *elementFormDefault* and *attributeFormDefault* can be used as an exposure switch when required.

6.1.2 Elements, Attributes and Schema Types

6.1.2.1 Naming Conventions

Each schema MUST adhere to the following naming conventions. Names SHOULD prefer readability to brevity and only well-known acronyms and abbreviations that are clearly recognizable should be used. Elements and Schema types MUST adopt a capital camel case convention (also known as "camel case, leading upper"). Schema types SHOULD generally follow element names with the additional postfix of "Type".

- Acronyms MUST be treated as a single character and, therefore, all characters in the acronyms MUST share the same case as defined by the element and attribute rules.
- Abbreviations MUST be treated as normal phases in a name and follow general element and attribute naming rules.
- All names, abbreviations, and acronyms MUST only contain alpha-numeric characters.

Attributes MUST adopt a lower camel case (also known as "camel case, leading lower") convention.

Naming Examples

Schema Component	Examples
Complex/Simple Type Names	HeadEndType, CustomerAccountType, OCAPDeviceType, AcknowledgementMsgType
Element Names	HeadEnd, CustomerAccount, OCAPDevice, AcknowledgementMsg
Attribute Names	isOCAPReady, customerId, type, ocapDevice

6.1.2.2 Elements Versus Attributes

Elements MUST be used to represent core base values or data objects. These core base value types are typically easy to distinguish within a given data model, however it becomes much more difficult to decide if a related value should be represented as a child element or an attribute of the base element. This is largely left as an engineering design decision to be made using the following guidelines.

- Elements SHOULD be considered the default model.
- If size of the instance document is of concern, prefer attributes as they produce more concise documents.
- If the value has no children and describes an attribute / characteristic of an element, then make it an attribute.
- If the value might repeat, then it should be a child element since attributes cannot repeat.
- If the related value might need to be extended in the future, make it a child element.

6.1.2.3 Global Versus Local Element and Type Definition

There are a number of competing design patterns or models which have been identified to provide guidance in developing XML Schemas. The goal is to promote reuse through global type definitions while keeping the overall schema simple and uncluttered with redundant global element definitions. For reference, a global element or type is one that is declared as a direct child of the <*schema*> tag while local elements and types are declared within another component. Complex and Simple types that could be reused MUST be defined as global types. Global elements MUST be limited to standalone root level data types and containers.

The recommended practice most closely resembles the Venetian Blind[1] model, which defines reusable components as global complex or simple types while allowing local types to be defined in the case where the local type is only valid within its local scope. The globally defined complex and simple types are then available for reuse and extension. Only top-level or root elements are defined globally, keeping the schema simple and easy to comprehend. In addition, the *elementFormDefault* can be used as a switch to hide or expose component namespaces like a Venetian Blind.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Only root element defined globally, acts a container for all other elements -->
    <xsd:element name="SetTopBox" type="SetTopBoxType"/>

    <xsd:complexType name="SetTopBoxType">
        <xsd:sequence>
            <!-- Brand is declared based on common naming conventions not referenced -->
            <xsd:element name="Brand" type="BrandType"/>
            <!-- Incorrect declaration for OCAPDevice -->
            <!-- xsd:element name="OCAPDevice" type="xsd:boolean"/ -->
            <!-- Correct declaration for OCAPDevice -->
            <!-- Assuming OCAPDevice is declared in several places now there is one Type
definition to maintain -->
            <xsd:element name="OCAPDevice" type="OCAPDeviceType"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="DVDBoxType">
        <xsd:sequence>
            <!-- Brand is declared based on common naming conventions not referenced -->
            <xsd:element name="Brand" type="BrandType"/>
        </xsd:sequence>
        <xsd:attribute name="isUpScaleCapable" type="xsd:boolean"/>
    </xsd:complexType>

    <xsd:simpleType name="BrandType">
        <xsd:restriction base="xsd:string">
            <xsd:minLength value="1"/>
            <xsd:maxLength value="20"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!-- Although simply defined now we anticipate future changes and updates so instead
of declaring the boolean type directly in all instances we define a global type so that
all instances can be centrally managed -->
    <xsd:simpleType name="OCAPDeviceType">
        <xsd:restriction base="xsd:boolean"/>
    </xsd:simpleType>

</xsd:schema>
```

6.1.3 Schema Versioning

Each schema MUST define both a major and minor version. The major version MUST be defined within the namespace identifier, while the minor version MUST be defined by the Schema version attribute. Major versions MUST start at "1" and in general may not be fully backwards compatible with previous major versions. Minor versions MUST start at "0" and MUST be backwards compatible with other minor versions within the same major version. In addition an optional data element can be defined <MinSchemaVersion> to denote compatibility within major versions. It is left to specific implementations to define the details of this data element.

[1] Other XML Schema design patterns considered were the Russian Doll (fully nested, with hidden inner types) and the Salami Slice (composite elements assembled from granular element definitions)

Versioning Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.cablelabs.com/namespaces/safi/xsd/sms/1" <!-- major version -->
    xmlns:com="http://www.cablelabs.com/namespaces/safi/xsd/com/2"
    targetNamespace="http://www.cablelabs.com/namespaces/safi/xsd/sms/1"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified" version="0"> <!-- minor version -->
</xs:schema>
```

As a general guiding principal, schema revisions SHOULD be made backwardly compatible whenever practical, while preserving the clarity and function of the message. The most common example of a backwardly compatible change is the addition of an optional element (i.e., minOccurs="0") or attribute (use="optional"). Backwardly incompatible changes, which SHOULD be avoided in most cases, include re-ordering elements in a sequence, adding required elements or attributes, and changing the type of a data element.

