SCTE | STANDARDS

Data Standards Subcommittee

SCTE STANDARD

SCTE 159-2 2017 (R2021)

Multimedia Application and Service Part 2: IPCablecom Multimedia Web Services

NOTICE

The Society of Cable Telecommunications Engineers (SCTE) Standards and Operational Practices (hereafter called "documents") are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interoperability, interchangeability, best practices, and the long term reliability of broadband communications facilities. These documents shall not in any way preclude any member or non-member of SCTE from manufacturing or selling products not conforming to such documents, nor shall the existence of such standards preclude their voluntary use by those other than SCTE members.

SCTE assumes no obligations or liability whatsoever to any party who may adopt the documents. Such adopting party assumes all risks associated with adoption of these documents and accepts full responsibility for any damage and/or claims arising from the adoption of such documents.

NOTE: The user's attention is called to the possibility that compliance with this document may require the use of an invention covered by patent rights. By publication of this document, no position is taken with respect to the validity of any such claim(s) or of any patent rights in connection therewith. If a patent holder has filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license, then details may be obtained from the standards developer. SCTE shall not be responsible for identifying patents for which a license may be required or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Patent holders who believe that they hold patents which are essential to the implementation of this document have been requested to provide information about those patents and any related licensing terms and conditions. Any such declarations made before or after publication of this document are available on the SCTE web site at https://scte.org.

140 Philips Road
Exton, PA 19341

Document Types and Tags

Document Type: Specification
Document Tags:

☐ Test or Measurement	☐ Checklist	☐ Facility
☒ Architecture or Framework	☐ Metric	☒ Access Network
☐ Procedure, Process or Method	☐ Cloud	☐ Customer Premises

Document Release History

Release	Date
SCTE 159-2 2010	*1/18/2010*
SCTE 159-2 2017	*2/27/2017*

Note: This document is a reaffirmation of SCTE 159-2 2017. No substantive changes have been made to this document. Information components may have been updated such as the title page, NOTICE text, headers, and footers.

Contents

Figures

Tables

1 SCOPE

NOTE: This document is identical to SCTE 159-2 2010 except for informative components which may have been updated such as the title page, NOTICE text, headers and footers. No normative changes have been made to this document.

1.1 Introduction and Purpose

This specification provides a simple, open interface between a generic Application Server (AS) and an IPCablecom Multimedia Application Manager (AM). Specifically, this specification defines a common Web Service (WS) interface to the IPCablecom Multimedia Application Manager (AM) that enables an AS to dynamically request network resources on the cable operator's access network. The primary goal of this interface is to allow AS developers to rapidly create new applications in shorter timeframes and without having a deep knowledge of the cable operator's access technology. This interface is based on the SOAP/eXtensible Markup Language (SOAP/XML).

As indicated above, the scope of this specification is limited to the interface between the AS and the AM. The following figure identifies all the interfaces that are involved in this scenario. It also provides a high-level view of the involved operations:

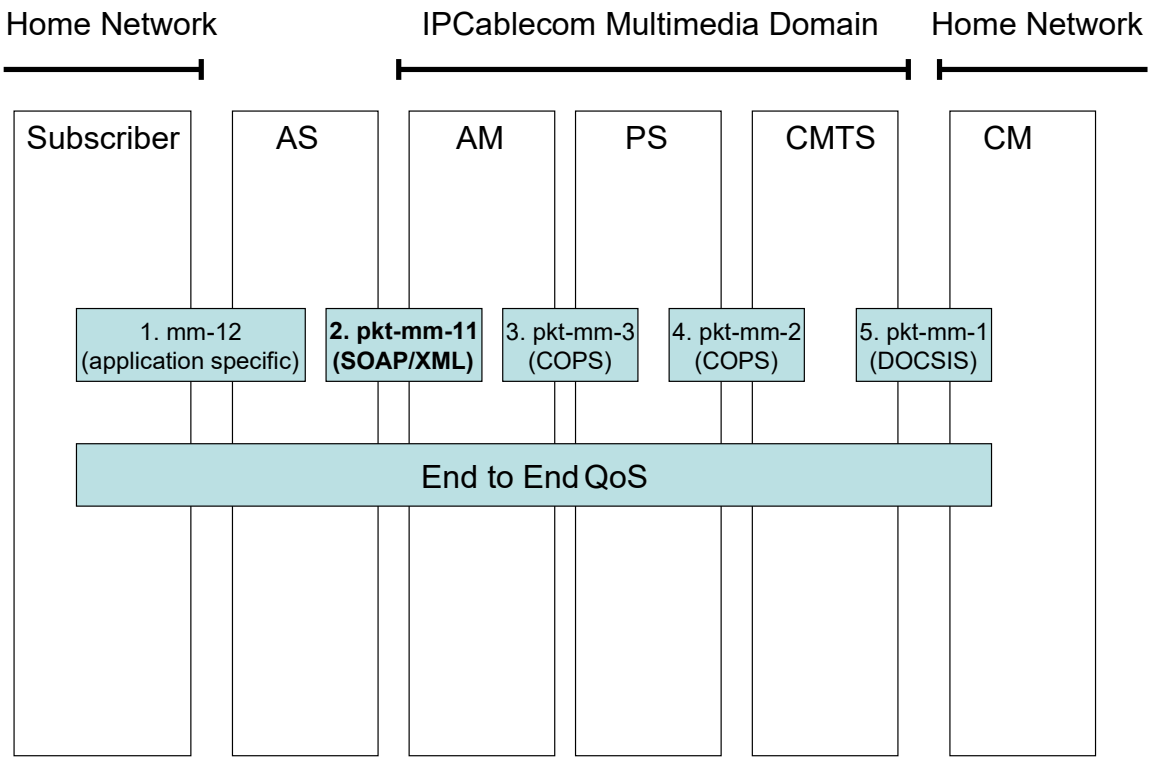

Figure 1 - End to End QoS

The operations are:

1. Subscriber requests an application from an AS, using an application specific interface (mm-12) such as HTTP, SIP or others.
2. The AS requests access network resources via the pkt-mm-11 interface.
3. The AM requests policy changes through the pkt-mm-3 interface.
4. The Policy Server sets gates on the CMTS through the pkt-mm-2 interface.
5. The CMTS installs service flows to/from the CM through the pkt-mm-1 interface.

In case of success, the AM begins with the "forwarding" to the application and the end-to-end QoS is established.

This document is intended for the following audience:

- Application developer on the AS side.
- AM vendors providing this interface.

1.2 Organization of document

Section 5 provides one use case, which shows that this interface has to be usable for applications, requiring the rendezvous protocol. It also provides a policy information model, which was used for developing this interface.

Section 6 defines the message interface including the WS requirements this interface has to fulfill.

Section 7 outlines the security requirements for the interface.

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

The following documents contain provisions, which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and users of this document are encouraged to investigate the possibility of applying the most recent editions of the documents listed below.

[HTTPS] IETF RFC 2818, HTTP Over TLS, May 2000.

[MM] SCTE 159-2 2010, IPCablecom Multimedia Specification

[SEC] SCTE 165-10 2009, IPCablecom 1.5 Part 10:Security Specification

[SOAP] SOAP Version 1.2:
W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework", 24 June 2003.
W3C Recommendation, "SOAP Version 1.2 Part 2: Adjuncts", 24 June 2003.

[WSS] OASIS standard, "Web Services Security: Username Token Profile 1,0", OASIS Standard 200401, March 2004.

2.2 Informative References

[CIM] DMTF Common Information Model (CIM) Specification, +CIM Schema: Version 2.9.1, January 2005.

[RFC2222] IETF RFC 2222, Simple Authentication and Security Layer (SASL), October 1997.

[RFC2617] IETF RFC 2617, HTTP Authentication: Basic and Digest Access Authentication, June 1999.

[UPNP] UPnP QoS Manager:1, Service Document, December 16, 2004.

[WSA] W3C Candidate Recommendation, Web Services Addressing (WS-Addressing), 17 August 2005.

[WSDL] Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl

[WSE] Appendix I of this document, Web Services Eventing (WS-Eventing), Public Draft Release, August 2004.

[XML] Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml

[XPATH] W3C Recommendation, "XML Path Language (XPath)", 16 November 1999.

2.3 Reference Acquisition

- Internet Engineering Task Force (IETF), Internet: http://www.ietf.org
- Distributed Management Task Force, Inc. (DMTF), Internet: http://www.dmtf.org
- World Wide Web Consortium, Internet: http://www.w3.org
- Organization for the Advancement of Structured Information Standards (OASIS), Internet: http://www.oasis-open.org

3 TERMS AND DEFINITIONS

This specification uses the following terms:

Application Manager	A system that interfaces to Policy Server(s) for requesting QoS-based service on behalf of an end-user or network management system.
Cable Modem Termination System	Device at a cable head-end which implements the DOCSIS® RFI MAC protocol and connects to CMs over an HFC network.
Policy Server	A system that primarily acts as an intermediary between AM(s) and CMTS(s). It applies network policies to AM requests and proxies messages between the AM and CMTS.
Quality of Service	Method used to reserve network resources and guarantee availability for applications.

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations:

AM Application Manager

AS Application Server

BEEP Blocks Extensible Exchange Protocol

CIM Common Information Model

CM DOCSIS® Cable Modem

CMTS Cable Modem Termination System

COPS Common Open Policy Service. Defined in RFC2748.

DMTF Distributed Management Task Force, Inc.

EP SIP Edge Proxy

FQDN Fully Qualified Domain Name

HTTP Hypertext Transport Protocol

IETF Internet Engineering Task Force

IP Internet Protocol

MAC Media Access Control

POP3 Post Office Protocol version 3

PS Policy Server

QoS Quality of Service

OP Originating Proxy

RFC Request for Comments

RSVP Resource Reservation Protocol. Defined in RFC2205.

SIP Session Initiation Protocol

SMTP Simple Mail Transport Protocol

TCP Transmission Control Protocol

TP Terminating Proxy

UA	User Agent
UDP	User Datagram Protocol
URI	Universal Resource Identifier
URL	Universal Resource Locator
VOD	Video On Demand
VoIP	Voice over IP
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Service Description Language
XML	eXtensible Markup Language

5 TECHNICAL OVERVIEW

This section consists of the background material used for writing this specification. Some readers may find the policy model useful to understand the detailed protocol interface specifications that follow in Section 6. This section also provides insight into the service goals and application considerations that this WS interface design addresses.

5.1 Service Goals

This document provides technical requirements for application servers in order to request DOCSIS-based network resources. It is a goal of this specification that application developers are able to rapidly develop new applications without being experts in the access technology of the cable operator's network. The simplicity of this interface is achieved by introducing an abstract service layer, which provides an access point to the underlying network policies. In the simplest case, application developers only need pass subscriber and service identifiers as arguments when requesting network resources.

5.2 Application Considerations

This WS interface specification needs to accommodate a wide range of existing applications and be generic enough for future applications. The goal of this section is to present generic application considerations and to outline how their needs are met given the presently defined IPCablecom Multimedia framework. This WS interface has considered applications with the following characteristics:

- End-client only application invocation (e.g., turbo button, media streaming, etc.) where resource requests, such as QoS, originate predominantly from the subscriber's end client.
- Rendezvous protocols where one end client may not know the IP addresses or other needed attributes of other end clients until communication with these other end clients has occurred (e.g., some "call/session control" protocols).
- Application designs where the application is stateful (e.g., most VoIP softswitch designs).
- Application designs where the application is stateless (e.g., many stateless HTTP and SIP designs).

Rendezvous protocols are important for VoIP, videoconferencing and some types of VoD applications. Stateless applications are also foundational in the design of some highly scalable Internet system architectures.

The characteristics of rendezvous and non-rendezvous applications and stateful and stateless applications are important design considerations for this WS interface definition. Additionally the semantics of the services defined in the WS interface has also accommodated these characteristics.

Because rendezvous applications are such an important application class, it is worthwhile to explicitly outline their generic behavior in order that the service semantics (described later in this document) will become apparent. The next section outlines characteristics of these rendezvous applications. This next section also highlights the so-called two-phase "reserve then commit" resource model used in IPCablecom Multimedia. This two-phase model also impacts the semantics behind the "reserve" and "commit" service definitions in the web services interface that is defined later in this document.

5.2.1 Protocol Design Considerations

A stateless Service Control Domain (SCD) rendezvous application in an IPCablecom Multimedia environment is used here as an example. This example is modeled after HTTP or SIP operation and is greatly simplified; only particular IPCablecom Multimedia and proxy elements are shown to illustrate the Application Server (AS) to Applications Manager (AM) interface. In this example, the AS is a stateless HTTP or SIP proxy element. Commentary for the slightly less complex stateful SCD case is deferred to the end of this section.

For purposes of clarity, both ends of the communication, the origination end and the termination end are depicted here in Figure 2. However, only the origination end is considered here for the purposes of the web services interface explanation. The AS in this case is the Originating Proxy (OP). Figure 2 depicts the relevant application signaling and web services operations. Only Messages 2, 7 and 10 between the AS and AM are part of this web services interface specification.

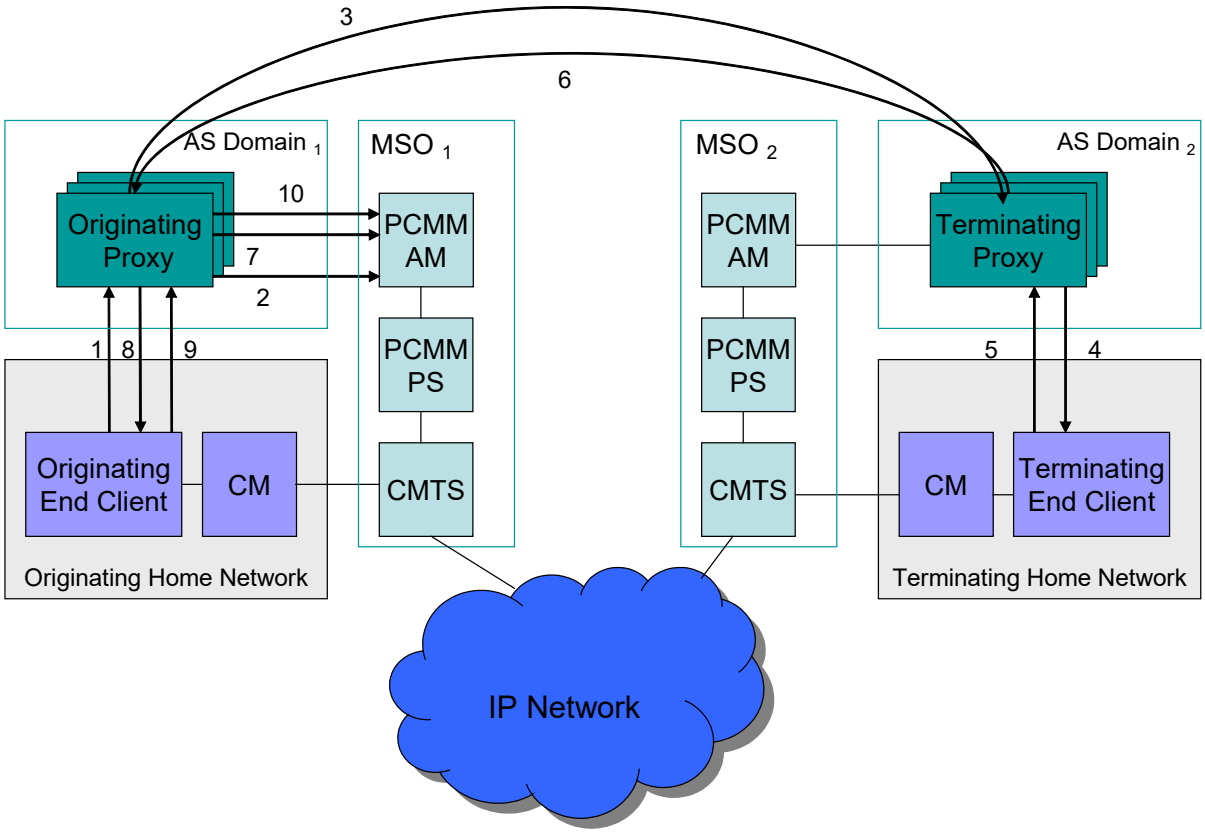

Figure 2 - Stateless SCD Rendezvous Protocol Example

Message 1: The Originating End Client initiates communications to some end client, depicted here as the Terminating End Client. Depending on the application involved, this step usually involves the resolution of a URI or URL (e.g., a SIP telephone URL) to one or more potential terminating end clients. Indeed, resolution to multiple terminating end clients may occur at a later stage of signaling (e.g., after Message 3).

Message 2: At this step, the OP may not know many of the Terminating End Client attributes (such as their IP addresses or ports) as the terminating end client(s) have not yet been contacted. However, the OP (acting here as an AS) sends a ReserveResourcesRequest message via the WS interface to the AM for the purpose of reserving resources, such as QoS, for the originating end of the communication. At the AM, this session is identified via the BaseID component of the ContextID (6.2.1.2), which has been provided to it by the OP.

Message 3: The OP forwards the message and it eventually reaches the Terminating Proxy (TP). At this step, it is instructive to note that, due to forking and other features of these protocols, there may be multiple

TPs.

Message 4: The TP sends the message to the Terminating End Client. At this step it is instructive to note that there may be multiple Terminating End Clients served by a singular TP. The key point here is to note that a singular "Message 1" may result in a multiplicity of "Message 4" messages to multiple Terminating End Clients.

Message 5: The Terminating End Client replies to "Message 4". The reply contains the IP address of the Terminating End Client and optionally other session or other end client attributes.

Message 6: The TP forwards the response to the OP. The response includes the IP address of the Terminating End Client.

Message 7: For each message received from a different Terminating End Client, the OP sends an updated ReserveResources message via the WS interface to the AM with additional, previously undetermined, information such as the IP Address of the Terminating End Client. Each of these ReserveResource messages is associated with a different Terminating End Client. At the AM each of these reservations has a unique ContextID that is: 1) associated with the same "Message 2" reservation by virtue of each having the same "BaseID" portion of the ContextID, and 2) differentiated from each other by virtue of differing (and unique) "ModifierID" portions of the ContextID.

Message 8: The OP forwards the response from each Terminating End Client (Message 5 and 6) to the Originating End Client.

Message 9: The Originating End Client selects one (or more) of the responses from the Terminating End Clients to establish communication with the Terminating End Client and sends one (or more) messages to the OP.

Message 10: The OP sends a CommitResource message to the AM for each of the Terminating End Client sessions it wishes to communicate with. This action thereby commits the resources that were previously reserved in the ReserveResource of Message 2 and re-reserved in the updated ReserveResource Message 7.

Message 11: (Not shown) The OP forwards the (Message 9) message to the Terminating End Clients to complete the rendezvous signaling.

Note that Message 9 and 10 would also be sent to each Terminating End Clients that the Originating End Client did not wish to establish communicate with. In this case, the action in Message 10 would be to send a ReleaseResource message via the WS interface to the AM indicating that it did not require the reserved resource for communication with these Terminating End Clients. Reserved resources are also released if there is not a timely CommitResource or ReserveResource sent to the AM (a ReserveResource message can be used to refresh an existing reservation).