6.1.4 Objects

This section provides some general guidelines for using XML Schema to create object oriented data models.

6.1.4.1 Object Inheritance and Composition

The general guideline is to develop a logical structuring and encapsulation of data elements, as they exist in their business context. As a general rule, object composition SHOULD be favored over inheritance as it provides a more loosely coupled structure and better data encapsulation. The level of structural granularity is left to the designer but consideration should be given to ensure the structure provides as much contextual understanding as possible.

Example Containment Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="ContentIdentifierType">
        <xs:sequence>
            <xs:element name="ContentName" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="providerID" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="MovieContentType">
        <xs:sequence>
            <xs:element name="ContentIdentifier" type="ContentIdentifierType"/>
            <xs:element name="MovieSpecificElement" type="xs:string"/>
            <!-- more movie data elements added by containment here -->
        </xs:sequence>
        <xs:attribute name="movieSpecificAttribute" type="xs:int"/>
        <!-- more movie attributes added here -->
    </xs:complexType>

    <xs:complexType name="MusicContentType">
        <xs:sequence>
            <xs:element name="MusicSpecificElement" type="xs:string"/>
            <!-- more music data elements added by containment here -->
        </xs:sequence>
        <xs:attribute name="MusicSpecificAttribute" type="xs:int"/>
        <!-- more music attributes added here -->
    </xs:complexType>

    <xs:complexType name="ContentContainerType">
        <xs:sequence>
            <xs:element name="MovieContent" type="MovieContentType" minOccurs="1"
                maxOccurs="unbounded"/>
            <xs:element name="MusicContent" type="MusicContentType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
```

```
    </xs:complexType>
</xs:schema>
```

Example Inheritance Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="ContentBaseType" abstract="true">
        <xs:sequence>
            <xs:element name="ContentName" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="providerID" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="MovieContentType">
        <xs:complexContent>
            <xs:extension base="ContentBaseType">
                <xs:sequence>
                    <xs:element name="MovieSpecificElement" type="xs:string"/>
                    <!-- more movie elements added here -->
                </xs:sequence>
                <xs:attribute name="movieSpecificAttribute" type="xs:int"/>
                <!-- more movie attributes added here -->
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="MusicContentType">
        <xs:complexContent>
            <xs:extension base="ContentBaseType">
                <xs:sequence>
                    <xs:element name="MusicSpecificElement" type="xs:string"/>
                    <!-- more music elements added here -->
                </xs:sequence>
                <xs:attribute name="MusicSpecificAttribute" type="xs:int"/>
                <!-- more music attributes added here -->
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="ContentContainerType">
        <xs:sequence>
            <xs:element name="MovieContent" type="MovieContentType" minOccurs="1"
                maxOccurs="unbounded"/>
            <xs:element name="MusicContent" type="MusicContentType" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

6.1.4.2 Polymorphic Behavior

Polymorphic behavior SHOULD favor object inheritance designs over alternatives like substitution groups where possible. Advantages of inheritance based polymorphism:

- Provides hierarchies of types rather than of elements (Substitution groups provide hierarchies of instances/elements rather than types).
- Supported through class inheritance and casting in tool chains.
- Shorter and cleaner schema (only one global element defined, the container Items with all other elements locally defined).

Example Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="Items" type="ItemsType"/>
```

```
    <xsd:complexType name="ItemsType">
        <xsd:sequence>
            <xsd:element name="Product" type="ProductType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="ProductType">
        <xsd:sequence>
            <xsd:element name="Number" type="xsd:int"/>
            <xsd:element name="Name" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Shirt">
        <xsd:complexContent>
            <xsd:extension base="ProductType">
                <xsd:sequence>
                    <xsd:element name="Size" type="ShirtSizeType"/>
                    <xsd:element name="Color" type="ColorType"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="ColorType">
        <xsd:restriction base="xsd:string">
            <xsd:maxLength value="10"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="ShirtSizeType">
        <xsd:restriction base="xsd:int">
            <xsd:minInclusive value="0"/>
            <xsd:maxInclusive value="10"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="Hat">
        <xsd:complexContent>
            <xsd:extension base="ProductType">
                <xsd:sequence>
                    <xsd:element name="Size" type="HatSizeType"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="HatSizeType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="S"/>
            <xsd:enumeration value="M"/>
            <xsd:enumeration value="L"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

Example XML Instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Product>
        <Number>999</Number>
        <Name>Special Seasonal</Name>
    </Product>

    <Product xsi:type="Hat">
        <Number>563</Number>
        <Name>Ten-Gallon Hat</Name>
        <Size>L</Size>
    </Product>
```

```
    <Product xsi:type="Shirt">
        <Number>557</Number>
        <Name>Short-Sleeved Line Blouse</Name>
        <Size>10</Size>
        <Color>blue</Color>
    </Product>
</Items>
```

6.1.5 Extensibility

Extensibility occurs in two flavors: provisions to enable the standard to grow (and be backward compatible) and private extensions. Private extensions are always outside the scope of the specification, however, controlling their location and occurrence is within the specification's scope. Private extensions MUST always conform to the standard's extensibility guidelines.

6.1.5.1 Attribute Wildcards

Attribute wildcards allow additional attributes to appear in a complex type. Only complex type definitions include attribute extensibility (i.e., attribute wildcards). Attribute wildcarding is implemented using the anyAttribute syntax. The namespace attribute in the schema declaration SHOULD be set to "##any" attribute. This allows attributes to be included from any namespace. Any namespace is the best choice for both backward compatibility and private namespace inclusion as any namespace allows needed attributes to be added without negatively impacting the existing document.

The processContents as "lax" does not force schemas to be present for attribute validation. This allows attributes to be received and ignored rather than causing exceptions.

The attribute wildcard SHOULD typically be included on every global complex type. Thus, every global element is extensible.

Example:

```
<xsd:anyAttribute namespace=##any" processContents="lax"/>
```

6.1.5.2 Element Extensibility

If extensibility is desired or allowed, a single well known and easily recognized complex type SHOULD be globally defined and used. This complex type SHOULD be named “ExtType” and each extensibility point SHOULD be declared with an element named "Ext". The extensibility type SHOULD allow for both attribute wildcarding and element wildcarding and SHOULD allow for any number of elements to be included. Element wildcarding is enabled via the xsd:any element and SHOULD allow elements from all namespaces with a lax validation processing requirement.

The Ext element SHOULD be included in strategic locations in a complex type object hierarchy where private expansion and/or growth are anticipated. It SHOULD NOT be used in every complex type and it SHOULD typically be the last element included in a complex type declaration.

Example:

```
<xsd:element name="Ext" type="ExtType">
    <xsd:annotation>
        <xsd:documentation>Extensibility - elements from any
namespace.</xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:complexType name="ExtType">
```

```
    <xsd:sequence>
        <xsd:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##any" processContents="lax"/>
</xsd:complexType>
```

6.1.5.3 Private Extensibility

Private extensions to either attributes or elements MUST always be identified using a private namespace. The private namespace MUST never be the default target namespace. Attributes MUST be prefixed using the included private namespace prefix and MUST only appear when an element includes an extensible form of the <any> attribute expansion. Elements MUST be prefixed using the private namespace prefix and MUST only appear inside of the allowed expansion element (i.e., the Ext element).

6.1.6 Miscellaneous Guidelines

6.1.6.1 <Choice>

In general the <choice> SHOULD be avoided due to lack of full support across tool chains. In many cases the following two examples generate the exact same code even though the intention is very different.

Example:

```
<xsd:complexType name="ExampleType">
    <xsd:sequence>
        <xsd:element name="FirstElement" type="xsd:string"/>
        <xsd:choice>
            <xsd:element name="Choice1" type="Type1"/>
            <xsd:element name="Choice2" type="Type2"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ExampleType">
    <xsd:sequence>
        <xsd:element name="FirstElement" type="xsd:string"/>
        <xsd:element name="Choice1" type="Type1" minOccurs="0"/>
        <xsd:element name="Choice2" type="Type2" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

6.1.6.2 <All>

The XML Schema tag <all> is misleading in that it does not imply inclusion of all contained or child items. Items contained within an <all> allow any order and each child can occur zero or one time. In order to promote clarity of intent, the use of <all> SHOULD be avoided and replaced with the <sequence>. The <sequence> avoids most of the ambiguity by enforcing element order and ensuring explicit rules around required, optional, and repeating elements.

6.1.6.3 Required and Optional Ordering

For clarity, consistency, and ease of reading, required attributes and elements SHOULD be listed before optional attributes and elements.

7 WEB SERVICES CONTRACT

7.1 Web Services Description Language Profile

The Web Services Description Language [WSDL] provides a metadata structure for describing what a service does, how the service can be accessed and where the service can be accessed. It is broken into both abstract and concrete descriptions which are discussed in detail below.

7.1.1 WSDL Abstract Parts

The Abstract part provides a description around what a service does. In essence, this is the public interface or Application Programming Interface (API) of the service. The term abstract is somewhat misleading in that the abstract portion fully describes the interface but it is abstract in that the same interface could have multiple concrete implementations at the messaging and transport layer.

7.1.1.1 Abstract Port Type

A port type is the entry point to the web service and is sometimes referred to as the interface definition. The port type provides a container for a set of related Operation definitions. Any particular web service could define any number of port types. All port type definitions MUST use upper camel case to name the port type and be prefixed with a lower case 'pt'. The name should be descriptive of the interface or service being provided.

Example:

```
<wsdl:portType name="ptServiceName"/>
```

7.1.1.2 Abstract Operation Definition

The Operation definition provides the detailed description of the public interface. It is very similar to a class's method signature as the operation defines the available messages that are sent and received over an interface. These messages are defined within the Message Definition. All operation definitions MUST use upper camel case and be prefixed by 'op'. The name of the operation SHOULD begin with a verb and be consistent with the port type name.

Example:

```
<wsdl:operation name="opGetServiceName"/>
```

7.1.1.3 Abstract Message Definition

The Message Definition provides a high-level structure for the data that will be exchanged over a web service operation. Operations can be made up of one or more of the following message definition types. Typically if an operation is comprised of only one message type that type is of *Input*. The name of the message MUST use upper camel case and be prefixed with 'msg'. The name SHOULD be consistent with the port type name.

- Input - Messages that will be sent to the web service as "input" by the client.
- Output - Messages that the web will send to the client.
- Fault - Error messages sent by web service to the client.

Example:

```
<wsdl:message name="msgServiceName"/>
```

7.1.1.4 Abstract Type Definition

The Type Definition provides the low level data type information for input, output and fault messages used within a web service. Types MUST be defined external to the WSDL and then imported or included within the type definition.

Example:

```
<wsdl:types>
    <xs:schema>
       <xs:import namespace="http://www.cablelabs.com/namespaces/project/xsd/comp/1"
                  schemaLocation="SchemaFileName.xsd"/>
    </xs:schema>
 </wsdl:types>
```

7.1.1.5 Abstract Policy Definition

The policy definition provides a standard way to describe any behavioral constraints on both the abstract and concrete definition types. A web service can have any number of policies defined. It SHOULD include a WS-Addressing policy definition. Details of the WS-Addressing can be found within Section 11.2 of this document.

Example:

```
<wsp:Policy wsu:Id="addressing-policy" wsdl:required="true">
        <wsam:Addressing>
            <wsp:Policy>
                <wsam:AnonymousResponse/>
                <wsam:MessageId/>
            </wsp:Policy>
        </wsam:Addressing>
</wsp:Policy>
```

7.1.2 WSDL Concrete Parts

The concrete portion of the WSDL provides the details required to access an implementation of the web service as defined within the abstract definition. A single abstract definition of a service may have multiple concrete definitions specifying multiple transport and messaging protocols. The concrete part has two major areas that identify the *how* and *where* a service can be reached. How to reach the service is addressed by the Port Type Binding definition, while "the where" is addressed in the Service Binding Definition.

7.1.2.1 Concrete Port Type Binding Definition

The port type binding provides both the messaging protocol binding and transport protocol binding. The web service MUST provide a message binding for SOAP1.2 using document literal as the style and HTTP as the transport.

Example:

```
<wsdl:binding name="bdComp-SOAP12HTTP" type="ptServiceName">
    <wsp:PolicyReference URI="#policy"/>
    <wsoap12:binding style="document"
transport="http://www.w3.org/2003/05/soap/bindings/HTTP/"/>
    <wsdl:operation name="opServiceMessage">
        <wsoap12:operation
soapAction="http://www.cablelabs.com/namespaces/project/wsdl/comp/1/opServiceMessage/msgS
erviceMessage" soapActionRequired="true" wsdl:required="true"/>
        <wsdl:input>
            <wsoap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
           <wsoap12:body use="literal"/>
```

```
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

7.1.2.1.1 Operation Binding Definition

The operation binding definition, like the port type binding, is used to specify message and transport binding protocols. Alternatively if not specified, the operation binding will inherit bindings from the port type binding definition.

Example:

```
<wsdl:operation name="opServiceMessage">
  <wsoap12:operation
    soapAction=
"http://www.cablelabs.com/namespaces/project/wsdl/comp/1/opServiceMessage/msgServiceMessage" soapActionRequired="true" wsdl:required="true"/>
        …
</wsdl:operation>
```

7.1.2.1.2 Message Binding Definition

Message binding provides the capability for message or transport binding to be overloaded at each specific *input*, *output*, or *fault* message levels. If it is not present, it will inherit binding definitions from either its parent operation or port type.

Example:

```
<wsdl:input>
  <wsoap12:body use="literal"/>
</wsdl:input>
<wsdl:output>
  <wsoap12:body use="literal"/>
</wsdl:output>
```

7.1.2.1.3 Policy Binding Definition

As like with abstract parts, policies can be applied to the concrete parts of a service contract. In many cases policies applied at the abstract level can have different outcomes depending on the messaging protocol or transport protocol use. For this reason policies SHOULD be applied to specific concrete definitions in order to avoid any ambiguity.

Example:

```
<wsp:PolicyReference URI="#AbstractPolicy"/>
```

7.1.2.2 Concrete Service Binding Definition

The service binding definition provides a basic container for one or more end point definitions. The service binding definition provides no additional configuration or definition.

Example:

```
<wsdl:service name="svServiceName">
        <wsdl:port name="comp-soap12-http" binding="bdComp-SOAP12HTTP">
            <wsoap12:address location="http://cablelabs.com/ServiceName.svc/"/>
        </wsdl:port>
</wsdl:service>
```

7.1.2.2.1 End Point Definition

The endpoint definition provides a container to hold address definitions. Each port type binding definition needs at least one endpoint definition.

Example:

```
<wsdl:port name="comp-soap12-http" binding="bdComp-SOAP12HTTP">
</wsdl:port>
```

7.1.2.2.2 Address Definition

The address definition provides the physical network address identifier corresponding to the appropriate transport protocol defined within the port type binding definition either at the port type, operation or message layer. For services using HTTP transport, this is typically a Uniform Resource Locator (URL).

Example:

```
<wsoap12:address location="http://cablelabs.com/ServiceName.svc/"/>
```

7.1.2.2.3 Policy Definition

Allows policy definitions to be applied directly to the addressing layer of the concrete definition.

Example:

```
<wsp:PolicyReference URI="#policy"/>
```

8 MESSAGING

8.1 SOAP Messaging Profile

All messages to be transmitted between service endpoints MUST be a [SOAP 1.2] complaint message. A SOAP message is an ordinary XML document containing the elements described in the sections below.

8.1.1 Envelope

The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message.

Example:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
</soap:Envelope>
```

8.1.2 Header

A SOAP header is an extension mechanism that provides a way to pass information in SOAP messages that is not application payload. The SOAP Header element contains application-specific information about the SOAP message, such as WS-addressing or WS-security, etc.

Example:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Header>
        ...
    </soap:Header>

</soap:Envelope>
```

8.1.2.1 Must Understand Attribute

The SOAP mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process. If you add mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header MUST recognize the element. If the receiver does not recognize the element it will fail when processing the Header.

Example:

```
soap:mustUnderstand="0|1"
```

8.1.2.2 Actor Attribute

A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. However, not all parts of a SOAP message may be intended for the ultimate endpoint; instead, it may be intended for one or more of the endpoints on the message path. The SOAP actor attribute is used to address the Header element to a specific endpoint.

Example:

```
soap:actor="URI"
```

8.1.2.3 Encoding Style Attribute

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements. A SOAP message has no default encoding.

Example:

```
soap:encodingStyle="URI"
```

8.1.3 Body

The required SOAP Body element contains the actual SOAP message/playload intended for the ultimate endpoint of the message.