This example illustrates that the WS interface must allow the ContextIDs (with BaseIDs and optional ModifierIDs) created by the AS and provided to the AM to evolve over time as the communication is fully established for the stateless Service Control Domain example provided here.

A slightly simpler model can be used for stateful Service Control Domains; in this case unique ContextIDs may optionally be created by the AM and provided to the AS for every candidate Terminating End Client.

Therefore, the WS interface allows the AS or AM to provide the ContextID, dependent on whether stateless or stateful SCD designs are employed by the application.

5.3 Architecture and Policy Model

ASes use this interface to request resources in the cable operator network on behalf of its clients. Looking at the architecture of the IPCablecom Multimedia environment and understanding the policy model, used by the IPCablecom Multimedia components, makes it easier to understand the WS operations and their arguments. In the following subsections, a policy model, which is inspired by the Common Information Model (CIM) Schema 2.9.1 [CIM], is discussed.

This document does not mandate the implementation of the CIM Policy model by the AM. The following model was used to develop this interface. It is up to AM vendors to decide exactly how the policies and services are modeled internally.

As illustrated in Figure 3, the AM retrieves the service and policy data from a repository. The interfaces to this repository are not within the scope of this specification.

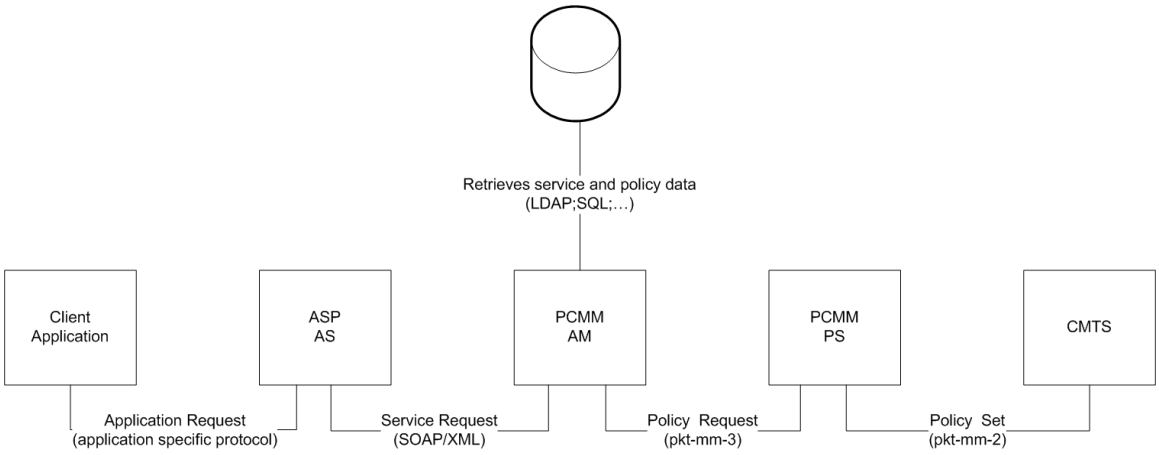

Figure 3 - IPCablecom Multimedia Architecture

This information is provisioned by the cable operator to the data repository, which could be a relational database, an LDAP server or any other storage medium. The AM converts this data to its internal information model, which includes all the relationships between the subscribers, offered services and defined policies.

These relationships are shown in Figure 4.

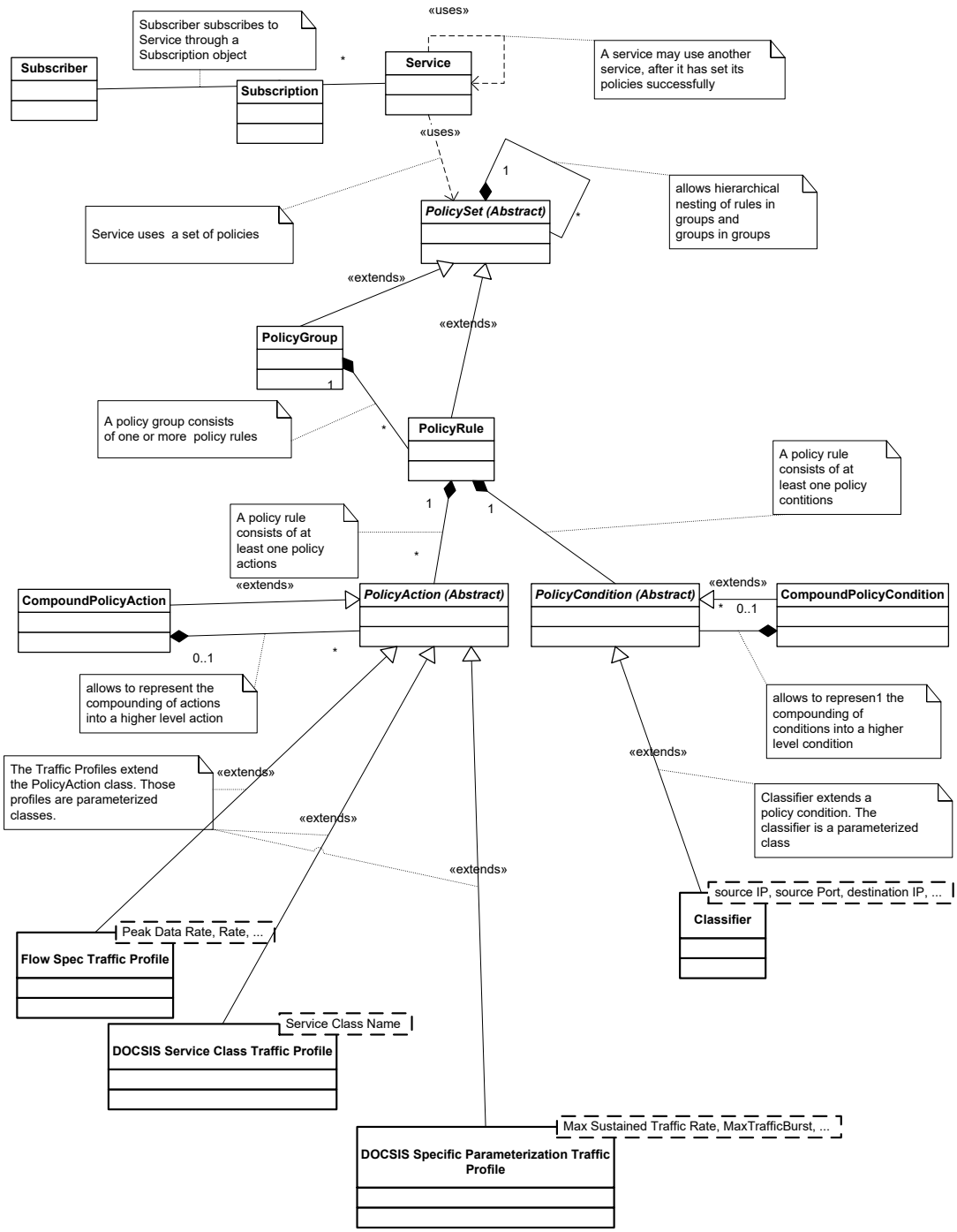

Figure 4 - Policy Model

After the AM retrieved all the data and converted the data internally into the object information model, depicted in Figure 4, it is able to send policy requests down to the PS. Those policy requests may be triggered by service

requests via the SOAP/XML interface from an AS. If the service or resource request only contains the identifier of the service and the identification of the end-customer, the AM sends the policy model with its default values (defined by the cable operator) down to the PS. The service requester has also the opportunity to "overwrite" some of those default values, by including the new values in the service request. The argument list is discussed later in Section 6.

The objects and relationships of the information model, depicted in Figure 4 are discussed in the remainder of this Section.

5.3.1 Subscriber

A subscriber is an entity that can subscribe to services. In this context, "subscribe to a service" means that a subscriber agreed to the terms of the service.

5.3.2 Subscription

A subscription is an object that represents an enrollment to a given service for a specific subscriber. In other words, a subscriber enrolls or subscribes to a service through a subscription object (e.g., the subscription object creates a relationship between the subscriber and a service).

5.3.3 Service

A service is an object that contains the information to represent the functionality offered to a subscriber. Service objects "use" a set of policies, which implement this offered functionality in the cable operator network.

5.3.4 PolicySet

The PolicySet object provides an abstraction for a set of rules to manage and control access to network resources. PolicyGroup and PolicyRule objects are derived from the PolicySet object.

5.3.5 PolicyGroup

A PolicyGroup object represents an aggregation of policy sets that have the same decision strategy. The decision strategy defines the evaluation method used for policies contained in the policy group (e.g., "first matching"). A PolicyGroup object can be seen as a container for a set of related policy rules and policy groups.

5.3.6 PolicyRule

This object is used to organize the conditions and actions that comprise the policy rule. PolicyRule objects consist of conditions that are used to match traffic and actions that specify the action to be taken if the traffic matches.

5.3.7 PolicyCondition

This abstract class is used to identify conditions. It is also the superior class of the Classifier object.

5.3.8 Classifier

Classifiers are used for defining 'classifiers' that a packet or packet flow must contain; for example, network protocol, source and destination addresses. This is a "dynamic" or "parameterized" object, meaning the values of those fields provided in a ResourceRequest can be dynamically substituted in place of those provided in the service definition.

5.3.9 CompoundPolicyCondition

CompoundPolicyCondition is used to represent compound conditions formed by aggregating simpler policy conditions.

5.3.10 PolicyAction

Policy actions are used to define the 'action' that the network takes on packets that match conditions. The PolicyAction object serves a superior class for FlowSpecTrafficProfile, DOCSISServiceClassTrafficProfile and DOCSISSpecificParameterizationTrafficProfile.

5.3.11 FlowSpecTrafficProfile

The FlowSpecTrafficProfile class defines the Traffic Profile of the IP flow associated with an IPCablecom Multimedia Gate through an RSVP-like parameterization scheme. This is a "dynamic" or "parameterized" class, meaning the values of attributes on an instance can be dynamically substituted. The list of attributes can be found in [MM].

5.3.12 DOCSISServiceClassTrafficProfile

The DOCSISServiceClassNameTrafficProfile contains the DOCSIS Service Class Name, indicating the DOCSIS Service Class to be used to describe the QoS attributes. The DOCSIS Service Class Name can be substituted.

5.3.13 DOCSISSpecificParameterizationTrafficProfile

This class allows defining a Traffic Profile, using explicit DOCSIS parameters of the DOCSIS flow. The exact list of attributes is specified in [MM].

5.3.14 CompoundPolicyAction

A CompoundPolicyAction is used to represent compound conditions formed by aggregating simpler policy actions.

6 WS INTERFACE DESCRIPTION

This section describes the interface between a generic Application Server (AS) and the Application Manager.

The section explains in detail the syntax of all the arguments or elements passed via this interface, and the semantics of the function calls that can be performed using this WS interface.

6.1 WS Profile

This section defines the WS standards this interface MUST comply with.

6.1.1 Encoding

This interface is presented as a sequence of XML-serialized messages. Application Managers and Application Servers MUST support SOAP document-style encoding, which is also known as "document/literal" or "doc/lit".

6.1.2 SOAP standard

Application Managers and Application Servers MUST conform to the SOAP 1.2 standard [SOAP]. This allows for extensibility and fault-transmission. This implies that the SOAP 1.2 processing model MUST be followed, including the "soap:role" (with default value "ultimateRecipient") and "soap:mustUnderstand" attributes in the SOAP header. The "soap:Fault" semantics and format MUST be followed. This is described in more detail in Section 6.3.6.

6.1.3 Transport layer

Application Managers MUST support [HTTPS] as the transport protocol and conform to the security requirements in Section 7.2 of this document.

Application Servers MUST support [HTTPS] as the transport protocol and conform to the security requirements in Section 7.2 of this document.

The SOAP 1.2 format is transport-agnostic; hence, this interface MAY in the future support transport protocols (e.g., BEEP, SMTP/POP3) other than HTTPS. However, specifications relating to alternate transports are not covered by this document.

6.1.4 WS-Addressing

Application Managers and Application Servers MAY support the use of WS-Addressing [WSA] with this SOAP interface.

Application Servers SHOULD support the use of WS-Addressing [WSA] with this SOAP interface if they use transport protocols other than HTTPS for SOAP transport. However, specifications relating to alternate transports are not covered by this document.

Application Managers SHOULD support the use of WS-Addressing [WSA] with this SOAP interface if they support transport protocols other than HTTPS for SOAP transport. However, specifications relating to alternate transports are not covered by this document.

If an Application Server supports WS-Addressing, it MUST tag each message with an identifier that is unique within that client - the MessageId (as defined in [WSA]) with a soap:mustUnderstand attribute set to value '0'.

If an Application Manager supports WS-Addressing, it MUST set the WS-Addressing "RelatesTo" header element of response messages to reflect the MessageId, if present, of the request message. If an AM does not support WS-Addressing, it must treat any WS-Addressing related headers like MessageId in accordance with the soap:mustUnderstand attribute specified for those headers in conformance with the SOAP [SOAP] specification.

6.1.5 WS-Security: Username Token Profile

Application Managers MUST require the specification of the Username from the WS-Security: Username Token Profile [WSS].

Application Servers MUST provide the Username in the SOAP header for every request as per [WSS]. The Username will be a value assigned by the cable operator and identifying an AS.

Application Managers MAY provide an authentication and authorization mechanism based on [WSS] as described in Section 7.3 of this specification.

6.1.6 Support for Event subscription

This specification provides a way for the AS to subscribe to events indicating autonomous changes to sessions by the network. This allows the AS to be notified of changes to requested services (such as when a CMTS preempts a gate or a gate timer expires or the usage limit on a gate is exceeded).

The AM MAY support this eventing feature.

The WS-Eventing Specification [WSE] MUST be used as the basis for subscribing to events, in case the AM provides this feature. The WS-Eventing specification is described in Appendix I.

An AM supporting this feature MUST provide SOAP endpoints supporting the WS-Eventing WSDL port types EventSource and SubscriptionManager.

An AS that uses this Event subscription feature MUST support the use of WS-Addressing.

An AM that uses this Event subscription feature MUST support the use of WS-Addressing.

6.1.6.1 IPCablecom Multimedia Event Filter Dialect

The WS-Eventing specification allows the AS to specify filters to allow an AS to only receive events matching certain criteria. The WS-Eventing allows multiple filter dialects to be used for filtering events. It also defines one such dialect to be the XPath specification [XPATH]. However, arbitrary XPath filters may be expensive for the AM to implement. Therefore this specification defines a filter dialect (called the IPCablecom Multimedia Event Filter dialect) identified by the namespace URL http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS for use with this specification.

Filter criteria using this dialect are specified using the query-context-req request message as defined in Annex B. See Section 6.3.5.1 QueryContexts message for semantics of the matching operation for this filter. The following example depicts an event subscription with an event filter using this specification.

```
<Subscribe xmlns="http://schemas.xmlsoap.org/ws/2004/08/eventing"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
   <Delivery Mode="http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push">
      <NotifyTo>
         <wsa:Address>http://www.example.org/PCMM/EventListener</wsa:Address>
      </NotifyTo>
   </Delivery>
   <Expires>P0Y0M0DT0H30M0S</Expires>
   <Filter Dialect=" http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS ">
      <pcmm: QueryContextsReq xmlns:pcmm=" http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS " >
```

```
            <pcmm:ServiceName>Turbo</pcmm: ServiceName>
            <pcmm:SubscriberID>…</pcmm: SubscriberID>
            <pcmm:ContextID><…./></pcmm:ContextID>
          </pcmm: QueryContextsReq>
        </Filter>
      </Subscribe>
```

WS-Eventing EventSource implementations for an AM MUST support the IPCablecom Multimedia Event Filter dialect.

6.2 Services

This WS provides an interface for requesting (reserving and/or committing) and releasing network resources on the cable operator network. It also provides the ability to retrieve all the applications or services a given AS can request. Furthermore, actual sessions can be queried, based on a given end customer, given application or given session identifier. ASes also have the opportunity to register for events, and the AM delivers updates according to this registry. Those services require a set of elements (or arguments), which are discussed in more detail.

Table 1 - Web Service Operations

Operation Name	Style	Messages	
ReserveResources	Document	**Input**	ReserveResourcesRequest
		Output	ReserveResourcesResponse
CommitResources	Document	**Input**	CommitResourcesRequest
		Output	CommitResourcesResponse
ReleaseResources	Document	**Input**	ReleaseResourcesRequest
		Output	ReleaseResourcesResponse
QueryAvailableServices	Document	**Input**	QueryAvailableServicesRequest
		Output	QueryAvailableServicesResponse
QueryContexts	Document	**Input**	QueryContextsRequest
		Output	QueryContextsResponse
ResourceStateNotification	Document	**Output**	ResourceStateNotification

6.2.1 Element list

This section describes the elements, which are either passed in the request and response messages of this WS interface, defined in Section 6.3, or used for the authentication and authorization layer of individual SOAP (request) messages, described in 7.3.

6.2.1.1 Classifier

The Classifier element allows the AS to identify the traffic flow for which it is requesting resources. The AS MUST format the element as defined in Annex A.

The Classifier identifies a single flow in a given direction. The direction indicated in the Classifier provided by the client is considered the upstream or inbound direction. The inverse of this flow defined by replacing the source IP address and port, with the destination IP address and port and vice versa, identifies the downstream or outbound flow.

If the Classifier elements were defined during the service creation, the AS MAY omit the Classifier element. If the AS chooses to include the Classifier element in a resource request message, the AS MUST include only a single

Classifier element with at least one sub element populated. The AS requires knowledge of whether or not a service definition requires Classifiers to be specified, and which particular elements are necessary.

The AS MUST NOT mix unicast and multicast destination IP addresses for a single flow.

For Multicast flows (where the Destination IP Address in the classifier identifies a specific Multicast IP address) the AS encodes the classifier as follows:

- The Source IP address & the Mask/Prefix Length combination MUST resolve to a specific valid Unicast IP address value i.e., not resolve into a range of addresses. If the Source IP address & the Mask/Prefix Length combination resolve to the All-Zeros IP Address, it indicates that all Source IP addresses are permitted, i.e., Any Source Multicast (ASM).
- The Destination IP address & the Mask/Prefix Length combination MUST resolve to a single specific Multicast address
- The Source and Destination Port Number fields in the classifier are ignored by the CMTS for a Multicast flow.

If the AS includes a Classifier element with the initial resource request, it MUST include a Classifier element with all subsequent resource requests. If the AS does not include a Classifier element in the initial resource request, it MAY include the Classifier element in a subsequent request.

The AM MUST use the following precedence rules when determining which value to use for each Classifier element:

- If the element is included in the Classifier, use the value defined in the Classifier element.
- Else if the element was defined during the service and policy definition, use the value defined in the service and policy definition.
- Else use the 'assumed value' defined for each element below.

There are two formats defined for Classifiers – an IPv4 format and an IPv6 format corresponding to the use of IPv4 or IPv6 addresses in the classifier.

6.2.1.1.1 IPv4 Classifier

The IPv4 Classifier is denoted by the element named ‘Classifier’ in the XML Schema. The contents of the IPv4 classifier element are described as follows:

- The protocol element identifies the type of protocol. Integer values range from 0 to 257. The value 0 indicates no protocol match. The value 256 indicates any protocol. The value 257 matches packets with protocol type TCP or UDP. If this element is omitted, its assumed value should be 0.
- The string value of the sourceIpAddress element identifies the IP address (as seen at the CMTS) of the originator of the IP flow. A value of "0.0.0.0" indicates any IP address. If this element is omitted, its assumed value should be "0.0.0.0", and the assumed value of sourceIpMask should be "0.0.0.0".
- The string element sourceIpMask is used for identifying the mask value for the source IP address in order to classify multiple IP addresses for a single Classifier object. A value of "0.0.0.0" indicates any subnet. This element MUST NOT be included if sourceIpAddress is not included. If sourceIpAddress is omitted, or the sourceIpAddress has a value of "0.0.0.0", then the assumed value of sourceIpMask should be "0.0.0.0". Otherwise, if this element is omitted and sourceIpAddress is not omitted, then its assumed value should be "255.255.255.255".
- The element sourcePortStart identifies the start or low-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted its assumed value should be 0.

- The element sourcePortEnd identifies the end or high-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted and the sourcePortStart is included, then its assumed value should be the same as the value of sourcePortStart. If this element is omitted and the sourcePortStart is omitted, then its assumed value should be 65535.
- The string value of the destinationIpAddress element specifies the matching value of the IP address of the termination point of the IP flow. A value of "0.0.0.0" indicates any IP address. If this element is omitted, its assumed value should be "0.0.0.0", and the assumed value of destinationIpMask should be "0.0.0.0".
- The string element destinationIpMask is used for identifying a subnet for the destination IP address in order to classify multiple IP addresses for a single Classifier object. A value of "0.0.0.0" indicates any subnet. This element MUST NOT be included if destinationIpAddress is not included If destinationIpAddress is omitted, or the destinationIpAddress has a value of "0.0.0.0", then the assumed value of destinationIpMask should be "0.0.0.0". Otherwise, if this element is omitted, and destinationIpAddress is not omitted, then its assumed value should be "255.255.255.255".
- The element destinationPortStart identifies the low-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted, its assumed value should be 0.
- The element destinationPortEnd identifies the high-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted, and the sourcePortStart is included, then its assumed value should be the same as the value of sourcePortStart. If this element is omitted and the sourcePortStart is omitted, then its assumed value should be 65535.

6.2.1.1.2 Ipv6 Classifier

The IPv6 Classifier is denoted by the element named ‘IPv6Classifier’ in the XML Schema. The contents of the IPv6 classifier element are described as follows:

- The nextHeader element identifies the type of transport protocol the classifier should match. It performs the same function as the protocol field in IPv4 classifier. The protocol element identifies the type of protocol. Integer values range from 0 to 257. The value 0 indicates no protocol match. The value 256 indicates any protocol. The value 257 matches packets with protocol type TCP or UDP. If this element is omitted, its assumed value should be 0.
- The string value of the sourceIpAddress element identifies the IP address (as seen at the CMTS) of the originator of the IP flow. An IP address value consisting of 16 zero octets (i.e., ‘::’ or equivalent string representation) indicates the classifier matches any source IP address.
- The numeric element sourcePrefixLen is used for identifying the number of significant prefix bits of the sourceIpAddress to classify multiple IP addresses for a single Classifier object. A value of 0 indicates that the classifier matches any address. This element MUST NOT be included if sourceIpAddress is not included. If sourceIpAddress has a value of "::" or equivalent, then the assumed value of sourcePrefixLen is 0. Otherwise, if this element is omitted and sourceIpAddress is has a value other than ‘::’ or equivalent, then sourcePrefixLen is assumed to have a value of 128.
- The element sourcePortStart identifies the start or low-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted its assumed value should be 0.
- The element sourcePortEnd identifies the end or high-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted and the sourcePortStart is included, then its assumed value should be the same as the value of sourcePortStart. If this element is omitted and the sourcePortStart is omitted, then its assumed value should be 65535.
- The string value of the destinationIpAddress element identifies the IP address (as seen at the CMTS) of the terminator of the IP flow. An IP address value consisting of 16 zero octets (i.e., ‘::’ or equivalent string representation) indicates that the classifier matches any destination IP address.
- The numeric element destinationPrefixLen is used for identifying the number of significant prefix bits of the destinationIpAddress to classify multiple IP addresses for a single Classifier object. A value of 0 indicates that the classifier matches any address. This element MUST NOT be included if destinationIpAddress is not

included. If destinationIpAddress has a value of "::" or equivalent, then the assumed value of destinationPrefixLen is 0. Otherwise, if this element is omitted and destinationIpAddress has a value other than '::' or equivalent, then destinationPrefixLen is assumed to have a value of 128.

- The element destinationPortStart identifies the low-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted its assumed value should be 0.
- The element destinationPortEnd identifies the high-end of the UDP or TCP port ranges for matching the IP flow. If this element is omitted and the sourcePortStart is included, then its assumed value should be the same as the value of sourcePortStart. If this element is omitted and the sourcePortStart is omitted, then its assumed value should be 65535.
- The elements trafficClassMask, trafficClassLow, traffixClassHigh identify the values of the traffic class field in the IP header that match this classifier. If trafficClassMask is omitted, it is assumed to have a value of 0. If trafficClassMask is 0 or unspecified, trafficClassHigh and trafficClassLow MUST NOT be specified. If trafficClassLow is omitted, and trafficClassMask has a non-zero value, trafficClassLow is assumed to have a value of 0. If trafficClassHigh is omitted, and trafficClassMask has a non-zero value, trafficClassHigh is assumed to have a value of 255.
- The element flowLabel identifies the IPv6 flow label value that the classifier matches. If this element is omitted, comparison of the IPv6 flow label is irrelevant for this entry.

6.2.1.2 ContextID

The ContextID element is the identifier that associates the resources of a given session. The ContextID element consists of a baseId element and optional idExtension elements. The same ContextID or an extended version of the same ContextID (extended via new idExtensions) MUST be used in subsequent resource request messages when resources are modified after the initial request.

The AS MAY assign a ContextID for a context it creates by providing a value for the ContextID element for the network resources request. If the AS chooses to provide the ContextID, it MUST do so such that the ContextID is guaranteed to be unique at a given time across all Application Servers that use the same AS Username. If the AS chooses to provide the ContextID, it MUST format the element as defined in Annex A. The AS MAY choose to create optional idExtension elements as described in 6.2.1.2.1. When using optional idExtension elements, the ContextID so created must also be unique; uniqueness of ContextIDs with optional idExtensions is defined in the following section. If the AS chooses to provide the ContextID, it SHOULD ensure ContextIDs created are not reused for a reasonable period of time (e.g., weeks) after the context corresponding to them no longer exist.

If the AS provided a ContextID value to the AM, the AM MUST use this ContextID and MUST NOT overwrite or replace a ContextID value submitted to it by the AS.

The AM MUST assign a ContextID for the requested context if the AS did not provide a ContextID value in the network resource request. If the AM assigns the ContextID, it MUST format the element as defined in Annex A. The AM MUST do so such that the ContextID so created is guaranteed to be unique at a given time across all Application Servers that use a particular AS Username value. If the AM assigns the ContextID, it MAY choose to create optional idExtension attributes. When using optional idExtension attributes, the ContextID so created must also be unique; uniqueness of ContextIDs with optional idExtensions is defined in the following section. If the AM chooses to provide the ContextID, it SHOULD ensure that ContextID values created are not reused for a reasonable period of time (e.g., weeks) after the context corresponding to them no longer exist.

The AM SHOULD treat the combination of the AS Username (provided in SOAP header as per [WSS]) and ContextID service attributes as the unique identifier for a context.

6.2.1.2.1 Context ID: Uniqueness and Use in Identifying sessions

This section defines how to identify sessions by the ContextID element and defines uniqueness for the ContextID. The capability of the AS to provide the ContextID is useful for the stateless rendezvous protocol case of Section

5.2.1. For both the stateless and stateful rendezvous protocol cases, a multiplicity of potential contexts may be created that are associated with the same originating request due to forking of the original request. Such a case is shown below in Figure 5.

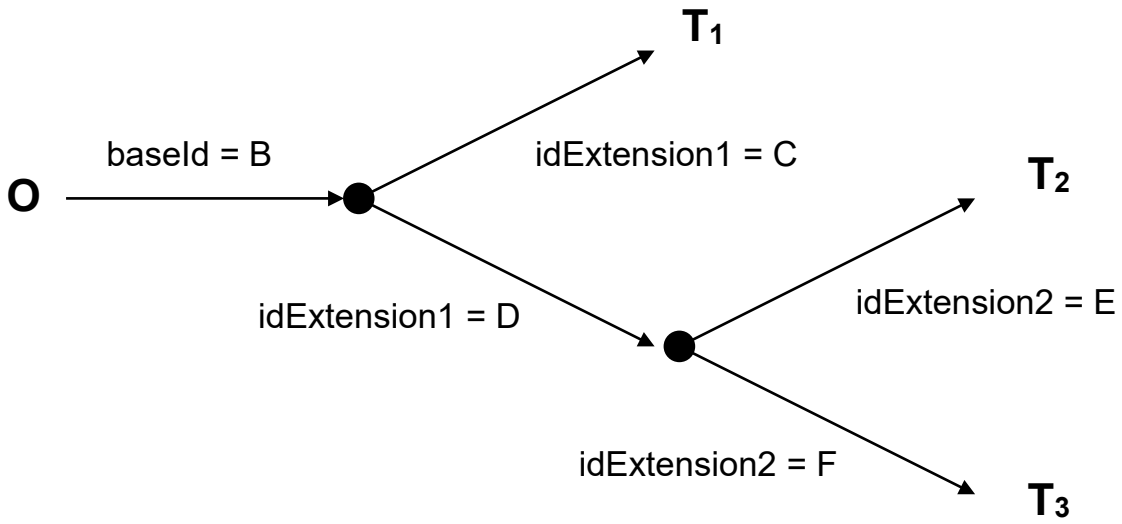

Figure 5 - ContextID creation

In this figure, a session request from an originating end client ("O") was forked to three potential terminating clients ("T_1", "T_2" and "T_3"). The idExtension elements of the ContextID are used to differentiate the legs for such cases. For example the session leg to T_2 is {B,D,E}. In this notation the elements in the brackets denote the ContextID and is of the form ContextID = {baseId, idExtension1, idEntension2, …}. Likewise, the session leg to T_3 is {B,D,F} and the session leg to T_1 is simply {B,C}. Note that although some legs have a different number of idExtension elements, all three ContextID attributes share the same baseId element, and are unique. The AM can therefore associate all the different ContextIDs for these three session legs with the same outbound originating request by the baseId element.

For the case where the AM received the ContextID from the AS, the AM MUST perform accounting of the individual session legs and perform required operations for each leg individually (e.g., reservation timeouts) and associate forked requests that have originated from the same originating end client request (by virtue of such requests having the same baseId).

The AS MUST perform this accounting dynamically for those legs added by virtue of additional optional-id attributes being added to previously defined ContextIDs in subsequent web services operations.

The ContextID is considered unique when the combination of the base-id attribute together with the same number of optional id-extension attributes is unique. Whenever a new branch is formed, a new level of id-extension attribute MUST be created so that the new ContextID has the same BaseID and idExtension attributes of the previous level. Whenever a new leaf is formed from an existing branch, a new level of idExtension MUST NOT be created and a new, unique idExtension at the existing level MUST be created.

6.2.1.2.2 ContextID: Wildcarding operations

As all sessions are uniquely associated via the ContextID and the WS messages described later will refer to the ContextID, it is useful to define wildcarding operation on the ContextID here. Using the notation where the ContextID = {baseId, idExtension1, idEntension2, …} the wildcarding operation is highly dependent on where the wildcard is placed. Deleting resources for contentId = {B,D,*} for the case above will delete resources for T_2 and T_3, while leaving resources for T_1 unaffected. Deleting resources for {B,*} will delete all resources for this context. Deleting resources for {*} will delete all resources for a given SubscriberID (and given AS Username and serviceName) and should normally only be invoked by administrative operations.

Therefore, the AS SHOULD normally use wildcarding of the form {B,*} to affect a web services operation on all sessions with baseId = B (i.e., the * wildcard matches on one, many or null). Note that ContextID = {B} does not match the ContextID for any of the call legs in the example above, as the ContextID is defined as including all optional idExtensions defined up to the present. In the above example, ContextID = {B} may have at one time existed prior to the ContextID being extended by optional idExtensions.

Considering optional idExtension elements, wildcarding of the form {B,*} is highly RECOMMENDED for all AS operations on all sessions with baseId = {B} (i.e., even for applications that do not use optional-id elements).

6.2.1.3 ContextInfo

An array of ContextInfo elements is returned in the QueryContexts response message (described in 6.3.5.1). The ContextInfo element contains the ContextID element, described in 6.2.1.2, and the element ContextStatus, which identifies the status of the associated IPCablecom Multimedia gate for the session. A context can be in a reserved or committed state.

The AM MUST format the ContextInfo element as defined in Annex A.

6.2.1.4 QoSChangeEvent

The QoSChangeEvent element describes the change in state for a flow of a session. QoSChangeEvent is present in notification messages sent to ASes which have subscribed to the state of the resources. The AM MUST format the QoSChangeEvent element as defined in Annex A.

The QoSChangeEvent element contains a direction element indicating whether this change is for the upstream / downstream flows and consists of two sub elements.

The QoSChangeType element indicates the type of change being signaled. This attribute is an enumeration, the AM MUST indicate one of the following states:

1 = Idle/Closed

2 = Authorized

3 = Reserved

4 = Committed

5 = Committed-Recovery

Reason is a 2-byte unsigned integer field, which MUST indicate one of the following reasons for this update:

1 = Close initiated by CMTS due to reservation reassignment

2 = Close initiated by CMTS due to lack of DOCSIS MAC-layer responses

3 = Close initiated by CMTS due to timer T1 expiration

4 = Close initiated by CMTS due to timer T2 expiration

5 = Inactivity timer expired due to Service Flow inactivity (timer T3 expiration)

6 = Close initiated by CMTS due to lack of Reservation Maintenance

7 = Gate state unchanged, but volume limit reached

8 = Close initiated by CMTS due to timer T4 expiration

9 = Gate state unchanged, but timer T2 expiration caused reservation reduction

10 = Gate state unchanged, but time limit reached

11 = Close initiated by Policy Server or CMTS, volume limit reached

12 = Close initiated by Policy Server or CMTS, time limit reached

13 = Close initiated by CMTS, other

14 = Gate state unchanged, but SharedResourceID updated

15 = Close initiated by CMTS due to loss of shared resource

65535 = Other

The optional message element contains a human readable description of the cause.

6.2.1.5 ServiceName

The ServiceName element identifies the service for which network resources are requested. The string value for service names is determined by mutual agreement between the AM and the AS. In addition, all available service identifiers can be retrieved by the QueryAvailableServices operation, described in 6.3.2.1.

The AS MUST format the ServiceName element as defined in Annex A.

6.2.1.6 SubscriberID

The AM requires the SubscriberID element for identifying the CMTS, where the gate has to be created for a given request. The SubscriberID identifies the subscriber for which gates are to be created. It can be an IPv4 or IPv6 address, a fully qualified domain name (FQDN), or a MAC address. The SubscriberID MUST be formatted as specified in Annex A.

The AM MUST return an error-code, identifying an "IllegalArgumentException" if the AS does not follow the syntax defined in Annex A. The error procedures are described in 6.3.8.

6.2.1.7 Timeout

The Timeout element is optional. The AS MAY use the Timeout element to specify a Timeout for resources. This Timeout value means different things when used within a ReserveResourcesRequest verse when it is used within a CommitResourcesRequest.

When a Timeout value is submitted with a ReserveResourcesRequest, the Timeout value specified determines the duration for which the resources will remain in a 'reserved' state. If no further ReserveResourcesRequest or CommitResourcesRequest are issued for the given session within the duration of this Timeout value, the session will be deleted. A Timeout value of '0' disables the Timeout. In this case, a ReleaseResources request MUST be issued in order to "un-reserve" the network resources. In this case the timer is analogous to the T2 timer as defined in [MM].

When a Timeout value is submitted with a CommitResourcesRequest, the Timeout value specified determines the duration for which the ReserveResourcesResponse message arguments resources will remain in a 'committed' state with no traffic detected for the session. In other words, it determines the duration for which a committed session should be allowed to stay idle. If a session remains idle for a duration which exceeds the value specified in the CommitResource request, the session will be deleted. In this case the timer is equivalent to the T3 timer as defined in [MM].

If no Timeout element is specified by the AS, the Timeout if any is determined by the AM.

The AM MUST use the Timeout value as the T2 or T3 timer as appropriate when creating or modifying the associated IPCablecom Multimedia gate(s).

Notifications of timer expiries will be sent to the AM via GateReportState COPS messages. If the AM supports generating events and an AS has subscribed to the state of its resources, it MUST generate notifications for these events as described in Section 6.3.7.

The AS MUST format the Timeout element as defined in Annex A.

6.2.1.8 TimeUsageLimit

The TimeUsageLimit element is optional. The AS MAY use to the TimeUsageLimit attribute to specify a limit on the lifetime of a context. The TimeUsageLimit is specified in seconds. An AS MAY specify multiple TimeUsageLimit elements to request different limits in the upstream and downstream directions.

If no TimeUsageLimit element is specified by the AS, the TimeUsageLimit if any is determined by the AM.

The AM MUST use the TimeUsageLimit value as the time usage limit when creating or modifying the associated IPCablecom Multimedia gate(s).

If a time limit was previously specified but not specified in the current operation, the AM MUST continue to use the previously provided value.

The TimeUsageLimit element follows the operation defined in Section 6.5.7 of [MM].

The AS MUST format the TimeUsageLimit element as defined in Annex A.

6.2.1.9 TrafficProfile

The TrafficProfile element allows the AS to provide information on the bandwidth and QoS characteristics desired for a request.

The AS MUST format the TrafficProfile element as defined in Annex A.

The TrafficProfile element contains a "direction" element indicating whether the requested traffic profile is for upstream or downstream directions or both. An AS MAY include multiple traffic profile elements to request distinct traffic requirements in the upstream and downstream directions.

If no traffic profile is specified, the AM MUST use the default TrafficProfile, which was set up during service definition.

The TrafficProfile MUST be expressed in one of four ways, by specifying a simple requested bandwidth parameter, by specifying a FlowSpec, by specifying a TrafficClass, or by specifying upstream drop. Further, each of these requests can be accompanied with the priority parameter.

6.2.1.9.1 TrafficProfile Bandwidth

The Bandwidth element allows the AS to indicate a desired bandwidth in bytes per second. The exact mapping of this value to peak / reserved bandwidth is determined by the AM.

6.2.1.9.2 TrafficProfile FlowSpec

The FlowSpec element is an alternative means for the AS to indicate its desired bandwidth characteristics. The following diagram depicts the structure of the FlowSpec element and it MUST be formatted as defined in Annex A.

The FlowSpec element consists of seven elements, which map to specific DOCSIS parameters. The mapping is described in more detail in Section 9.2 in [MM].

- The bucketRate element expects values in bytes/second.
- The bucketDepth element value is specified in bytes.
- The value of the maximumDatagramSize element is specified in bytes.
- The minimumPolicedUnit element value is specified in bytes.
- The peakRate element expects values in bytes/second.
- The value of the reservedRate element is specified in bytes/second.
- The slackTerm element is specified in microseconds.