Example:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body>
        ...
    </soap:Body>
</soap:Envelope>
```

8.1.4 Fault

The optional SOAP Fault element is used to indicate error messages. If a Fault element is present, it MUST appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

Example:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body>

        <soap:Fault> ..... </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

The SOAP Fault element has the following sub elements:

Sub Element	Description
<faultcode>	A code for identifying the fault
<faultstring>	A human readable explanation of the fault
<faultfactor>	Information about who caused the fault to happen

<**detail**>	Holds application specific error information related to the Body element

The faultcode values defined below MUST be used in the faultcode element when describing faults:

Error	**Description**
<**VersionMismatch**>	Found an invalid namespace for the SOAP Envelope element
<**MustUnderstand**>	An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
<**Client**>	The message was incorrectly formed or contained incorrect information
<**Server**>	There was a problem with the server so the message could not proceed

9 TRANSPORT

9.1 HTTP Transport Profile

Hypertext Transport Protocol (HTTP) is an application layer request/response messaging protocol used in many client server applications. The World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) jointly developed the HTTP specification. It is currently maintained by the IETF as [RFC 2616]. HTTP is the protocol used by the World Wide Web and, therefore, has a number of advantages.

- Proven scalability
- High level of interoperability
- Almost universal support across platforms, tools and operating system
- Simplicity - small number of operations

9.1.1 HTTP Binding

The binding construct of the WSDL is used to define a transport protocol. There MUST be a binding construct for each web service operation that is defined. In addition, each message construct (input, output, fault) declared in the corresponding operation MUST be specified in the binding. An HTTP SOAP 1.2 binding MUST be declared.

9.1.1.1 HTTP SOAP 1.2 Binding

```
<soap12:binding style="document"
transport="http://www.w3.org/2003/05/soap/bindings/HTTP/"/>
```

9.1.1.2 SOAP Action

When using HTTP as a transport protocol, the *soapAction* MUST also be defined within the operation binding. This ensures that the *soapAction* HTTP header is added to all messages transmitting a SOAP payload. The *soapAction* provides message routing capability to the message recipient. The *soapAction* MUST be defined as follows where the *soapAction* attribute is defined by the WSDL namespace plus the operation name.

Example:

```
<soap12:operation soapAction="<namespace + operationName>" soapActionRequired="true"
wsdl:required="true"/>
```

The addition of the soapAction results in an HTTP message with the following additional header:

Example:

```
SOAPAction: http://www.cablelabs.com/namespaces/wsdl/project/component/operationName
```

10 SECURITY

Different applications and messages require different levels/forms of security. Not everything requires securing, and exchanging messages that do not require security is always safer than any form of security technology. Where security is required, the subset of the WS-I Basic Security Profile 1.0 [WS-I] MUST be used, as specified below.

10.1 Security HTTPS and VPN Profile

The message exchange MUST be secured using either transport encryption (HTTPS) or a virtual private network (ESP Security or IPSec). The recommended practice does not preclude the use of both HTTPS and VPN together if deemed necessary.

- If using HTTPS, it MUST be either SSL 3.0 or TLS 1.0, and it MAY use mutual client/server authentication.
    - Asymmetric encryption for authentication and key exchange MUST use the RSA algorithm with at least 1024-bit keys, with 2048-bit keys preferred.
    - Symmetric encryption for bulk encryption MUST use at least 128-bit keys, with 256-bit keys preferred. The algorithm SHOULD be AES.

Authentication MUST be done with at least HTTP Basic Authentication, with either HTTP Digest [RFC 2617] or X.509 [ITU-T X.509] being preferred.

10.1.1 Auditing and Logging

While not part of interoperability, and therefore not a part of this specification, it is strongly encouraged that the usage of secured resources be logged. At minimum, the following SHOULD be logged:

- User authentication failure and success
- User privilege change failure and success
- User audit log access failure and success

10.1.2 Future Considerations

Topics not currently being addressed are considerations such as data encryption and signatures multi-system authentication (single sign on, PKI registry, etc.), trust between systems, transactions, guaranteed delivery, and related concerns. Those issues are currently not seen as significant problems in the implementation environments, and therefore, are being set aside until it is determined what the needs of the entities are.

11 INTEROPERABILITY

11.1 Idempotence

As much as possible, services SHOULD be designed to be idempotent. (In other words, multiple invocations with the same data are functionally the same as one invocation.) This dramatically eases transactional, delivery and data integrity requirements because if there is ever a doubt (such as not receiving an ACK), then the message can reliably be resent without causing duplication problems.

11.2 WS-Addressing

Message exchanges complying with this specification MAY adhere to WS-Addressing standards as recommended by the W3C. WS-Addressing is therefore currently ***optional***, but if used, messages MUST conform to the W3C standard as indicated below.

When using WS-Addressing, in accordance with the specification, the message sender MUST provide:

- An endpoint reference identifying the destination, <wsa:To>.
- A message information block conveying the action to be performed, <wsa:Action>.

In addition, the message sender SHOULD provide:

- A unique message identifier, <wsa:MessageID>.

Finally, the message sender MAY provide:

- An endpoint reference identifying the message source, <wsa:From>.
- Add to the possibly null list of elements identifying related messages, <wsa:RelatesTo>.
- An endpoint reference identifying the destination of a reply message, <wsa:ReplyTo>.
- An endpoint reference identifying an alternate destination, in the event of a message fault, <wsa:FaultTo>.

As described in the WS:Addressing specification, the following semantic rules MUST be observed:

- If the message is a reply, <wsa:ReplyTo> and <wsa:RelatesTo> MUST be provided
- If the message includes a reply destination <wsa:ReplyTo> or a fault destination <wsa:FaultTo>, then .<wsa:MessageID> MUST be provided.

11.2.1 Guidelines

For near term implementations that will utilize HTTP as the transport, use of WS-Addressing is optional. However, it is described here with the understanding that in the future, compliance with the WS-Addressing standards can provide the benefits listed below.

- Ability to audit and track message traffic through the use of <wsa:MessageID>, which factors out the message id from a particular message schema.
- Facilitation of message dispatch through the use of <wsa:Action>, which provides a protocol agnostic means of conveying high-level message semantics in the absence of (1) the SOAPAction header field which is HTTP-specific and (2) service method name which is not present in the message body in the case of the document/literal style binding.
- Ensures addressing information is retained through potential intermediate endpoints which may not be using HTTP.

Example:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">

    <env:Header>
        <wsa:MessageID></wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>http://site.client.endpoint/client</wsa:Address>
        </wsa:ReplyTo>
        <wsa:To>http://site.server.endpoint/Server</wsa:To>
        <wsa:Action>http://site.server.endpoint/ActionName</wsa:Action>
    </env:Header>

    <env:Body>
        ....soap body goes here...
    </env:Body>

</env:Envelope>
```