6.2.1.9.3 TrafficProfile TrafficClass

The TrafficClass element is an enumerated variable that can be assigned to one of the following list of values and is taken from [UPNP]:

- Network Control
- Streaming Control
- Voice
- AV
- Data
- Audio
- Images
- Gaming
- Other
- Background

6.2.1.9.4 TrafficProfile Priority

The Priority element is a range of integers [0-7] used to signal the layer 2 traffic priority. This element allows the AS to indicate a desired layer 2 priority for the service request in addition to its parameterized request. No assumptions are made within this specification on the relative priority of the values in the priority range as each layer 2 technology tends to define its own meaning of the priority values. The use of this value requires knowledge of the underlying layer 2 technology being used for the service. The range of 0-7 was chosen to best align with the

majority of layer 2 priority schemes currently supported. When no priority value is given, the default value shall be 0. The AM MAY choose to ignore this parameter based on local policy.

6.2.1.9.5 Traffic Profile Upstream Drop

The element TrafficProfileUpstreamDrop specifies that the AM must create a gate using traffic profile type Upstream Drop to cause traffic in the upstream direction that matches the corresponding classifier to be dropped at the CM. This element has no contents.

A Traffic profile specified as Upstream Drop MUST have an upstream direction; downstream and bidirectional traffic flows MUST NOT specify upstream drop as the traffic profile. Also, a traffic profile of type Upstream Drop MUST NOT have a priority parameter.

6.2.1.10 UserName

The UserName element is optional. The AS MAY use it to specify the subscriber who requested the service. This element, if present, MUST uniquely identify the name of the subscriber for which the Service request is associated.

The AS MUST format the UserName element as defined in Annex A.

6.2.1.11 VolumeUsageLimit

The VolumeUsageLimit element is optional. The AS MAY use the VolumeUsageLimit element to specify a limit on the total volume of traffic allowed for a session before it should expire. The volume usage limit is specified in bytes. This element also contains a 'direction' element indicating whether the limit is for the upstream or downstream directions. An AS MAY specify multiple volume usage limit elements to request different limits in the upstream and downstream directions.

If no VolumeUsageLimit element is specified by the AS, the volume usage limit if any is determined by the AM.

The AM MUST use the VolumeUsageLimit value as the time volume limit when creating or modifying the associated IPCablecom Multimedia gate(s).

If a VolumeUsageLimit was previously specified but not specified in the current operation, the AM MUST continue to use the previously provided value.

The VolumeUsageLimit element follows the operation defined in Section 6.5.7 of [MM].

The AS MUST format the VolumeUsageLimit element as defined in Annex A.

6.3 WS Protocol Operation

This section describes the methods associated with this WS interface, which can be used by the AS to dynamically activate services on behalf of their clients. The associated WSDL definition which specifies the message encoding format and the transport binding can be found in Annex B. Application Servers and Application Managers MUST conform to the WSDL definition as presented in Annex B.

6.3.1 Operation Sequence for Requesting a Service

The following sections provide requirements for requesting services. In particular how an AS reserves and commits resources and how the AM processes such requests.

6.3.1.1 ReserveResources Messages

If an application requires reserving network resources before attempting to commit them, the AS MUST invoke the ReserveResourcesRequest message of this interface. The AS MUST include the elements SubscriberID and serviceName. All other attributes are optional and MAY be present.

If the AM receives a ReserveResourcesRequest with a ContextID, it MUST check for existence of the ContextID. If the ContextID already exists, the AM MUST treat this as an update to an existing resource reservation. In such a case the AM MUST verify that the provided ServiceName and SubscriberID match that which was previously provided. If either is not found to match, the AM MUST generate a SOAP Fault and send it to the requesting AS. If both attributes are found to match, the AM MUST perform the appropriate IPCablecom Multimedia GateSet operations with Authorized and Reserved traffic profiles envelopes, any optional attributes, and the previously assigned GateID(s) to update the QoS reservation. The AM MUST return an acknowledgement message to the requesting AS upon the successful update of the reservation. In the event the reservation of network resources was not successful, the AM MUST generate a SOAP Fault and send it to the requesting AS.

If the AM does not find a matching ContextID, it MUST treat the ReserveResourcesRequest as a new request. In such a case, the AM MUST perform the appropriate IPCablecom Multimedia GateSet operations with Authorized and Reserved traffic profile envelopes and any optional attributes to reserve the requested QoS resources in the IPCablecom Multimedia network. The AM MUST return an acknowledgement message to the requesting AS upon the successful reservation. In the case where the ReserveRequest did not contain a ContextID, the AM MUST locally generate a ContextID and return it in the success response. Otherwise, the received ContextID MUST be returned. In the event the reservation of network resources was not successful, the AM MUST generate a SOAP Fault and send it to the requesting AS.

Table 2 - ReserveResourcesRequest message arguments

Argument	Required or Optional	Comments
SubscriberID	R	Identifies the Subscriber who requests QoS. See 6.2.1.6 for formatting information.
ServiceName	R	Identifies the offered service, which requires QoS to be set up. See 6.2.1.5 for formatting information.
ContextID	O	Some use-cases (such as VoIP) require that the AS set the session identifier. Section 6.2.1.2 describes the ContextID element.
Classifier	O	This optional element is provided to identify the traffic flow for which the application is requesting the network resources. See Section 6.2.1.1. Only a single Classifier element can be included.
TrafficProfile	O	Specifies the network resources. More information is available in 6.2.1.9.
VolumeUsageLimit	O	Specifies a volume-based-usage-limit. More information is available in 6.2.1.11.
TimeUsageLimit	O	Specifies a time-based-usage-limit. More information is available in 6.2.1.8.
Timeout	O	Specifies duration, for how long the requested service remains in a "reserve" state. More information is available in 6.2.1.7.
UserName	O	Specifies the name of a subscriber. More information is available in 6.2.1.10.

Table 3 - ReserveResourcesResponse message arguments

Argument	Required or Optional	Comments
ContextID	R	Identifies the session identifier, see Section 6.2.1.2.

The reservation of network resources is not a required operation in order to commit resources (or activate the specified service).

6.3.1.2 CommitResources Messages

In case an application requires network resources, the AS MUST invoke the CommitResourcesRequest message of this interface. The AS MUST pass values for the SubscriberID and ServiceName arguments. The AS MAY invoke the CommitResourcesRequest operation directly without having previously reserved resources using the ReserveResourcesRequest. If the AS previously used the ReserveResourcesRequest to reserve resources for this session, it MUST provide a ServiceName value identical to the one used in the ReserveRequest, and a ContextID value matching the original used for reserving resources (either AS or AM assigned).

If the ContextID element is present in the CommitResourcesRequest message, it may be an attempt to modify an existing service or an attempt to commit resources without having reserved them prior. In order to make this determination, the AM MUST compare the provided ContextID with its known set of ContextIDs using the rules defined in Section 6.2.1.2.

If a match is found, the AM MUST consider this an update to an existing service request, which may be in either the Reserved or Committed state. Regardless of the state, the AM MUST verify that the provided ServiceName and SubscriberID match that which was previously provided. If either is not found to match, the AM MUST generate a SOAP Fault and send it to the requesting AS. If both attributes are found to match, the AM MUST perform the appropriate IPCablecom Multimedia GateSet operations with Authorized, Reserved and Commit traffic profile envelopes, any optional attributes, and the previously assigned GateID(s) to update the QoS reservation. The AM MUST return an acknowledgement message to the requesting AS upon the successful update of the reservation. In the event the Commit of network resources was not successful, the AM MUST generate a SOAP Fault and send it to the requesting AS.

If the AM does not find a matching ContextID, it MUST treat the CommitResourcesRequest operation as a new request. In such a case, the AM MUST perform the appropriate IPCablecom Multimedia GateSet operations with Authorized, Reserved and Commit traffic profile envelopes, and any optional attributes to Commit the requested QoS resources in the IPCablecom Multimedia network. The AM MUST return a CommitResourcesResponse message to the requesting AS upon the successful reservation. In the case where the CommitResourcesRequest did not contain a ContextID, the AM MUST locally generate a ContextID and return it in the success response. Otherwise, the received ContextID MUST be returned. In the event the commitment of network resources was not successful, the AM MUST generate a SOAP Fault and send it to the requesting AS.

Table 4 - CommitResourcesRequest message arguments

Argument	Required or Optional	Comments
SubscriberID	R	Identifies the Subscriber who requests QoS. See 6.2.1.6 for formatting information.
ServiceName	R	Identifies the offered service, which requires QoS to be set up. See 6.2.1.5 for formatting information.
ContextID	O	Some use-cases (such as VoIP) require that the AS set the session identifier. Section 6.2.1.2 describes the ContextID argument.
Classifier	O	This optional argument is provided to identify the traffic flow for which the application is requesting the network resources. See Section 6.2.1.1. Only a single Classifier object can be submitted.
TrafficProfile	O	Specifies the network resources. More information is available in 6.2.1.9.
VolumeUsageLimit	O	Specifies a volume-based-usage-limit. More information is available in 6.2.1.11.
TimeUsageLimit	O	Specifies a time-based-usage-limit. More information is available in 6.2.1.8.
Timeout	O	Specifies duration, for how long the service can be idle, before it is deleted. More information is available in 6.2.1.7.
UserName	O	Specifies the name of a subscriber. It is not required that this name is unique. More information is available in 6.2.1.10.

Table 5 - CommitResourcesResponse message arguments

Argument	Required or Optional	Comments
ContextID	R	Identifies the session identifier, see Section 6.2.1.2.

6.3.2 Procedures for Querying a Service Status

If the AS wants to know which services it can request, the AS MAY invoke the QueryAvailableServices request message.

6.3.2.1 QueryAvailableServices Messages

The AS MUST send a QueryAvailableServices request message to the AM in order to retrieve all available services, it can request. The AM MUST return a response message, which includes a list of the available service names to the requesting AS if the query is successful. If no services are available, the AM must send a response message containing an empty list of available services. In the event the query of the available services was not successful, the AM MUST generate a SOAP Fault and send it to the requesting AS.

The QueryAvailableServices request message has no arguments.

Table 6 - QueryAvailableServices response message arguments

Argument	Required or Optional	Comments
ServiceName	R	List of ServiceName elements. See 6.2.1.5 for formatting information.

6.3.3 Procedures for Modifying a Service Request

This interface does not provide explicit messages for modifying services in the Reserved and Committed states. If applications require modifying services, the AS MUST use the already introduced methods ReserveResources and CommitResources. The AS identifies the services to be modified with the submitted ServiceName and ContextID arguments. The ServiceName value MUST be identical to the ServiceName value for the original session. The ContextID value MUST match the ContextID for the original session as per the rules for matching sessions in Section 6.2.1.2. The AM MUST overwrite the current argument values with the submitted arguments. Further, this interface does not support the ability to move from a Committed state back to a Reserved state. If the AM receives a ReserveResource request for an existing ContextID with it in the Committed state, it MUST return a SOAP fault indicating an invalid state transition. See Section 6.3.8 for error procedures.

6.3.3.1 Modifying Time and Volume Limits

To modify the Usage Limits associated with an existing service, an AS MAY send a CommitResourcesRequest message with the ContextID and ServiceName of the service to be modified. Once the rules defined in Section 6.3.3 have been satisfied, if the TimeUsageLimit or VolumeUsageLimit of the service is present, then the existing limits associated with this/these parameter(s) MUST be replaced with the new parameter(s). However, the absence of these parameters from a CommitResourcesRequest message indicates that the existing Time or VolumeUsageLimit(s) of the service still apply. If these parameters are not present in a CommitResourcesRequest message, the existing limits MUST be maintained and their associated counters/timers MUST continue from their current value without resetting.

6.3.4 Procedure for Deleting a Service Request

When an application no longer needs the allocated resources (Reserved or Committed), the AS MUST invoke the ReleaseResources request message in order to tell the AM that the resources are no longer needed. In such cases, the AM MUST perform the appropriate IPCablecom Multimedia GateDelete operation with the previously assigned GateID(s) to release the QoS reservation.

When releasing resources, the AS MUST submit either the SubscriberID or ContextID values. The AS MUST provide a ContextID value to release resources for a single session. If the ContextID is omitted, the AM MUST clear all resource reservations for the given SubscriberID.

The AS MAY also include the ServiceName with the SubscriberID. This has the effect of clearing all resources allocated for a given subscriber and a specific Service. Upon receipt of a ReleaseResources request message with a SubscriberID and ServiceName, the AS MUST delete all resources associated with the indicated subscriber and the indicated ServiceName.

6.3.4.1 ReleaseResources Messages

The AS sends a ReleaseResources request message to the AM for removing all resources associated with a ContextID. Upon receipt of a ReleaseResources message, the AM MUST perform the appropriate IPCablecom Multimedia GateDelete operation(s) to delete any resources in the IPCablecom Multimedia network. Upon successful release of all resources, the AM MUST return a ReleaseRsourceRsp message to the requesting AS. In the event that the deletion of the service is not successful, the AM MUST generate a SOAP Fault and send it to the requesting AS.

The ReleaseResources response message has no arguments

Table 7 - ReleaseResources request message arguments

Argument	Required or Optional	Comments
SubscriberID	R	Identifies the Subscriber who requests QoS. See 6.2.1.5 for formatting information.
ServiceName	O	Identifies the offered service, which requires QoS to be set up.
ContextID	O	Identifies the session to be released. Section 6.2.1.2 describes the ContextID argument.
UserName	O	Specifies the name of a subscriber. It is not required that this name is unique. More information is available in 6.2.1.10.

6.3.5 Procedures for Synchronization

The AS MAY query the AM for committed or reserved resources. The query can be based on a given subscriber and/or a given service and/or a given session. The AS MUST provide at least one value of those arguments in a QueryContexts request message.

6.3.5.1 QueryContexts Messages

The AM MUST return a response message to the requesting AS upon the successful context retrieval. The response MUST include a list of ContextInfo elements, containing all ContextIDs and their associated state, which match the values of the QueryContexts request message parameters below. If no contexts were retrieved, the AM MUST return an empty ContextInfo element. In the event the query was not successful, the AM MUST generate a SOAP Fault and send it to the requesting AS.

Table 8 - QueryContexts request message arguments

Argument	Required or Optional	Comments
SubscriberID	O	Identifies the subscriber for which the AS wants to find context and/or service information for. See 6.2.1.5 for formatting information.
ServiceName	O	Identifies the service that the AS wants to find context and/or service information for.
ContextID	O	Identifies the specific context to query status for. Section 6.2.1.2 describes the ContextID argument.

Note: The AS MUST submit at least one of these arguments. The AS MAY submit more than one.

Table 9 - QueryContexts response message arguments

Argument	Required or Optional	Comments
ContextInfo	R	Contains the information of a single session. See 6.2.1.5 for formatting information.

6.3.6 Procedures for Subscribing to Events

To subscribe to events the AS MUST use the SubscribeOp operation on the EventSource SOAP service provided by the AM. See Appendix I for more information on subscribing to events and managing subscriptions.

6.3.6.1 Filtering Events

The AS MAY specify filters to restrict the events it wants to receive. ASes that need to filter events for specific ContextID; SubscriberID and / or ServiceName SHOULD use the IPCablecom Multimedia Event Filter dialect to specify filters.

The WS-Eventing EventSource for an AM MUST apply filters in the IPCablecom Multimedia Event dialect following the rules for finding sessions using the QueryContexts request message previously defined in Section 6.3.5 and its subsections.

When a WS-Eventing EventSource service accepts subscriptions for events, the AS Username of the AS is implicitly a filter criterion; i.e., regardless of whether or not the AS specifies a filter, it can only receive events relating to contexts created with the same AS Username. Therefore, an AS that specifies no filters receives all events relating to sessions it created, but not for other sessions.

An AS MAY subscribe for events as many times as it needs.

6.3.7 Generating and Notifying Events

The AM MUST generate ResourceStatusNotification events when it knows about a status change for a context, e.g., when it receives a GateReportState message from a PS. The AM MUST notify all ASes with subscriptions to events subject to the filtering criteria as specified in Section 6.3.6.1, in case the AM supports the eventing feature.

6.3.7.1 ResourceStatusNotification

This message is used by the AM to indicate to an interested AS about status changes to contexts.

The ResourceStatusNotification message contains a ContextID element identifying the affected context, a notification-cause identifying the cause for this notification and optional pcmm-context-status-change element indicating changes for each component of the resource reservation. The notification-cause element is an enumeration with three defined values:

- 'Deleted' indicating complete deletion of the context,
- 'PartiallyDeleted' indicating that a component of the resource reservation has been deleted,
- 'Informational' indicating an informational message.

As noted earlier, the status-change element does allow extensions and that should allow notifications for other IPCablecom Multimedia Gate changes to be included.

6.3.8 Error Procedures

This section describes how exceptional conditions are communicated back to the AS via the SOAP interface for AS initiated messages. The AM MUST use the SOAP fault message to carry error information back to the calling client (AS), hence the presence of the element soapenv:Fault inside the body indicates an error has occurred. The following elements are available within the SOAP fault:

code This element MUST be present in a SOAP fault and is used to determine the basic error. This interface MUST use the SOAP faultcode "Receiver", in case the error occurs within the IPCablecom Multimedia framework, for example if the PS is not able to set the required gate.

The AM MUST set the SOAP faultcode to "Sender", if some information, passed from the AS is not correct. One example for this case would be that the AS passes a SubscriberID string, which was not created according to the syntax, outlined in 6.2.1.6.

A hierarchical namespace of values can be obtained by separating the fault values with a dot (.) character, e.g., Receiver. UnknownGate, Receiver.OtherError or Sender.UnauthorizesAS.
The error-codes and error-types are specified in Table 10 and Table 11.

The AM MUST use the element "VersionMismatch" if the SOAP envelop of the request contains an invalid namespace.

The AM MUST use the element "mustUnderstand", if the SOAP header of the request included a header element with mustUnderstand="1", but was not understood.

reason This element is used to pass a descriptive human-readable error to the caller. The AM MUST use this element and MUST pass the message of the original exception.

node This element provides information about who caused the fault and usually contains the URI of the perpetrator. The AM MAY use this element.

role This element identifies the role the node was operating in at the time of fault. The AM MAY use this element.

detail This AM MUST use this element to pass a more detailed message back to the caller. The following is an extract of the SOAP message, containing a SOAP fault:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns:xsd='http://www.w3.org/2001/XMLSchema'
          xmlns:pcmm='http://www.IPCablecom.com/pcmm/…'>
  <env:Body>
    <env:Fault xmlns:fault="">
      <code>soap:Sender</code>
      <reason>The format of the SubscriberID is not valid!</reason>
      <detail>
                        <pcmm:error-code>1234</pcmm:error-code>
                        <pcmm:error-type>InvalidArgument</pcmm:error-type>
                        <pcmm:error-message>
Original Message: The format of the SubscriberID is not valid!
                    at org.apache.axis.message.SOAPFaultBuilder.createFault(SOAPFaultBuilder.java:221)
           at org.apache.axis.message.SOAPFaultBuilder.endElement(SOAPFaultBuilder.java:128)
…
                        </pcmm:error-message>
             </detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

The remainder of this section lists the exceptions, which are classified as server faults and client faults.