Appendix I Use Case Example (Informative)

I.1 Service Measurement Summary (SMS)

I.1.1 SMS Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.cablelabs.com/namespaces/AdvAds/xsd/sms/0.3"
xmlns:common="http://www.cablelabs.com/namespaces/AdvAds/xsd/com/0.2"
targetNamespace="http://www.cablelabs.com/namespaces/AdvAds/xsd/sms/0.3"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:import namespace="http://www.cablelabs.com/namespaces/AdvAds/xsd/com/0.2"
schemaLocation="CLP-SP-AA-COM-D02-090505.xsd"/>
    <!-- Service Measurement Message containers -->
    <xs:element name="ServiceMeasurementMessage" type="ServiceMeasurementMessageType"/>
    <xs:element name="AcknowledgementMessage" type="common:AcknowledgementMessageType"/>
    <!-- Message Type Definitions -->
    <xs:complexType name="ServiceMeasurementType">
        <xs:sequence>
            <xs:element name="SMTimeRange" type="common:TimeRangeType"/>
            <xs:element name="GeoCode" type="common:GeographicCodeType"/>
            <xs:element name="Measurement" type="MeasurementType" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ServiceMeasurementMessageType">
        <xs:sequence>
            <xs:group ref="ServiceMeasurementMessageHeaderGroup"/>
            <xs:element name="ServiceMeasurement" type="ServiceMeasurementType"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <!-- Measurement Types Definitions -->
    <xs:complexType name="MeasurementType" abstract="true">
        <xs:sequence>
            <xs:annotation>
                <xs:documentation>Type placeholder for extension</xs:documentation>
            </xs:annotation>
        </xs:sequence>
        <xs:attribute name="process" type="MessageProcessingType" use="required"/>
        <xs:attribute name="reporting" type="MessageReportingStatusType" use="required"/>
    </xs:complexType>
    <xs:complexType name="InteractiveResponseType">
        <xs:complexContent>
            <xs:extension base="MeasurementType">
                <xs:sequence>
                    <xs:element name="InteractivePackage" type="InteractivePackageType"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="AdPlacementSummaryType">
        <xs:complexContent>
            <xs:extension base="MeasurementType">
                <xs:sequence>
                    <xs:element name="AdPlacementSummaryPackage"
type="AdPlacementSummaryPackageType" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="AdPlacementSessionDataType">
        <xs:complexContent>
            <xs:extension base="MeasurementType">
                <xs:sequence>
```

```
                    <xs:element name="AdPlacementSessionDataPackage"
type="AdPlacementSessionDataPackageType" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- General Complex Type Definitions -->
    <xs:complexType name="EventType">
        <xs:sequence>
            <xs:element name="PlacementCount" type="PlacementCountType"/>
            <xs:element name="PlacementContext" type="PlacementContentType"/>
        </xs:sequence>
        <xs:attribute name="EventID" type="common:EventIDType" use="optional"/>
    </xs:complexType>
    <xs:complexType name="ResultType">
        <xs:simpleContent>
            <xs:extension base="xs:integer">
                <xs:attribute name="Parameters" type="common:ParametersType"
use="required"/>
                <xs:attribute name="TotalInterval" type="xs:duration" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="PlacementType">
        <xs:sequence>
            <xs:element name="Ad" type="ContentPackageType"/>
            <xs:element name="PlacementTime" type="common:TimeRangeType"/>
            <xs:element name="PlacementAction" type="PlacementActionType"/>
            <xs:element name="TrackingId" type="common:TrackingType"/>
            <xs:element name="SegmentationElements" type="SegmentationElementsType"
minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="EventID" type="common:EventIDType" use="optional"/>
    </xs:complexType>
    <!-- Complex Types providing proper "packaging" to support repeating groups of
elements -->
    <xs:complexType name="ContentPackageType">
        <xs:sequence>
            <xs:element name="ProviderId" type="ProviderIdType"/>
            <xs:element name="AssetId" type="AssetIdType"/>
        </xs:sequence>
        <xs:attribute name="type" type="ContentType"/>
    </xs:complexType>
    <xs:complexType name="ContentPackageSummaryType">
        <xs:complexContent>
            <xs:extension base="ContentPackageType">
                <xs:sequence>
                    <xs:element name="Event" type="EventType" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="ContentPackageSessionDataType">
        <xs:complexContent>
            <xs:extension base="ContentPackageType">
                <xs:sequence>
                    <xs:element name="SMSessionDataTime" type="common:TimeRangeType"/>
                    <xs:element name="Placement" type="PlacementType"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="AdPlacementSummaryPackageType">
        <xs:sequence>
            <xs:element name="ContentPackageSummary" type="ContentPackageSummaryType"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="PEID" type="common:PeidType" use="required"/>
        <xs:attribute name="EPSID" type="common:EpsidType" use="required"/>
    </xs:complexType>
    <xs:complexType name="AdPlacementSessionDataPackageType">
```

```
        <xs:sequence>
            <xs:element name="ContentPackageSessionData"
type="ContentPackageSessionDataType" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="PEID" type="common:PeidType" use="required"/>
        <xs:attribute name="EPSID" type="common:EpsidType" use="required"/>
    </xs:complexType>
    <xs:complexType name="InteractivePackageType">
        <xs:sequence>
            <xs:element name="Result" type="ResultType" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="PEID" type="common:PeidType" use="required"/>
        <xs:attribute name="EPSID" type="common:EpsidType" use="required"/>
        <xs:attribute name="EventID" type="common:EventIDType" use="optional"/>
    </xs:complexType>
    <!-- Simple Type Definitions -->
    <xs:simpleType name="ProviderIdType">
        <xs:annotation>
            <xs:documentation xml:lang="en">A unique identifier for the provider of the
Asset. The
                providerID MUST be set to a registered Internet domain name restricted to
at most 20
                lower-case characters and belonging to the provider. For example a valid
providerID
                for CableLabs is "cablelabs-films.com" (19 chars).</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="20"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="AssetIdType">
        <xs:annotation>
            <xs:documentation xml:lang="en"> An identifier for the asset that is unique
within a
                provider's assetID space. The unique global identifier of an asset is the
                combination of its providerID and assetID</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:length value="20"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="PlacementCountType">
        <xs:annotation>
            <xs:documentation xml:lang="en">An integer count for all placements within
content of an
                Asset for a designated time period.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:int">
            <xs:minInclusive value="0"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="PlacementContentType">
        <xs:annotation>
            <xs:documentation>Represents content which was altered by
placement.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="vod"/>
            <xs:enumeration value="linear"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="PlacementActionType">
        <xs:annotation>
            <xs:documentation>Represents how the avail was filled. Specifically was
content replaced
                or inserted.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="replaced"/>
            <xs:enumeration value="inserted"/>
        </xs:restriction>
```

```
    </xs:simpleType>
    <xs:simpleType name="SegmentationElementsType">
        <xs:annotation>
            <xs:documentation>Addressable attributes.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="ContentType">
        <xs:annotation>
            <xs:documentation>Represents type of content being
identified.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="target"/>
            <xs:enumeration value="enhancement"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="MessageProcessingType">
        <xs:annotation>
            <xs:documentation>Defines how this message should be processed in relation to
others.
                Messages flagged with the "additive" attribute should be added to other
received
                reports for same time period and identifiers. While messages with the
"overwrite"
                attribute should replace records for the same time period and
                identifiers</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="additive"/>
            <xs:enumeration value="overwrite"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="MessageReportingStatusType">
        <xs:annotation>
            <xs:documentation>Defines if this message contains final/complete data or if
it is a partial/incremental update.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="final"/>
            <xs:enumeration value="partial"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- Group Definitions -->
    <xs:group name="ServiceMeasurementMessageHeaderGroup">
        <xs:sequence>
            <xs:element name="MinSchemaVersion" type="common:MinSchemaVersionType"/>
            <xs:element name="MessageTime" type="xs:dateTime"/>
        </xs:sequence>
    </xs:group>
</xs:schema>
```

I.1.2 SMS XML Sample

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:ServiceMeasurementMessage
xmlns:ns1="http://www.cablelabs.com/namespaces/AdvAds/xsd/com/0.2"
    xmlns:ns2="http://www.cablelabs.com/namespaces/AdvAds/xsd/sms/0.3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ns2:MinSchemaVersion>10</ns2:MinSchemaVersion>
    <ns2:MessageTime>2009-05-21T18:13:51.0Z</ns2:MessageTime>
    <ns2:ServiceMeasurement>
        <ns2:SMTimeRange starttime="2009-05-21T18:13:51.0Z" endtime="2009-05-
21T18:13:51.0Z"/>
        <ns2:GeoCode>
            <ns1:Zipcode>80020</ns1:Zipcode>
        </ns2:GeoCode>
        <ns2:Measurement process="additive" reporting="partial"
            xsi:type="ns2:InteractiveResponseType">
            <ns2:InteractivePackage PEID="PEID11111111111111111" EPSID="67" EventID="1">
                <ns2:Result Parameters="A" TotalInterval="PT3600M">13456</ns2:Result>
```

```
                <ns2:Result Parameters="B" TotalInterval="PT3600M">456</ns2:Result>
                <ns2:Result Parameters="C" TotalInterval="PT3600M">134</ns2:Result>
                <ns2:Result Parameters="TA" TotalInterval="PT3600M">234</ns2:Result>
                <ns2:Result Parameters="TO" TotalInterval="PT5670S">189</ns2:Result>
            </ns2:InteractivePackage>
            <ns2:InteractivePackage PEID="PEID111111111111111111" EPSID="67" EventID="2">
                <ns2:Result Parameters="A" TotalInterval="PT3600M">45456</ns2:Result>
                <ns2:Result Parameters="B" TotalInterval="PT3600M">1456</ns2:Result>
                <ns2:Result Parameters="C" TotalInterval="PT3600M">334</ns2:Result>
                <ns2:Result Parameters="TA" TotalInterval="PT3600M">34</ns2:Result>
                <ns2:Result Parameters="TO" TotalInterval="PT5670S">189</ns2:Result>
            </ns2:InteractivePackage>
        </ns2:Measurement>
        <ns2:Measurement xsi:type="ns2:InteractiveResponseType" process="overwrite"
            reporting="final">
            <ns2:InteractivePackage PEID="PEID111111111111111112" EPSID="68" EventID="1">
                <ns2:Result Parameters="A" TotalInterval="PT3600M">543</ns2:Result>
                <ns2:Result Parameters="B" TotalInterval="PT3600M">444</ns2:Result>
                <ns2:Result Parameters="C" TotalInterval="PT3600M">879</ns2:Result>
                <ns2:Result Parameters="TA" TotalInterval="PT3600M">23</ns2:Result>
                <ns2:Result Parameters="TO" TotalInterval="PT5670S">189</ns2:Result>
            </ns2:InteractivePackage>
            <ns2:InteractivePackage PEID="PEID111111111111111112" EPSID="68" EventID="2">
                <ns2:Result Parameters="A" TotalInterval="PT3600M">5456</ns2:Result>
                <ns2:Result Parameters="B" TotalInterval="PT3600M">345</ns2:Result>
                <ns2:Result Parameters="C" TotalInterval="PT3600M">337</ns2:Result>
                <ns2:Result Parameters="TA" TotalInterval="PT3600M">54</ns2:Result>
                <ns2:Result Parameters="TO" TotalInterval="PT5670S">189</ns2:Result>
            </ns2:InteractivePackage>
        </ns2:Measurement>
    </ns2:ServiceMeasurement>
</ns2:ServiceMeasurementMessage>
```