Table 10 - Server Errors

Error-Code	Error-Type	Comments
1	InsufficientResources	Thrown by IPCablecom Multimedia component, in case of insufficient resources.
2	UnknownGate	Thrown by IPCablecom Multimedia component, in case GateID is unknown.
6	MissingObject	Thrown by IPCablecom Multimedia component, in case of a missing required object.
7	InvalidObject	Thrown by IPCablecom Multimedia component, in case of an invalid object.
10	SessionClassLimitException	Thrown by IPCablecom Multimedia component, in case of session class limit was exceeded.
11	UnknownServiceClass	Thrown by IPCablecom Multimedia component, in case of undefined service class definition.
12	InvalidEnvelop	Thrown by IPCablecom Multimedia component, in case of incompatible envelops.
13	InvalidSubscriber	Thrown by IPCablecom Multimedia component, in case of invalid SubscriberID.
14	UnauthorizedAM	Thrown by IPCablecom Multimedia component, in case of un-authorized AMID.
15	UnsupportedClassifier	Thrown by IPCablecom Multimedia component, in case of an unsupported number of Classifiers.
127	OtherError	Thrown by IPCablecom Multimedia component, in case of any other unspecified errors.

Table 11 - Client Errors

Error-Code	Error-Type	Comments
1025	IllegalSubscriberFormat	AM throws "IllegalArgumentException", if the SubscriberID value is not compliant to specified format.
1026	UnauthorizesAS	AM throws "AuthorizationException", if the AM finds out that the AS was not authorized to perform this request.
1027	InvalidResourceState	Thrown by the AM when it detects an attempt to make a state change that is not allowed, e.g., when a ReserveResourcesRequest message is received for a session that has already been committed.

7 SECURITY REQUIREMENTS

7.1 Introduction

When considering how to secure Web Services it is helpful to decompose the problem into 3 separate "layers":

1. Transport Encryption.
2. Transport Endpoint Authentication,
3. Message (or payload) authentication and authorization.

Each of these layers may be considered independently and can be both implemented and enforced separately. The benefits of this approach are:

- Each of the layers described is essentially separable. This gives developers flexibility in what is implemented at each layer and when it must be implemented.
- The decoupling of each of these layers allows different deployment architectures to be considered. Thus, it is possible to introduce middleware that will enable the routing of SOAP payloads once they have been received by the first "hop" from the client. This middleware may be required to address performance and scale concerns.
- With the separation of message authentication from the transport, it becomes much easier to consider the substitution of different transports. For example, it becomes relatively simple to substitute HTTPS for HTTP without having to change endpoint authentication or message authentication. Or it becomes possible to substitute BEEP for HTTP(S) without having to change message authentication.
- Transport endpoint authentication is, by its very nature, transport dependent. Although HTTP is ubiquitous today, other transports (e.g., BEEP) are coming. Thus, it is unwise to tie the authorization & authentication solely to that layer as it precludes easy migration and would not be as transparent to application developers.
- When considering scalability, reliability and availability, the peer endpoint that a client initially authenticates with may not be the final destination for a message as the message may be routed via one or more middleware hops before it is delivered to the implementer of the web service. Putting the username/password token in the SOAP header allows the information necessary to authenticate and authorize a particular message to pass through the network as part of the WS message payload. As the information is carried in each message, we can make use of load balancers and other techniques that, if we relied on transport endpoint authentication only, would be closed to us without major rework. Given the potential scale of cable networks and the future "self-service" directions it is probably wise to build this in from the start.

Where appropriate, the functions of adjacent layers may be collapsed into a single mechanism. For this specification, Transport Encryption and Transport Endpoint Authentication have been combined into a single mechanism.

7.2 Transport Encryption and Mutual Endpoint Authentication

The Application Server – Application Manager interface MUST be secured using the IPCablecom Profile for TLS as described in section 6.9 of the IPCablecom 1.5 Specifications for Security, see [SEC]. This provides for mutual endpoint authentication and confidentiality of messages.

7.3 Message Authentication & Authorization

The final layer is to provide for authentication and authorization of individual SOAP messages. The AM MAY provide the ability to authenticate and/or authorize the individual SOAP messages. If the AM provides this feature, it SHOULD follow the WS-Security: Username Token Profile V1.0 specification, see [WSS]. This allows a security element to be embedded in the SOAP header which may look like:

```
<wss:Security xmlns:wss="http://schemas.xmlsoap.org/ws/2002/06/secext">
  <wss:usernameToken SOAP:mustUnderstand="1">
    <wss:Username>wallace</wss:Username>
    <wss:Password type="wss:PasswordText">grommit</wss:Password>
  </wss:usernameToken>
</wss:Security>
```

The use of such an approach does not intrude into the message payload of the web service, keeping the issue of security orthogonal to the developers' typical daily tasks while building applications.

Once this username token is received by the service implementation, what use it makes of it is open to choice. Thus, the message may only be authenticated if the requirements are simple or authorization may also be applied.

AN AM MAY mandate the use of WS-Security: Username Token Profile 1.0 [WSS] to authenticate requests. This would be used if multiple users existed within the AS administrative domain that the AM wished to individually authenticate and authorize.

If the AM requires it the AS connecting over HTTP to the AM MUST use a SOAP header conforming to the WS-Security Username Token Profile specification to provide per-message authentication.

If the AM or AS supports WS-Security they MAY require the use of wss:Password elements of either type #PasswordDigest or of type #PasswordText. Either choice is acceptable as the use of the IPCablecom Profile for TLS [SEC] ensures that the passwords will not be vulnerable to snooping or replay.

8 OPEN ISSUES

The following issues remain open topics of discussion. These issues may be addressed in future revisions.

- Currently all provided values for a given element must be used by the AM and over write any values associated with the requested service. There is still an open question to the value of allowing the provided values to be hints and only be used if the service requested allows, otherwise the provisioned value is used.
- Support for passing Accounting information (correlation Ids) from the AS to the AM.

Annex A XML Schema (Normative)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:pcmm="http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS-I02"
xmlns:tns="http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS-I02"
targetNamespace="http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS-I02" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:simpleType name="IPPortNumber">
      <xs:restriction base="xs:nonNegativeInteger">
         <xs:minInclusive value="0"/>
         <xs:maxInclusive value="65535"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="IPv4Address">
      <xs:restriction base="xs:string">
         <xs:pattern value="(([0-9]{1,2}|[1][0-9]{2}|[2][0-4][0-9]|[2][5][0-5])\.){3}([0-9]{1,2}|[1][0-9]{2}|[2][0-4][0-9]|[2][5][0-
5])"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="IPv6Address">
      <xs:annotation>
         <xs:documentation>An IPv6 Address in string format (See RFC 1884, sec 2.2)</xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
         <xs:minLength value="0"/>
         <xs:maxLength value="39"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="MACAddress">
      <xs:restriction base="xs:hexBinary">
         <xs:length value="6"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="CorrelationKeyType">
      <xs:annotation>
         <xs:documentation>An Opaque type used to communicate correlation keys.</xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:hexBinary" />
   </xs:simpleType>
   <xs:simpleType name="IPHostname">
      <xs:restriction base="xs:string">
         <xs:maxLength value="256"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="IPAddress">
      <xs:sequence>
         <xs:choice>
            <xs:element name="ipv4" type="tns:IPv4Address"/>
         </xs:choice>
      </xs:sequence>
   </xs:complexType>
   <!--==========
      Direction
      ==========  -->
   <xs:simpleType name="Direction">
      <xs:restriction base="xs:string">
         <xs:enumeration value="upstream"/>
         <xs:enumeration value="downstream"/>
         <xs:enumeration value="bidirectional"/>
```

```
      </xs:restriction>
   </xs:simpleType>
   <!--==========
      Classifier
      ========== -->

   <xs:complexType name="IPv4Classifier">
      <xs:sequence>
         <xs:element name="protocol" default="0" minOccurs="0">
            <xs:simpleType>
               <xs:restriction base="xs:int">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="257"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="sourceIpAddress" type="tns:IPv4Address" default="0.0.0.0" minOccurs="0"/>
         <xs:element name="sourceIpMask" type="tns:IPv4Address" default="255.255.255.255" minOccurs="0"/>
         <xs:element name="sourcePortStart" type="tns:IPPortNumber" default="0" minOccurs="0"/>
         <xs:element name="sourcePortEnd" type="tns:IPPortNumber" default="65535" minOccurs="0"/>
         <xs:element name="destinationIpAddress" type="tns:IPv4Address" default="0.0.0.0" minOccurs="0"/>
         <xs:element name="destinationIpMask" type="tns:IPv4Address" default="255.255.255.255" minOccurs="0"/>
         <xs:element name="destinationPortStart" type="tns:IPPortNumber" default="0" minOccurs="0"/>
         <xs:element name="destinationPortEnd" type="tns:IPPortNumber" default="65535" minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
   </xs:complexType>
   <xs:complexType name="IPv6Classifier">
      <xs:sequence>
         <xs:element name="nextHeader" default="0">
            <xs:annotation>
               <xs:documentation>This field in IPv6 represents the next protocol header type value and has a similar function to the
protocol type value in IPv4</xs:documentation>
            </xs:annotation>
            <xs:simpleType>
               <xs:restriction base="xs:int">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="257"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="sourceIpAddress" type="tns:IPv6Address"/>
         <xs:element name="sourcePrefixLen" default="128" minOccurs="0">
            <xs:simpleType>
               <xs:restriction base="xs:int">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="128"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="sourcePortStart" type="tns:IPPortNumber" default="0" minOccurs="0"/>
         <xs:element name="sourcePortEnd" type="tns:IPPortNumber" default="65535" minOccurs="0"/>
         <xs:element name="destinationIpAddress" type="tns:IPv6Address"/>
         <xs:element name="destinationPrefixLen" default="128" minOccurs="0">
            <xs:simpleType>
               <xs:restriction base="xs:int">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="128"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
```

```
        <xs:element name="destinationPortStart" type="tns:IPPortNumber" default="0" minOccurs="0"/>
        <xs:element name="destinationPortEnd" type="tns:IPPortNumber" default="65535" minOccurs="0"/>
        <xs:element name="trafficClassLow" default="0" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="255"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="trafficClassHigh" default="255" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="255"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="trafficClassMask" default="255" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="255"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="flowLabel" default="0" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:int">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="1048576"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>
    <xs:complexType name="Classifier">
      <xs:annotation>
        <xs:documentation>The Classifier element allows the AS to identify the traffic flow for which it is requesting resources
for. (Section 6.2.1.3)</xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:choice>
          <xs:element name="IPv4Classifier" type="tns:IPv4Classifier"/>
          <xs:element name="IPv6Classifier" type="tns:IPv6Classifier"/>
        </xs:choice>
        <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>   <!--==========
      ContextID
      ==========  -->
    <xs:complexType name="ContextID">
      <xs:annotation>
        <xs:documentation>
The ContextID element is the identifier that associates the resources of a given session. The ContextID element consists of a
baseId element and optional idExtension elements. (Section 6.2.1.4)
        </xs:documentation>
      </xs:annotation>
      <xs:sequence>
```

```
        <xs:element name="idExtension" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="baseId" type="xs:string"/>
        <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="wildcard" type="xs:boolean" use="optional" default="false"/>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>
    <!--==========
      ContextInfo
      ========== -->
    <xs:complexType name="ContextInfo">
      <xs:annotation>
        <xs:documentation>
The ContextInfo element contains the ContextID element, described in 6.2.1.4, and the element ContextStatus, which identifies the status of the associated IPCablecom Multimedia gates for the session. A context can be in a reserved or committed state.
        </xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="contextId" type="tns:ContextID"/>
        <xs:element ref="tns:ContextStatus" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>
    <xs:complexType name="ContextStatus">
      <xs:sequence>
        <xs:element name="status" type="tns:QoSStatus"/>
        <xs:element name="direction" type="tns:Direction"/>
        <xs:any namespace="##any" processContents="lax" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##any" processContents="lax"/>
    </xs:complexType>
    <xs:simpleType name="QoSStatus">
      <xs:restriction base="xs:string">
        <xs:enumeration value="reserved"/>
        <xs:enumeration value="committed"/>
        <xs:enumeration value="unknown"/>
      </xs:restriction>
    </xs:simpleType>
    <!--==========
      QosChangeEvent
      ========== -->
    <xs:complexType name="QosChangeEvent">
      <xs:sequence>
        <xs:element name="direction" type="tns:Direction"/>
        <xs:element name="changeType">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Idle"/>
              <xs:enumeration value="Authorized"/>
              <xs:enumeration value="Reserved"/>
              <xs:enumeration value="Committed"/>
              <xs:enumeration value="Committed-Recovery"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="reason">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="[0-9]{5}"/>
            </xs:restriction>
```

```
            </xs:simpleType>
          </xs:element>
          <!-- xs:element name="message" type="xs:string" minOccurs="0"/ -->
          <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:complexType>
      <!--==========
        ServiceName
        ========== -->
      <xs:simpleType name="ServiceName">
        <xs:restriction base="xs:string">
          <xs:maxLength value="255"/>
        </xs:restriction>
      </xs:simpleType>
      <!--==========
        SubscriberID
        ========== -->
      <xs:complexType name="SubscriberID">
        <xs:annotation>
          <xs:documentation>
          Identifies the network boundary address for the subscriber. See subscriber ID in PKT-SP-MM-I03-102909 [MM].
          </xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:choice>
            <xs:element name="IPv4Address" type="tns:IPv4Address"/>
            <xs:element name="hostname" type="tns:IPHostname"/>
            <xs:element name="IPv6Address" type="tns:IPv6Address"/>
            <xs:element name="MACAddress" type="tns:MACAddress"/>
          </xs:choice>
          <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:complexType>
      <!--==========
        Timeout
        ========== -->
      <xs:simpleType name="Timeout">
        <xs:restriction base="xs:integer"/>
      </xs:simpleType>
      <!--==========
        TimeUsageLimit
        ========== -->
      <xs:simpleType name="TimeUsageLimit">
        <xs:annotation>
          <xs:documentation>
The TimeUsageLimit is specified in seconds. This element also has a 'direction' attribute indicating whether the limit is for the upstream or downstream directions or for both direction.
          </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:integer"/>
      </xs:simpleType>
      <!--==========
        TrafficProfile
        ========== -->
      <xs:simpleType name="TrafficProfileBandwidth">
        <xs:restriction base="xs:float"/>
      </xs:simpleType>
      <xs:complexType name="TrafficProfileFlowSpec">
        <xs:sequence>
          <xs:element name="serviceNumber">
```

```
            <xs:simpleType>
               <xs:restriction base="xs:int">
                  <xs:enumeration value="2"/>
                  <xs:enumeration value="5"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="bucketRate" type="xs:float"/>
         <xs:element name="bucketDepth" type="xs:float"/>
         <xs:element name="peakRate" type="xs:float"/>
         <xs:element name="maxDatagramSize" type="xs:integer"/>
         <xs:element name="minPolicedUnit" type="xs:integer"/>
         <xs:element name="reservedRate" type="xs:float"/>
         <xs:element name="slackTerm" type="xs:integer"/>
         <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="TrafficProfileTrafficClass">
      <xs:restriction base="xs:string">
         <xs:enumeration value="NetworkControl"/>
         <xs:enumeration value="StreamingControl"/>
         <xs:enumeration value="Voice"/>
         <xs:enumeration value="AV"/>
         <xs:enumeration value="Data"/>
         <xs:enumeration value="Audio"/>
         <xs:enumeration value="Images"/>
         <xs:enumeration value="Gaming"/>
         <xs:enumeration value="Other"/>
         <xs:enumeration value="Background"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="TrafficProfilePriority">
      <xs:restriction base="xs:int">
         <xs:minInclusive value="0"/>
         <xs:maxInclusive value="7"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="TrafficProfileUpstreamDrop">
      <xs:sequence>
         <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="TrafficProfile">
      <xs:sequence>
         <xs:element name="priority" type="tns:TrafficProfilePriority" default="0" minOccurs="0"/>
         <xs:element name="direction" type="tns:Direction"/>
         <xs:choice>
            <xs:element name="bandwidth" type="tns:TrafficProfileBandwidth"/>
            <xs:element name="trafficClass" type="tns:TrafficProfileTrafficClass"/>
            <xs:element name="flowSpec" type="tns:TrafficProfileFlowSpec"/>
            <xs:element name="upstreamDrop" type="tns:TrafficProfileUpstreamDrop"/>
         </xs:choice>

         <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <!--==========
      UserName
      ========== -->
   <xs:simpleType name="UserName">
      <xs:annotation>
         <xs:documentation>
```

```
An identifier for the subscriber using a service
            </xs:documentation>
         </xs:annotation>
         <xs:restriction base="xs:string">
            <xs:maxLength value="256"/>
         </xs:restriction>
      </xs:simpleType>
      <!--==========
         VolumeUsageLimit
         ========== -->
      <xs:complexType name="VolumeUsageLimit">
         <xs:sequence>
            <xs:element name="limit" type="xs:integer"/>
            <xs:element name="direction" type="tns:Direction"/>
            <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
      <!--==========
         ResourceStateNotification
         ========== -->
      <xs:simpleType name="NotificationCause">
         <xs:restriction base="xs:string">
            <xs:enumeration value="Deleted"/>
            <xs:enumeration value="Partially Deleted"/>
            <xs:enumeration value="Informational"/>
         </xs:restriction>
      </xs:simpleType>
      <xs:complexType name="ResourceStateNotification">
         <xs:sequence>
            <xs:element name="contextID" type="tns:ContextID"/>
            <xs:element name="cause" type="tns:NotificationCause"/>
            <xs:element name="statusChange" type="tns:QosChangeEvent" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
      <xs:complexType name="PCMMFault">
         <xs:sequence>
            <xs:element name="error-code" type="xs:string"/>
            <xs:element name="error-type" type="xs:string"/>
            <xs:element name="error-message" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
      <!--==========
         Global element declarations
         ========== -->
      <xs:element name="AppCorrelationKey" type="tns:CorrelationKeyType"/>
      <xs:element name="AMCorrelationKey" type="tns:CorrelationKeyType"/>
      <xs:element name="QosChangeEvent" type="tns:QosChangeEvent"/>
      <xs:element name="PCMMFault" type="tns:PCMMFault"/>
      <xs:element name="ContextStatus" type="tns:ContextStatus"/>
      <!--==========
         WSDL message type definitions
         ========== -->
      <xs:element name="ReserveResourcesReq">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="SubscriberID" type="pcmm:SubscriberID"/>
               <xs:element name="ServiceName" type="pcmm:ServiceName"/>
               <xs:element name="ContextID" type="pcmm:ContextID" minOccurs="0"/>
               <xs:element name="Classifier" type="pcmm:Classifier" minOccurs="0"/>
               <xs:element name="TrafficProfile" type="pcmm:TrafficProfile" minOccurs="0" maxOccurs="unbounded"/>
               <xs:element name="VolumeUsageLimit" type="pcmm:VolumeUsageLimit" minOccurs="0"
maxOccurs="unbounded"/>
```

```
            <xs:element name="TimeUsageLimit" type="pcmm:TimeUsageLimit" minOccurs="0"/>
            <xs:element name="Timeout" type="pcmm:Timeout" minOccurs="0"/>
            <xs:element name="UserName" type="pcmm:UserName" minOccurs="0"/>
            <xs:element ref="tns:AppCorrelationKey" minOccurs="0"/>
            <xs:element name="Extension" minOccurs="0" >
              <xs:complexType>
                <xs:sequence>
                  <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:anyAttribute namespace="##any" processContents="lax"/>
              </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ReserveResourcesRsp">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ContextID" type="pcmm:ContextID"/>
            <xs:element ref="tns:AMCorrelationKey" minOccurs="0"/>
            <xs:element name="Extension" minOccurs="0" >
              <xs:complexType>
                <xs:sequence>
                  <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:anyAttribute namespace="##any" processContents="lax"/>
              </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="CommitResourcesReq">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="SubscriberID" type="pcmm:SubscriberID"/>
            <xs:element name="ServiceName" type="pcmm:ServiceName"/>
            <xs:element name="ContextID" type="pcmm:ContextID" minOccurs="0"/>
            <xs:element name="Classifier" type="pcmm:Classifier" minOccurs="0"/>
            <xs:element name="TrafficProfile" type="pcmm:TrafficProfile" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="VolumeUsageLimit" type="pcmm:VolumeUsageLimit" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="TimeUsageLimit" type="pcmm:TimeUsageLimit" minOccurs="0"/>
            <xs:element name="Timeout" type="pcmm:Timeout" minOccurs="0"/>
            <xs:element name="UserName" type="pcmm:UserName" minOccurs="0"/>
            <xs:element ref="tns:AppCorrelationKey" minOccurs="0"/>
            <xs:element name="Extension" minOccurs="0" >
              <xs:complexType>
                <xs:sequence>
                  <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:anyAttribute namespace="##any" processContents="lax"/>
              </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="CommitResourcesRsp">
        <xs:complexType>
          <xs:sequence>
```

```
        <xs:element name="ContextID" type="pcmm:ContextID"/>
        <xs:element ref="tns:AMCorrelationKey" minOccurs="0"/>
        <xs:element name="Extension" minOccurs="0" >
          <xs:complexType>
            <xs:sequence>
              <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:anyAttribute namespace="##any" processContents="lax"/>
          </xs:complexType>
        </xs:element>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="QueryAvailableServicesReq">
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="QueryAvailableServicesRsp">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ServiceName" type="pcmm:ServiceName" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ReleaseResourcesReq">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SubscriberID" type="pcmm:SubscriberID"/>
        <xs:element name="ServiceName" type="pcmm:ServiceName" minOccurs="0"/>
        <xs:element name="ContextID" type="pcmm:ContextID" minOccurs="0"/>
        <xs:element name="UserName" type="pcmm:UserName" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ReleaseResourcesRsp">
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="QueryContextsReq">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SubscriberID" type="pcmm:SubscriberID" minOccurs="0"/>
        <xs:element name="ServiceName" type="pcmm:ServiceName" minOccurs="0"/>
        <xs:element name="ContextID" type="pcmm:ContextID" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="QueryContextsRsp">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ContextInfo" type="pcmm:ContextInfo" minOccurs="0" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
```

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

Annex B WSDL Specification (Normative)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:pcmm="http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-
PCMM-WS-I02" xmlns:tns="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-PCMM-WS-I02"
targetNamespace="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-PCMM-WS-I02">
   <types>
      <xs:schema elementFormDefault="qualified" targetNamespace="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-
PCMM-WS-I02">
         <xs:import namespace="http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS-I02"
               schemaLocation="http://www.cablelabs.com/PCMM/1.0/xsd/reg/CLAB-PCMM-WS-I02.xsd"/>
      </xs:schema>
   </types>
   <message name="error">
      <part name="message" element="pcmm:PCMMFault"/>
   </message>
   <message name="ReserveResourcesReqMsg">
      <part name="message" element="pcmm:ReserveResourcesReq"/>
   </message>
   <message name="ReserveResourcesRspMsg">
      <part name="message" element="pcmm:ReserveResourcesRsp"/>
   </message>
   <message name="CommitResourcesReqMsg">
      <part name="message" element="pcmm:CommitResourcesReq"/>
   </message>
   <message name="CommitResourcesRspMsg">
      <part name="message" element="pcmm:CommitResourcesRsp"/>
   </message>
   <message name="QueryAvailableServicesReqMsg">
      <part name="message" element="pcmm:QueryAvailableServicesReq"/>
   </message>
   <message name="QueryAvailableServicesRspMsg">
      <part name="message" element="pcmm:QueryAvailableServicesRsp"/>
   </message>
   <message name="ReleaseResourcesReqMsg">
      <part name="message" element="pcmm:ReleaseResourcesReq"/>
   </message>
   <message name="ReleaseResourcesRspMsg">
      <part name="message" element="pcmm:ReleaseResourcesRsp"/>
   </message>
   <message name="QueryContextsReqMsg">
      <part name="message" element="pcmm:QueryContextsReq"/>
   </message>
   <message name="QueryContextsRspMsg">
      <part name="message" element="pcmm:QueryContextsRsp"/>
   </message>
   <portType name="PCMMPortType">
      <operation name="ReserveResourcesOp">
         <input message="tns:ReserveResourcesReqMsg"/>
         <output message="tns:ReserveResourcesRspMsg"/>
         <fault name="error" message="tns:error" />
      </operation>
      <operation name="CommitResourcesOp">
         <input message="tns:CommitResourcesReqMsg"/>
         <output message="tns:CommitResourcesRspMsg"/>
         <fault name="error" message="tns:error" />
      </operation>
```

```
    <operation name="QueryAvailableServicesOp">
      <input message="tns:QueryAvailableServicesReqMsg"/>
      <output message="tns:QueryAvailableServicesRspMsg"/>
      <fault name="error" message="tns:error" />
    </operation>
    <operation name="ReleaseResourcesOp">
      <input message="tns:ReleaseResourcesReqMsg"/>
      <output message="tns:ReleaseResourcesRspMsg"/>
      <fault name="error" message="tns:error" />
    </operation>
    <operation name="QueryContextsOp">
      <input message="tns:QueryContextsReqMsg"/>
      <output message="tns:QueryContextsRspMsg"/>
      <fault name="error" message="tns:error" />
    </operation>
  </portType>

  <!--============
    This is a SOAP 1.2 binding.
    ============ -->
  <binding name="PCMMSampleBinding" type="tns:PCMMPortType">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="ReserveResourcesOp">
      <soap12:operation
        soapAction="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-PCMM-WS/ReserveResources"
        soapActionRequired="true" />
      <input><soap12:body use="literal" /></input>
      <output><soap12:body use="literal" /></output>
      <fault name="error"><soap12:fault name="error" use="literal"/></fault>
    </operation>
    <operation name="CommitResourcesOp">
      <soap12:operation
        soapAction="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-PCMM-WS/CommitResources"
        soapActionRequired="true" />
      <input><soap12:body use="literal" /></input>
      <output><soap12:body use="literal" /></output>
      <fault name="error"><soap12:fault name="error" use="literal"/></fault>
    </operation>
    <operation name="QueryAvailableServicesOp">
      <soap12:operation
        soapAction="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-PCMM-WS/QueryAvailableServicesResources"
        soapActionRequired="true" />
      <input><soap12:body use="literal" /></input>
      <output><soap12:body use="literal" /></output>
      <fault name="error"><soap12:fault name="error" use="literal"/></fault>
    </operation>
    <operation name="ReleaseResourcesOp">
      <soap12:operation
        soapAction="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-PCMM-WS/ReleaseResources"
        soapActionRequired="true" />
      <input><soap12:body use="literal" /></input>
      <output><soap12:body use="literal" /></output>
      <fault name="error"><soap12:fault name="error" use="literal"/></fault>
    </operation>
    <operation name="QueryContextsOp">
      <soap12:operation
        soapAction="http://www.cablelabs.com/PCMM/1.0/wsdl/reg/CLAB-PCMM-WS/QueryContexts"
        soapActionRequired="true" />
      <input><soap12:body use="literal" /></input>
      <output><soap12:body use="literal" /></output>
      <fault name="error"><soap12:fault name="error" use="literal"/></fault>
    </operation>
```

```
    </binding>

    <service name="PCMMSampleService">
        <port name="PCMMSamplePort" binding="tns:PCMMSampleBinding">
            <soap12:address location="http://www.nonexistingserver.com/PCMMSampleAddress" />
        </port>
    </service>
</definitions>
```

Appendix I Web Services Eventing (WS-Eventing)

The following document is provided here in its entirety here as additional information to this specification.

August 2004

Authors

Don Box, Microsoft
Luis Felipe Cabrera, Microsoft
Craig Critchley, Microsoft
Francisco Curbera, IBM
Donald Ferguson, IBM
Alan Geller (Editor), Microsoft
Steve Graham, IBM
David Hull, TIBCO Software
Gopal Kakivaya, Microsoft
Amelia Lewis, TIBCO Software
Brad Lovering, Microsoft
Matt Mihic, BEA Systems
Peter Niblett, IBM
David Orchard, BEA Systems
Junaid Saiyed, Sun Microsystems
Shivajee Samdarshi, TIBCO Software
Jeffrey Schlimmer, Microsoft
Igor Sedukhin, Computer Associates
John Shewchuk, Microsoft
Bill Smith, Sun Microsystems
Sanjiva Weerawarana, IBM
David Wortendyke, Microsoft

Copyright Notice

Abstract

This specification describes a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages.

Composable Architecture

By using the XML, SOAP [SOAP 1.1, SOAP 1.2], and WSDL [WSDL 1.1] extensibility models, the Web service specifications (WS-*) are designed to be composed with each other to provide a rich set of tools to provide security in the Web services environment. This specification specifically relies on other Web service specifications to provide secure, reliable, and/or transacted message delivery and to express Web service and client policy.

Status

This specification is a public draft release and is provided for review and evaluation only. The authors hope to solicit your contributions and suggestions in the near future. The authors make no warrantees or representations regarding the specifications in any manner whatsoever.

Table of Contents

1. Introduction

Web services often want to receive messages when events occur in other services and applications. A mechanism for registering interest is needed because the set of Web services interested in receiving such messages is often unknown in advance or will change over time. This specification defines a protocol for one Web service (called a "subscriber") to register interest (called a "subscription") with another Web service (called an "event source") in receiving messages about events (called "notifications" or "event messages"). The subscriber may manage the subscription by interacting with a Web service (called the "subscription manager") designated by the event source.

To improve robustness, a subscription may be leased by an event source to a subscriber, and the subscription expires over time. The subscription manager provides the ability for the subscriber to renew or cancel the subscription before it expires.

There are many mechanisms by which event sources may deliver events to event sinks. This specification provides an extensible way for subscribers to identify the delivery mechanism they prefer. While asynchronous, pushed delivery is defined here, the intent is that there should be no limitation or restriction on the delivery mechanisms capable of being supported by this specification.

8.1.1.1.1 1.1 Requirements

This specification intends to meet the following requirements:

- Define means to create and delete event subscriptions.
- Define expiration for subscriptions and allow them to be renewed.
- Define how one Web service can subscribe on behalf of another.
- Define how an event source delegates subscription management to another Web service.
- Allow subscribers to specify how event messages should be delivered.
- Leverage other Web service specifications for secure, reliable, transacted message delivery.
- Support complex eventing topologies that allow the originating event source and the final event sink to be decoupled.
- Provide extensibility for more sophisticated and/or currently unanticipated subscription scenarios.
- Support a variety of encoding formats, including (but not limited to) both SOAP 1.1 [SOAP 1.1] and SOAP 1.2 [SOAP 1.2] Envelopes.

8.1.1.1.2 1.2 Delivery Modes

While the general pattern of asynchronous, event-based messages is extremely common, different applications often require different event message delivery mechanisms. For instance, in some cases a simple asynchronous message is optimal, while other situations may work better if the event consumer can poll for event messages in order to control the flow and timing of message arrival. Some consumers will require event messages to be wrapped in a standard "event" SOAP envelope, while others will prefer messages to be delivered unwrapped. Some consumers may require event messages to be delivered reliably, while others may be willing to accept best-effort event delivery.

In order to support this broad variety of event delivery requirements, this specification introduces an abstraction called a Delivery Mode. This concept is used as an extension point, so that event sources and event consumers may freely create new delivery mechanisms that are tailored to their specific requirements. This specification provides a minimal amount of support for delivery mode negotiation by allowing an event source to provide a list of supported delivery modes in response to a subscription request specifying a delivery mode it does not support.

This specification defines a single delivery mode, Push Mode, which is simple asynchronous messaging.

As an example of a possible extension, a feature may allow a subscriber to request that event messages be "wrapped" in a standard message. This feature is requested by specifying a new delivery mode, e.g. http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Wrap, in the Subscribe request. Use of this delivery mode would indicate that notification messages should be "wrapped". Similarly, this approach could be generalized to allow the subscriber to provide other information regarding their preferences for notification message delivery.

1.3 Subscription Managers

In some scenarios the event source itself manages the subscriptions it has created. In other scenarios, for example a geographically distributed publish-and-subscribe system, it may be useful to delegate the management of a subscription to another Web service. To support this flexibility, the response to a subscription request to an event source will

include the EPR of a service that the subscriber may interact with to manage this subscription. This EPR should be the target for future requests to renew or cancel the subscription. It may address the same Web service (Address and ReferenceProperties) as the event source itself, or it may address some other Web service to which the event source has delegated management of this subscription; however, the full subscription manager EPR (Address and Reference Properties and ReferenceParameters) must be unique for each subscription.

We use the term "subscription manager" in this specification to refer to the Web service that manages the subscription, whether it is the event source itself or some separate Web service.

1.4 Example

Table 1 lists a hypothetical request to create a subscription for storm warnings.

Table 1: Hypothetical request to create a subscription

```
(01) <s12:Envelope
(02)    xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)    xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)    xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)    xmlns:ew='http://www.example.com/warnings' >
(06)  <s12:Header>
(07)   <wsa:Action>
(08)     http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
(09)   </wsa:Action>
(10)  <wsa:MessageID>
(11)      uuid:d7c5726b-de29-4313-b4d4-b3425b200839
(12)   </wsa:MessageID>
(13)   <wsa:ReplyTo>
(14)    <wsa:Address>http://www.example.com/MyEventSink</wsa:Address>
(15)   </wsa:ReplyTo>
(16)  <wsa:To>http://www.example.org/oceanwatch/EventSource</wsa:To>
(17)  </s12:Header>
(18)  <s12:Body>
(19)   <wse:Subscribe>
(20)    <wse:Delivery>
(21)     <wse:NotifyTo>
(22)      <wsa:Address>
(23)        http://www.example.com/MyEventSink/OnStormWarning
(24)      </wsa:Address>
(25)      <wsa:ReferenceProperties>
(26)        <ew:MySubscription>2597</ew:MySubscription>
(27)       </wsa:ReferenceProperties>
(28)      </wse:NotifyTo>
(29)    </wse:Delivery>
(30)   </wse:Subscribe>
(31)  </s12:Body>
(32) </s12:Envelope>
```

Lines (07-09) in Table 1 indicate the message is a request to create a subscription, and Line (16) indicates that it is sent to a hypothetical event source of ocean events.

While Lines (13-15) indicate where a reply should be sent, Lines (20-29) indicate where and how notifications should be delivered; there is no requirement that these match. The absence of a Mode attribute on Line (20) indicates that notifications should be delivered using Push mode; that is, they should be asynchronously sent as SOAP messages to the endpoint described in

lines (21-28). Note that Lines (25-27) illustrate a typical pattern where the event sink lists a reference property (Line 26) that identifies the subscription and will be included in each notification.

Table 2 lists a hypothetical response to the request in Table 1.

Table 2: Hypothetical response to a subscribe request

```
(01) <s12:Envelope
(02)     xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)     xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)     xmlns:ow='http://www.example.org/oceanwatch' >
(06)   <s12:Header>
(07)     <wsa:Action>
(08)      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
(09)    </wsa:Action>
(10)    <wsa:RelatesTo>
(11)       uuid:d7c5726b-de29-4313-b4d4-b3425b200839
(12)    </wsa:RelatesTo>
(13)    <wsa:To>http://www.example.com/MyEventSink</wsa:To>
(14)   </s12:Header>
(15)   <s12:Body>
(16)    <wse:SubscribeResponse>
(17)     <wse:SubscriptionManager>
(18)      <wsa:Address>
(19)       http://www.example.org/oceanwatch/SubscriptionManager
(20)      </wsa:Address>
(21)      <wsa:ReferenceParameters>
(22)       <ow:MyId>
(23)         28
(24)       </ow:MyId>
(25)     </wsa:ReferenceParameters>
(26)     </wse:SubscriptionManager>
(27)    <wse:Expires>P0Y0M0DT30H0M0S</wse:Expires>
(28)   </wse:SubscribeResponse>
(29)  </s12:Body>
(30) </s12:Envelope>
```

Lines (07-09) in Table 2 indicate this message is a response to a request to create a subscription, and Lines (10-12) indicate that it is a response to the request in Table 1. Lines (17-26) provide the subscription manager EPR for this subscription, and Line (27) indicates the subscription will expire in 30 hours unless renewed.

2. Notations and Terminology

This section specifies the notations, namespaces, and terminology used in this specification.

2.1 Notational Conventions

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

This specification uses the following syntax to define normative outlines for messages:

- The syntax appears as an XML instance, but values in italics indicate data types instead of values.
- Characters are appended to elements and attributes to indicate cardinality:
  - "?" (0 or 1)
  - "*" (0 or more)
  - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.

- The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- An ellipsis (i.e. "...") indicates a point of extensibility that allows other child or attribute content. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. If a receiver does not recognize an extension, the receiver SHOULD NOT process the message and MAY fault.
- XML namespace prefixes (see Table 3) are used to indicate the namespace of the element being defined.

2.2 XML Namespaces

The XML namespace URI that MUST be used by implementations of this specification is:

http://schemas.xmlsoap.org/ws/2004/08/eventing

Table 3 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 3: Prefixes and XML namespaces used in this specification.

Prefix	XML Namespace	Specification(s)
s	(Either SOAP 1.1 or 1.2)	(Either SOAP 1.1 or 1.2)
s11	http://schemas.xmlsoap.org/soap/envelope/	SOAP 1.1 [SOAP 1.1]
s12	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2 [SOAP 1.2]
wsdl	http://schemas.xmlsoap.org/wsdl/	WSDL [WSDL 1.1]
wsa	http://schemas.xmlsoap.org/ws/2004/08/addressing	WS-Addressing [WS-Addressing]
wse	http://schemas.xmlsoap.org/ws/2004/08/eventing	This specification
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]

2.3 Terminology

Delivery Mode
: The mechanism by which event messages are delivered from the source to the sink.

Event Source
: A Web service that sends notifications and accepts requests to create subscriptions.

Event Sink
: A Web service that receives notifications.

Notification
: A one-way message sent to indicate that an event has occurred.

Push Mode
: A delivery mechanism where the source sends event messages to the sink as individual, unsolicited, asynchronous SOAP messages.

Subscriber
: A Web service that sends requests to create, renew, and/or delete subscriptions.

Subscription Manager
: A Web service that accepts requests to manage get the status of, renew, and/or delete subscriptions on behalf of an event source.