I.1.3 SMS WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ServiceMeasurementService"
    targetNamespace="http://www.cablelabs.com/namespaces/AdvAds/wsdl/sms/0.3"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:wsoap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sms="http://www.cablelabs.com/namespaces/AdvAds/xsd/sms/0.3"
    xmlns="http://www.cablelabs.com/namespaces/AdvAds/wsdl/sms/0.3"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
    xmlns:wsp="http://www.w3.org/ns/ws-policy">

    <wsp:Policy wsu:Id="addressing-policy" wsdl:required="true">
        <wsam:Addressing>
            <wsp:Policy>
                <wsam:AnonymousResponse/>
                <wsam:MessageId/>
            </wsp:Policy>
        </wsam:Addressing>
    </wsp:Policy>

    <wsdl:types>
        <wsdl:documentation>This import provides access to the complete SMS data and
messaging
            model</wsdl:documentation>
        <xs:schema>
            <xs:import namespace="http://www.cablelabs.com/namespaces/AdvAds/xsd/sms/0.3"
                schemaLocation="CLP-SP-AA-SMS-D03-090505.xsd"/>
        </xs:schema>
    </wsdl:types>
```

```
    <wsdl:message name="msgServiceMeasurementMessageSubmit">
        <wsdl:documentation/>
        <wsdl:part name="ServiceMeasurementMessage"
element="sms:ServiceMeasurementMessage"/>
    </wsdl:message>

    <wsdl:message name="msgServiceMeasurementMessageAck">
        <wsdl:documentation/>
        <wsdl:part name="Acknowledgement" element="sms:AcknowledgementMessage"/>
    </wsdl:message>

    <wsdl:portType name="ptServiceMeasurement">
        <wsdl:operation name="opSendServiceMeasurementMessage">
            <wsdl:input

wsaw:Action="http://www.cablelabs.com/namespaces/AdvAds/wsdl/sms/0.3/opSendServiceMeasure
mentMessage/msgServiceMeasurementMessageSubmit"
                message="msgServiceMeasurementMessageSubmit"/>
            <wsdl:output

wsaw:Action="http://www.cablelabs.com/namespaces/AdvAds/wsdl/sms/0.3/opSendServiceMeasure
mentMessage/msgServiceMeasurementMessageAck"
                message="msgServiceMeasurementMessageAck"/>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name="bdSMS-SOAP12HTTP" type="ptServiceMeasurement">
        <wsp:PolicyReference URI="#addressing-policy"/>
        <wsoap12:binding style="document"
transport="http://www.w3.org/2003/05/soap/bindings/HTTP/"/>
        <wsdl:operation name="opSendServiceMeasurementMessage">
            <wsoap12:operation

soapAction="http://www.cablelabs.com/namespaces/AdvAds/wsdl/sms/0.3/opSendServiceMeasurem
entMessage/msgServiceMeasurementMessageSubmit"
                soapActionRequired="true" wsdl:required="true"/>
            <wsdl:input>
                <wsoap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <wsoap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="svServiceMeasurementService">
        <wsdl:port name="sms-soap12-http" binding="bdSMS-SOAP12HTTP">
            <!--<wsoap12:address location="http://localhost:8080/sms/"/>-->
            <wsoap12:address
location="https://aggtest.cablelabs.com/ServiceMeasurementServiceD03.svc/"/>
            <wsam:EndpointReference>
                <!--<wsam:Address>http://localhost:8080/sms/</wsam:Address>-->

<wsam:Address>https://aggtest.cablelabs.com/ServiceMeasurementServiceD03.svc/</wsam:Addre
ss>
            </wsam:EndpointReference>
        </wsdl:port>
    </wsdl:service>

</wsdl:definitions>
```

Appendix II Acknowledgements

We wish to thank the participants contributing directly to this document:

Name	**Company**
Richard Field	Cablevision
Jim Moore	Canoe
Jason Li	Cablevision
Walt Michel	Comcast
Christopher Zarcone	Comcast
Allen Broome	CableLabs

John Kirby, CableLabs