2.4 Compliance

An implementation is not compliant with this specification if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined herein. A SOAP Node MUST NOT use the XML namespace identifier for this specification (listed in Section 2.2) within SOAP Envelopes unless it is compliant with this specification.

SOAP is not a requirement for using the constructs defined in this specification.

Normative text within this specification takes precedence over normative outlines, which in turn takes precedence over the XML Schema and WSDL descriptions.

3. Subscription Messages

To create, renew, and delete subscriptions, subscribers send request messages to event sources and subscription managers.

When an event source accepts a request to create a subscription, it typically does so for a given amount of time, although an event source may accept an indefinite subscription with no time-based expiration. If the subscription manager accepts a renewal request, it updates that amount of time. During that time, notifications are delivered by the event source to the requested event sink. An event source may support filtering to limit notifications that are delivered to the event sink; if it does, and a subscribe request contains a filter, the event source sends only notifications that match the requested filter. The event source sends notifications until one of the following happens: the subscription manager accepts an unsubscribe request for the subscription; the subscription expires without being renewed; or the event source cancels the subscription prematurely. In this last case, the event source makes a best effort to indicate why the subscription ended.

In the absence of reliable messaging at the application layer (e.g. [WS-ReliableMessaging]), messages defined herein are delivered using the quality of service of the underlying transport(s) and on a best-effort basis at the application layer.

3.1 Subscribe

To create a subscription, a subscriber sends a request message of the following form to an event source:

```
<s:Envelope ...>
 <s:Header ...>
  <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
  </wsa:Action>
  ...
 </s:Header>
 <s:Body ...>
  <wse:Subscribe ...>
   <wse:EndTo>endpoint-reference</wse:EndTo> ?
   <wse:Delivery Mode="xs:anyURI"? >xs:any</wse:Delivery>
   <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires> ?
   <wse:Filter Dialect="xs:anyURI"? > xs:any </wse:Filter> ?
   ...
  </wse:Subscribe>
 </s:Body>
</s:Envelope>
```

The following describes additional, normative constraints on the outline listed above:

/s:Envelope/s:Header/wsa:Action
: If a SOAP Action URI is used in the binding for SOAP, the value indicated herein MUST be used for that URI.

/s:Envelope/s:Body/*/wse:EndTo
: Where to send a SubscriptionEnd message if the subscription is terminated unexpectedly. (See 3.4 Subscription End.) If present, this element MUST be of type wsa:EndpointReferenceType. Default is not to send this message.

/s:Envelope/s:Body/*/wse:Delivery
: A delivery destination for notification messages, using some delivery mode. See section 1.2 Delivery Modes for details.

/s:Envelope/s:Body/*/wse:Delivery/@Mode

The delivery mode to be used for notification messages sent in relation to this subscription. Implied value is 'http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push', which indicates that Push Mode delivery should be used. See section 1.2 Delivery Modes for details.

If the event source does not support the requested delivery mode, the request MUST fail, and the event source MAY generate a wse:DeliveryModeRequestedUnavailable fault indicating that the requested delivery mode is not supported.

/s:Envelope/s:Body/*/wse:Delivery/@Mode='http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push'

Value of /s:Envelope/s:Body/*/wse:Delivery is a single element, wse:NotifyTo, that contains the endpoint reference to which notification messages should be sent.

/s:Envelope/s:Body/*/wse:Expires

Requested expiration time for the subscription. (No implied value.) The event source defines the actual expiration and is not constrained to use a time less or greater than the requested expiration. The expiration time may be a specific time or a duration from the subscription's creation time. Both specific times and durations are interpreted based on the event source's clock.

If this element does not appear, then the request is for a subscription that will not expire. That is, the subscriber is requesting the event source to create a subscription with an indefinite lifetime. If the event source grants such a subscription, it may be terminated by the subscriber using an Unsubscribe request, or it may be terminated by the event source at any time for reasons such as connection termination, resource constraints, or system shut-down.

If the expiration time is either a zero duration or a specific time that occurs in the past according to the event source, then the request MUST fail, and the event source MAY generate a wse:InvalidExpirationTime fault indicating that an invalid expiration time was requested.

Some event sources may not have a "wall time" clock available, and so are only able to accept durations as expirations. If such a source receives a Subscribe request containing a specific time expiration, then the request MAY fail; if so, the event source MAY generate a wse:UnsupportedExpirationType fault indicating that an unsupported expiration type was requested.

/s:Envelope/s:Body/*/wse:Filter

A Boolean expression in some dialect, either as a string or as an XML fragment (see /s:Envelope/s:Body/*/wse:Filter/@Dialect). If the expression evaluates to false for a notification, the notification MUST NOT be sent to the event sink. Implied value is an expression that always returns true. If the event source does not support filtering, then a request that specifies a filter MUST fail, and the event source MAY generate a wse:FilteringNotSupported fault indicating that filtering is not supported.

If the event source supports filtering but cannot honor the requested filtering, the request MUST fail, and the event source MAY generate a wse:FilteringRequestedUnavailable fault indicating that the requested filter dialect is not supported.

/s:Envelope/s:Body/*/wse:Filter/@Dialect

Implied value is 'http://www.w3.org/TR/1999/REC-xpath-19991116'.

While an XPath predicate expression provides great flexibility and power, alternate filter dialects may be defined. For instance, a simpler, less powerful dialect might be defined for resource-constrained implementations, or a new dialect might be defined to support filtering based on data not included in the notification message itself. If desired, a filter dialect could allow the definition of a composite filter that contained multiple filters from other dialects.

/s:Envelope/s:Body/*/wse:Filter/@Dialect=' http://www.w3.org/TR/1999/REC-xpath-19991116'

Value of /s:Envelope/s:Body/*/wse:Filter is an XPath [XPath 1.0] predicate expression (PredicateExpr); the context of the expression is:

- Context Node: the SOAP Envelope containing the notification.
- Context Position: 1.
- Context Size: 1.
- Variable Bindings: None.

- Function Libraries: Core Function Library [XPath 1.0].
- Namespace Declarations: The [in-scope namespaces] property [XML Infoset] of /s:Envelope/s:Body/*/wse:Filter.

Other message information headers defined by WS-Addressing [WS-Addressing] MAY be included in the request and response messages, according to the usage and semantics defined in WS-Addressing.

Other components of the outline above are not further constrained by this specification. If the event source accepts a request to create a subscription, it MUST reply with a response of the following form:

```
<s:Envelope ...>
  <s:Header ...>
   <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
   </wsa:Action>
    ...
  </s:Header>
  <s:Body ...>
  <wse:SubscribeResponse ...>
   <wse:SubscriptionManager>
    wsa:EndpointReferenceType
   </wse:SubscriptionManager>
   <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires>
    ...
   </wse:SubscribeResponse>
 </s:Body>
</s:Envelope>
```

The following describes additional, normative constraints on the outline listed above:

/s:Envelope/S:Header/wsa:RelatesTo MUST be the value of the wsa:MessageID of the corresponding request.

/s:Envelope/s:Body/*/wse:SubscriptionManager The EPR of the subscription manager for this subscription. In some cases, it is convenient for all EPRs issued by a single event source to address a single Web service and use a reference parameter to distinguish among the active subscriptions. For convenience in this common situation, this specification defines a global element, wsa:Identifier of type xs:anyURI, that MAY be used as a distinguishing reference parameter if desired by the event source.

/s:Envelope/s:Body/*/wse:Expires
The expiration time assigned by the event source. The expiration time MAY be either an absolute time or a duration but SHOULD be of the same type as the requested expiration (if any).

If this element does not appear, then the subscription will not expire. That is, the subscription has an indefinite lifetime. It may be terminated by the subscriber using an Unsubscribe request, or it may be terminated by the event source at any time for reasons such as connection termination, resource constraints, or system shut-down.

Other components of the outline above are not further constrained by this specification.

If the event source chooses not to accept a subscription, the request MUST fail, and the event source MAY generate a wse:EventSourceUnableToProcess fault indicating that the request was not accepted.

Table 4 lists another hypothetical request to create a subscription.

Table 4: Second hypothetical request to create a subscription

```
(01) <s12:Envelope
(02)     xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)     xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)     xmlns:ew='http://www.example.com/warnings' >
(06)   <s12:Header>
(07)     <wsa:Action>
```

```
(08)      http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
(09)    </wsa:Action>
(10)    <wsa:MessageID>
(11)      uuid:e1886c5c-5e86-48d1-8c77-fc1c28d47180
(12)    </wsa:MessageID>
(13)    <wsa:ReplyTo>
(14)      <wsa:Address>http://www.example.com/MyEvEntsink</wsa:Address>
(15)      <wsa:ReferenceProperties>
(16)        <ew:MySubscription>2597</ew:MySubscription>
(17)      </wsa:ReferenceProperties>
(18)    </wsa:ReplyTo>
(19)    <wsa:To>http://www.example.org/oceanwatch/EventSource</wsa:To>
(20)    </s12:Header>
(21)  <s12:Body>
(22)    <wse:Subscribe>
(23)      <wse:EndTo>
(24)        <wsa:Address>
(25)          http://www.example.com/MyEventSink
(26)        </wsa:Address>
(27)        <wsa:ReferenceProperties>
(28)          <ew:MySubscription>2597</ew:MySubscription>
(29)        </wsa:ReferenceProperties>
(30)      </wse:EndTo>
(31)        <wse:Delivery>
(32)          <wse:NotifyTo>
(33)            <wsa:Address>
(34)                http://www.other.example.com/OnStormWarning
(35)            </wsa:Address>
(36)            <wsa:ReferenceProperties>
(37)              <ew:MySubscription>2597</ew:MySubscription>
(38)            </wsa:ReferenceProperties>
(39)        </wse:NotifyTo>
(40)      </wse:Delivery>
(41)      <wse:Expires>2004-06-26T21:07:00.000-08:00</wse:Expires>
(42)      <wse:Filter xmlns:ow='http://www.example.org/oceanwatch'
(43)        Dialect='http://www.example.org/topicFilter' >
(44)       weather.storms
(45)      </wse:Filter>
(46)    </wse:Subscribe>
(47)  </s12:Body>
(48) </s12:Envelope>
```

Like the request in Table 1, Lines (07-09) of Table 4 indicate the message is a request to create a subscription. Line (19) indicates that it is sent to a hypothetical event source of ocean events.

Lines (13-18) indicate where to send the response to this request, Lines (23-30) indicate where to send a SubscriptionEnd message if necessary, and Lines (31-34) indicate how and where to send notifications.

Line (41) indicates the event sink would prefer to have the subscription expire on 26 June 2004 at 9:07 PM Pacific time.

Lines (42-45) indicate the event sink only wants weather reports where topic is storms, using a custom filter dialect.

Table 5 lists a hypothetical response to the request in Table 4.

Table 5: Hypothetical response to second subscribe request

```
(01) <s12:Envelope
(02)     xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)     xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)     xmlns:ew='http://www.example.com/warnings'
(06)     xmlns:ow='http://www.example.org/oceanwatch' >
```

```
(07) <  s12:Header>
(08)    <wsa:Action>
(09)    http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
(10)    </wsa:Action>
(11)    <wsa:RelatesTo>
(12)       uuid:e1886c5c-5e86-48d1-8c77-fc1c28d47180
(13)    </wsa:RelatesTo>
(14)    <wsa:To>http://www.example.com/MyEventSink</wsa:To>
(15)    <ew:MySubscription>2597</ew:MySubscription>
(16)  </s12:Header>
(17)  <s12:Body>
(18)    <wse:SubscribeResponse>
(19)     <wse:SubscriptionManager>
(20)      <wsa:Address>
(21)        http://www.example.org/oceanwatch/SubscriptionManager
(22)      </wsa:Address>
(23)      <wsa:ReferenceParameters>
(24)      <wse:Identifier>
(25)        uuid:22e8a584-0d18-4228-b2a8-3716fa2097fa
(26)      </wse:Identifier>
(27)     </wsa:ReferenceParameters>
(28)    </wse:SubscriptionManager>
(29)   <wse:Expires>2004-07-01T00:00:00.000-00:00</wse:Expires>
(30)  </wse:SubscribeResponse>
(31)  </s12:Body>
(32) </s12:Envelope>
```

Like the response in Table 2, Lines (08-10) of Table 5 indicate this message is a response to a request to create a subscription, and Lines (11-13) indicate that it is a response to the request in Table 4. Lines (14-15) indicate the response is sent to the event sink indicated in Lines (13-18) of Table 4. Lines (19-28) provide the address of the subscription manager for this subscription; note that this particular response uses the global wse:Identifier element defined by this specification. Finally, Line (29) indicates the subscription will expire on 1 July 2004 unless renewed; there is no requirement that this time be necessarily longer or shorter than the requested expiration (Line (41) of Table 4).

3.2 Renew

To update the expiration for a subscription, subscription managers MUST support requests to renew subscriptions.

To renew a subscription, the subscriber sends a request of the following form to the subscription manager:

```
<s:Envelope ...>
  <s:Header ...>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew
     </wsa:Action>
     ...
  </s:Header>
  <s:Body ...>
    <wse:Renew ...>
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires> ?
       ...
   </wse:Renew>
  </s:Body>
</s:Envelope>
```

Components of the outline listed above are additionally constrained as for a request to create a subscription (see Section 3.1 Subscribe). Other components of the outline above are not further constrained by this specification.

If the subscription manager accepts a request to renew a subscription, it MUST reply with a response of the following form:

```
<s:Envelope ...>
   <s:Header ...>
    <wsa:Action>
```

```
    http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse
   </wsa:Action>
   ...
  </s:Header>
  <s:Body ...>
   <wse:RenewResponse ...>
    <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires> ?
    ...
   </wse:RenewResponse>
  </s:Body>
</s:Envelope>
```

Components of the outline listed above are constrained as for a response to a subscribe request (see Section 3.1 Subscribe) with the following addition(s):

/s:Envelope/s:Body/*/wse:Expires
: If the requested expiration is a duration, then the implied start of that duration is the time when the subscription manager starts processing the Renew request.

If the subscription manager chooses not to renew this subscription, the request MUST fail, and the subscription manager MAY generate a wse:UnableToRenew fault indicating that the renewal was not accepted.

Other components of the outline above are not further constrained by this specification.

Table 6 lists a hypothetical request to renew the subscription created in Table 5.

Table 6: Hypothetical request to renew second subscription

```
(01) <s12:Envelope
(02)     xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)     xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)     xmlns:ow='http://www.example.org/oceanwatch' >
(06)   <s12:Header>
(07)     <wsa:Action>
(08)         http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew
(09)     </wsa:Action>
(10)     <wsa:MessageID>
(11)       uuid:bd88b3df-5db4-4392-9621-aee9160721f6
(12)     </wsa:MessageID>
(13)     <wsa:ReplyTo>
(14)       <wsa:Address>http://www.example.com/MyEventSink</wsa:Address>
(15)     </wsa:ReplyTo>
(16)     <wsa:To>
(17)       http://www.example.org/oceanwatch/SubscriptionManager
(18)      </wsa:To>
(19)     <wse:Identifier>
(20)       uuid:22e8a584-0d18-4228-b2a8-3716fa2097fa
(21)     </wse:Identifier>
(22)   </s12:Header>
(23)   <s12:Body>
(24)    <wse:Renew>
(25)     <wse:Expires>2004-06-26T21:07:00.000-08:00</wse:Expires>
(26)    </wse:Renew>
(27)   </s12:Body>
(28) </s12:Envelope>
```

Lines (07-09) indicate this is a request to renew a subscription. Lines (19-21) contain the reference parameter that indicates the subscription to be renewed is the one created in Table 5. Line (25) in Table 6 indicates the request is to extend the subscription until 26 June 2004 at 9:07 PM Pacific.

Table 7 lists a hypothetical response to the request in Table 6.

Table 7: Hypothetical response to renew request

```
(01) <s12:Envelope
(02)     xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)     xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)     xmlns:ow='http://www.example.org/oceanwatch' >
(06)  <s12:Header>
(07)   <wsa:Action>
(08)    http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse
(09)   </wsa:Action>
(10)   <wsa:RelatesTo>
(11)    uuid:bd88b3df-5db4-4392-9621-aee9160721f6
(12)   </wsa:RelatesTo>
(13)    <wsa:To>http://www.example.com/MyEventSink</wsa:To>
(14)  </s12:Header>
(15)  <s12:Body>
(16)    <wse:RenewResponse>
(17)     <wse:Expires>2004-06-26T12:00:00.000-00:00</wse:Expires>
(18)     </wse:RenewResponse>
(19)   </s12:Body>
(20) </s12:Envelope>
```

Line (17) in Table 7 indicates the subscription has been extended only until 26 June 2004 at noon.

3.3 GetStatus

To get the status of a subscription, the subscriber sends a request of the following form to the subscription manager:

```
<s:Envelope ...>
  <s:Header ...>
   <wsa:Action>
     http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
   </wsa:Action>
   ...
 </s:Header>
 <s:Body ...>
    ...
   </wse:GetStatus>
  </s:Body>
</s:Envelope>
```

Components of the outline listed above are additionally constrained as for a request to renew a subscription (see Section 3.2 Renew). Other components of the outline above are not further constrained by this specification.

If the subscription is valid and has not expired, the subscription manager MUST reply with a response of the following form:

```
<s:Envelope ...>
   <s:Header ...>
    <wsa:Action>
     http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse
    </wsa:Action>
     ...
  </s:Header>
  <s:Body ...>
    <wse:GetStatusResponse ...>
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires> ?
      ...
    </wse:GetStatusResponse>
   </s:Body>
</s:Envelope>
```

Components of the outline listed above are constrained as for a response to a renew request (see Section 3.2 Renew). Other components of the outline above are not further constrained by this specification.

Table 8 lists a hypothetical request to get the status of the subscription created in Table 5.

Table 8: Hypothetical request to get the status of the second subscription

```
(01) <s12:Envelope
(02)    xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)    xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)    xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)    xmlns:ow='http://www.example.org/oceanwatch' >
(06)   <s12:Header>
(07)     <wsa:Action>
(08)      http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
(09)     </wsa:Action>
(10)    <wsa:MessageID>
(11)      uuid:bd88b3df-5db4-4392-9621-aee9160721f6
(12)   </wsa:MessageID>
(13)   <wsa:ReplyTo>
(14)      <wsa:Address>http://www.example.com/MyEventSink</wsa:Address>
(15)    </wsa:ReplyTo>
(16)    <wsa:To>
(17)      http://www.example.org/oceanwatch/SubscriptionManager
(18)    </wsa:To>
(19)    <wse:Identifier>
(20)     uuid:22e8a584-0d18-4228-b2a8-3716fa2097fa
(21)   </wse:Identifier>
(22)  </s12:Header>
(23)  <s12:Body>
(24)    <wse:GetStatus />
(25)  </s12:Body>
(26) </s12:Envelope>
```

Lines (07-09) indicate this is a request to get the status of a subscription. Lines (16-21) indicate that the request is sent to the subscription manager for the subscription created in Table 5.

Table 9 lists a hypothetical response to the request in Table 8.

Table 9: Hypothetical response to get status request

```
(01) <s12:Envelope
(02)   xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)    xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)    xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)    xmlns:ow='http://www.example.org/oceanwatch' >
(06)  <s12:Header>
(07)    <wsa:Action>
(08)     http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse
(09)    </wsa:Action>
(10)    <wsa:RelatesTo>
(11)     uuid:bd88b3df-5db4-4392-9621-aee9160721f6
(12)    </wsa:RelatesTo>
(13)     <wsa:To>http://www.example.com/MyEventSink</wsa:To>
(14)  </s12:Header>
(15)  <s12:Body>
(16)   <wse:GetStatusResponse>
(17)     <wse:Expires>2004-06-26T12:00:00.000-00:00</wse:Expires>
(18)   </wse:GetStatusResponse>
(19)  </s12:Body>
(20) </s12:Envelope>
```

Line (17) in Table 9 indicates the subscription will expire on 26 June 2004 at noon.

3.4 Unsubscribe

Though subscriptions expire eventually, to minimize resources, the subscribing event sink SHOULD explicitly delete a subscription when it no longer wants notifications associated with the subscription.

To explicitly delete a subscription, a subscribing event sink sends a request of the following form to the subscription manager:

```
<s:Envelope ...>
  <s:Header ...>
   <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe
  </wsa:Action>
    ...
  </s:Header>
  <s:Body>
   <wse:Unsubscribe ...>
    ...
   </wse:Unsubscribe>
  </s:Body>
</s:Envelope>
```

Components of the outline above are additionally constrained only as for a request to renew a subscription (see Section 3.2 Renew). For example, the faults listed there are also defined for a request to delete a subscription.

If the subscription manager accepts a request to delete a subscription, it MUST reply with a response of the following form:

```
<s:Envelope ...>
  <s:Header ...>
   <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse
   </wsa:Action>
   <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
    ...
  </s:Header>
  <s:Body />
</s:Envelope>
```

Components of the outline listed above are not further constrained by this specification.

Table 10 lists a hypothetical request to delete the subscription created in Table 5.

Table 10: Hypothetical unsubscribe request to delete second subscription

```
(01) <s12:Envelope
(02)    xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)    xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)    xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)    xmlns:ow='http://www.example.org/oceanwatch' >
(06)   <s12:Header>
(07)    <wsa:Action>
(08)     http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe
(09)   </wsa:Action>
(10)   <wsa:MessageID>
(11)    uuid:2653f89f-25bc-4c2a-a7c4-620504f6b216
(12)   </wsa:MessageID>
(13)   <wsa:ReplyTo>
(14)    <wsa:Address>http://www.example.com/MyEventSink</wsa:Address>
(15)    </wsa:ReplyTo>
(16)    <wsa:To>
(17)     http://www.example.org/oceanwatch/SubscriptionManager
(18)    </wsa:To>
(19)    <wse:Identifier>
(20)     uuid:22e8a584-0d18-4228-b2a8-3716fa2097fa
(21)   </wse:Identifier>
```

```
(22)  </s12:Header>
(23)  <s12:Body>
(24)     <wse:Unsubscribe />
(25)   </s12:Body>
(26) </s12:Envelope>
```

Lines (07-09) in Table 10 indicate the message is a request to delete a subscription. Lines (16-21) indicate that the request is addressed to the manager for the subscription created in Table 5.

Table 11 lists a hypothetical response to the request in Table 10.

Table 11: Hypothetical response to unsubscribe request

```
(01) <s12:Envelope
(02)     xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing' >
(04)  <s12:Header>
(05)  <wsa:Action>
(06) http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse
(07)    </wsa:Action>
(08)     <wsa:RelatesTo>
(09)        uuid:2653f89f-25bc-4c2a-a7c4-620504f6b216
(10)     </wsa:RelatesTo>
(11)     <wsa:To>http://www.example.com/MyEventSink</wsa:To>
(12)   </s12:Header>
(13)   <s12:Body />
(14) </s12:Envelope>
```

3.5 Subscription End

If the event source terminates a subscription unexpectedly, the event source SHOULD send a Subscription End SOAP message to the endpoint reference indicated when the subscription was created (see 3.1 Subscribe.) The message MUST be of the following form:

```
<s:Envelope ...>
   <s:Header ...>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd
   </wsa:Action> ?
    ...
  </s:Header>
  <s:Body ...>
    <wse:SubscriptionEnd ...>
     <wse:SubscriptionManager>
      endpoint-reference
    </wse:SubscriptionManager>
    <wse:Status>
      [
   http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure |
   http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown |
   http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCanceling
      ]
    </wse:Status>
     <wse:Reason xml:lang="language identifier" >xs:string</wse:Reason> ?
    ...
  </wsa:SubscriptionEnd>
    ...
  </s:Body>
</s:Envelope>
```

The following describes additional, normative constraints on the outline listed above:

/s:Envelope/s:Body/*/wse:SubscriptionManager
Endpoint reference of the subscription manager. This element may be used to identify the subscription that has been terminated.

/s:Envelope/s:Body/wse:SubscriptionEnd/wse:Status =
"http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure" This value MUST be used if the event source terminated the subscription because of problems delivering notifications.

/s:Envelope/s:Body/wse:SubscriptionEnd/wse:Status = "http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown"

This value MUST be used if the event source terminated the subscription because the source is being shut down in a controlled manner; that is, if the event source is being shut down but has the opportunity to send a SubscriptionEnd message before it exits.

/s:Envelope/s:Body/wse:SubscriptionEnd/wse:Status = "http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCanceling"

This value MUST be used if the event source terminated the subscription for some other reason before it expired.

/s:Envelope/s:Body/wse:SubscriptionEnd/wse:Reason
This optional element contains text, in the language specified by the @xml:lang attribute, describing the reason for the unexpected subscription termination.

Other message information headers defined by WS-Addressing [WS-Addressing] MAY be included in the message, according to the usage and semantics defined in WS-Addressing.

Other components of the outline above are not further constrained by this specification.

Table 12 lists a hypothetical SubscriptionEnd message corresponding to an early termination of the subscription created in Table 4.

Table 12: Hypothetical subscription end message

```
(01) <s12:Envelope
(02)    xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)    xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)    xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
(05)    xmlns:ew='http://www.example.com/warnings' >
(06)  <s12:Header>
(07)   <wsa:Action>
(08)     http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd
(09)   </wsa:Action>
(10)   <wsa:To>http://www.example.com/MyEventSink</wsa:To>
(11)    <ew:MySubscription>2597</ew:MySubscription>
(12)  </s12:Header>
(13)  <s12:Body>
(14)    <wse:SubscriptionEnd>
(15)     <wse:SubscriptionManager>
(16)     <wsa:Address>
(17)       http://www.example.org/oceanwatch/SubscriptionManager
(18)    </wsa:Address>
(19)    <wsa:ReferenceParameters>
(20)      <wse:Identifier>
(21)       uuid:22e8a584-0d18-4228-b2a8-3716fa2097fa
(22)      </wse:Identifier>
(23) <     /wsa:ReferenceParameters>
(24)    </wse:SubscriptionManager>
(25)    <wse:Code>wse:SourceShuttingDown</wse:Code>
(26)    <wse:Reason xml:lang='en-US' >
(27)      Event source going off line.
(28)    </wse:Reason>
(29)   </wse:SubscriptionEnd>
```

```
(30)  </s12:Body>
(31) </s12:Envelope>
```

Line (08) is the action URI for Subscription End. Lines (10-11) indicate the message is sent to the EndTo of the subscribe request (Lines (23-30) in Table 4.). Line (25) indicates the event source is shutting down, and Lines (26-28) indicate that the event source was going off line.

4. Notifications

This specification does not constrain notifications because any message MAY be a notification.

However, if a subscribing event sink wishes to have notifications specifically marked, it MAY specify literal SOAP header blocks in the Subscribe request, in the /s:Envelope/s:Body/wse:Subscribe/wse:NotifyTo/wsa:ReferenceProperties or /s:Envelope/s:Body/wse:Subscribe/wse:NotifyTo/wsa:ReferenceProperties elements; per WS-Addressing [WS-Addressing], the event source MUST include each such literal SOAP header block in every notification sent to the endpoint addressed by /s:Envelope/s:Body/wse:Subscribe/wse:NotifyTo.

Table 13 lists a hypothetical notification message sent as part of the subscription created by the subscribe request in Table 4.

Table 13: Hypothetical notification message

```
(01) <s12:Envelope
(02)    xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
(03)    xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(04)    xmlns:ew='http://www.example.com/warnings'
(05)    xmlns:ow='http://www.example.org/oceanwatch' >
(06)   <s12:Header>
(07)    <wsa:Action>
(08)      http://www.example.org/oceanwatch/2003/WindReport
(09)    </wsa:Action>
(10)   <wsa:MessageID>
(11)      uuid:568b4ff2-5bc1-4512-957c-0fa545fd8d7f
(12)    </wsa:MessageID>
(13)    <wsa:To>http://www.other.example.com/OnStormWarning</wsa:To>
(14)     <ew:MySubscription>2597</ew:MySubscription>
(15)     <ow:EventTopics>weather.report weather.storms</ow:EventTopics>
(16)  </s12:Header>
(17)  <s12:Body>
(18)     <ow:WindReport>
(19)     <ow:Date>030701</ow:Date>
(20)      <ow:Time>0041</ow:Time>
(21)      <ow:Speed>65</ow:Speed>
(22)      <ow:Location>BRADENTON BEACH</ow:Location>
(23)      <ow:County>MANATEE</ow:County>
(24)      <ow:State>FL</ow:State>
(25)      <ow:Lat>2746</ow:Lat>
(26)      <ow:Long>8270</ow:Long>
(27)     <ow:Comments xml:lang='en-US' >
(28)       WINDS 55 WITH GUSTS TO 65. ROOF TORN OFF BOAT HOUSE. REPORTED
(29)       BY STORM SPOTTER. (TBW)
(30)     </ow:Comments>
(31)    </ow:WindReport>
(32)  </s12:Body>
(33) </s12:Envelope>
```

Lines (13-14) indicate the message is sent to the endpoint indicated by the subscribe request (Lines (32-39) in Table 4). Line (15) matches the filter in the subscribe request (Lines (42-45) in Table 4).

5. Faults

All fault messages defined in this specification MUST be sent according to the rules described in WS-Addressing section 4. They are sent to the [fault endpoint], if present and valid. Otherwise they are sent to the [reply endpoint] if present. If neither is present faults may be sent to the [source endpoint].

Endpoints compliant with this specification MUST include required message information headers on all fault messages. Fault messages are correlated as replies using the [relationship] property as defined in WS-Addressing. The [action] property below designates fault messages:

http://schemas.xmlsoap.org/ws/2004/08/addressing/fault

The definitions of faults use the following properties:

[Code] The fault code.

[Subcode] The fault subcode.

[Reason] The English language reason element.

[Detail] The detail element. If absent, no detail element is defined for the fault.

The properties above bind to a SOAP 1.2 fault as follows:

```
<S:Envelope>
 <S:Header>
  <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
  </wsa:Action>
  <!-- Headers elided for clarity. -->
 </S:Header>
 <S:Body>
  <S:Fault>
   <S:Code>
    <S:Value>[Code]</S:Value>
     <S:Subcode>
       <S:Value>[Subcode]</S:Value>
    </S:Subcode>
   </S:Code>
   <S:Reason>
    <S:Text xml:lang="en">[Reason]</S:Text>
  </S:Reason>
  <S:Detail>
   [Detail]
  </S:Detail>
  </S:Fault>
 </S:Body>
</S:Envelope>
```

The SOAP 1.1 fault is less expressive and map only [Subcode] and [Reason]. These the properties bind to a SOAP 1.1 fault as follows:

```
<S11:Envelope>
 <S11:Body>
  <S11:Fault>
   <faultcode>[Subcode]</faultcode>
   <faultstring xml:lang="en">[Reason]</faultstring>
  </S11:Fault>
 </S11:Body>
</S11:Envelope>
```

5.1 DeliveryModeRequestedUnavailable

This fault is sent when a Subscribe request specifies a delivery mode that is not supported by the event source. Optionally, this fault may contain a list of supported delivery mode URIs in the Detail property.

[Code] s12:Sender

[Subcode] wse:DeliveryModeRequestedUnavailable

[Reason] The requested delivery mode is not supported.

[Detail] <wse:SupportedDeliveryMode> +

Optional; one per delivery mode supported by the receiver

5.2 InvalidExpirationTime

This fault is sent when a Subscribe request specifies an expiration time that is in the past or an expiration duration of zero.

[Code] s12:Sender

[Subcode] wse:InvalidExpirationTime

[Reason] The expiration time requested is invalid.

[Detail] *none*

5.3 UnsupportedExpirationType

This fault is sent when a Subscribe request specifies an expiration time and the event source is only capable of accepting expiration durations; for instance, if the event source does not have access to absolute time.

[Code] s12:Sender

[Subcode] wse:UnsupportedExpirationType

[Reason] Only expiration durations are supported.

[Detail] *none*

5.4 FilteringNotSupported

This fault is sent when a Subscribe request contains a filter and the event source does not support filtering.

[Code] s12:Sender

[Subcode] wse:FilteringNotSupported

[Reason] Filtering is not supported.

[Detail] *none*

5.5 FilteringRequestedUnavailable

This fault is sent when a Subscribe request specifies a filter dialect that the event source does not support. Optionally, this fault may contain a list of supported filter dialect URIs in the Detail property.

[Code] s12:Sender

[Subcode] wse:FilteringRequestedUnavailable

[Reason] The requested filter dialect is not supported.

[Detail] <wse:SupportedDialect> +

Optional; one per filter dialect supported by the receiver

5.6 EventSourceUnableToProcess

This fault is sent when the event source is not capable of fulfilling a Subscribe request for local reasons unrelated to the specific request.

[Code] s12:Receiver

[Subcode] wse:EventSourceUnableToProcess

[Reason] Text explaining the failure; e.g., "The event source has too many subscribers".

[Detail] *none*

5.7 UnableToRenew

This fault is sent when the event source is not capable of fulfilling a Renew request for local reasons unrelated to the specific request.

[Code] s12:Receiver

[Subcode] wse:UnableToRenew

[Reason] *Text explaining the failure; e.g., "The event source has too many subscribers".*

[Detail] *none*

5.8 InvalidMessage

If a request message does not comply with the corresponding outline listed above, the request MUST fail and the event source or subscription manager MAY generate the following fault indicating that the request is invalid:

[Code] s12:Sender

[Subcode] wse:InvalidMessage

[Reason] The message is not valid and cannot be processed.

[Detail] *The invalid message*

6. Security Considerations

6.1 Message Security

It is strongly RECOMMENDED that the communication between services be secured using the mechanisms described in WS-Security [WS-Security]. In order to properly secure messages, the body and all relevant headers need to be included in the signature. Specifically, any headers identified in the <wse:NotifyTo> element and standard messaging headers, such as those

from WS-Addressing [WS-Addressing], need to be signed with the body in order to "bind" the two together. For messages with empty bodies, the <s12:Body> element should be signed so content cannot be added in transit.

Different security mechanisms may be desired depending on the frequency of messages. For example, for infrequent messages, public key technologies may be adequate for integrity and confidentiality. However, for high-frequency events, it may be more performant to establish a security context for the events using the mechanisms described in WS-Trust [WS-Trust] and WS-SecureConversation [WS-SecureConversation].

It should be noted that if a shared secret is used it is RECOMMENDED that derived keys be used to strengthen the secret as described in WS-SecureConversation.

The following list summarizes common classes of attacks that apply to this protocol and identifies the mechanism to prevent/mitigate the attacks:

• **Message alteration** – Alteration is prevented by including signatures of the message information using WS-Security.

• **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using WS-Security.

• **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by comparing secured policies – see WS-Policy [WS-Policy] and WS-SecurityPolicy [WS-SecurityPolicy]).

• **Authentication** – Authentication is established using the mechanisms described in WS-Security and WS-Trust. Each message is authenticated using the mechanisms described in WS-Security.

• **Accountability** – Accountability is a function of the type of and string of the key and algorithms being used. In many cases, a strong symmetric key provides sufficient accountability. However, in some environments, strong PKI signatures are required.

• **Availability** – All reliable messaging services are subject to a variety of availability attacks. Replay detection is a common attack and it is RECOMMENDED that this be addressed by the mechanisms described in WS-Security. Other attacks, such as network-level denial of service attacks are harder to avoid and are outside the scope of this specification. That said, care should be taken to ensure that minimal state is saved prior to any authenticating sequences.

• **Replay** – Messages may be replayed for a variety of reasons. To detect and eliminate this attack, mechanisms should be used to identify replayed messages such as the timestamp/nonce outlined in WS-Security. Alternatively, and optionally, other technologies, such as sequencing, can also be used to prevent replay of application messages.

6.2 Access Control

It is important for event sources to properly authorize requests. This is especially true for Subscribe requests, as otherwise the ability to subscribe on behalf of a third-party event sink could be used to create a distributed denial-of-service attack.

Some possible schemes for validating Subscribe requests include:

- Send a message to the event sink that describes the requested subscription, and then wait for a confirmation message to be returned by the event sink, before the event source accepts the subscription request. While this provides strong assurance that the event sink actually desires the requested subscription, it does not work for event sinks that are not capable of sending a confirmation, and requires additional logic on the event sink.
- Require user authentication on the Subscribe request, and allow only authorized users to Subscribe.

Other mechanisms are also possible. Note that event sources that are not reachable from the Internet have less need to control Subscribe requests.

7. Implementation Considerations

Implementations SHOULD generate expirations in subscribe and renew request and response messages that are significantly larger than expected network latency.

Event sinks should be prepared to receive notifications after sending a subscribe request but before receiving a subscribe response message. Event sinks should also be prepared to receive notifications after receiving an unsubscribe response message.

8. Acknowledgements

This specification has been developed as a result of joint work with many individuals and teams, including:

Josh Cohen (Microsoft)
Geary Eppley (Microsoft)
Omri Gazitt (Microsoft)
Peter Jarvis (Microsoft)
Chris Kaler (Microsoft) Ray McCollum (Microsoft)
Toby Nixon (Microsoft)
Denny Page (TIBCO Software)
Krish Srinivasan (Microsoft)
Anders Vinberg (Microsoft)
Alex Weinert (Microsoft)

9. References

[RFC 2119]	S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997.
[SOAP 1.1]	D. Box, et al, "Simple Object Access Protocol (SOAP) 1.1," May 2000.
[SOAP 1.2]	M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003.
[WS-Addressing]	D. Box et al, "Web Services Addressing (WS-Addressing)," August 2004
[WS-Policy]	D. Box, et al, "Web Services Policy Framework (WS-Policy)," May 2003.
[WS-ReliableMessaging]	R. Bilorusets, et al, "Web Services Reliable Messaging Protocol (WS-ReliableMessaging)," March 2004.
[WS-SecureConversation]	G. Della-Libera et al, "Web Services Secure Conversation Language (WS-SecureConversation)", May, 2004
[WS-Security]	A. Nadalin et al, "Web Services Security: SOAP Message Security 1.0", May, 2004
[WS-SecurityPolicy]	G. Della-Libera, et al, "Web Services Security Policy Language (WS-SecurityPolicy)," December 2002.
[WS-Trust]	S. Anderson, et al, "Web Services Trust Language (WS-Trust)," May 2004.
[WSDL 1.1]	E. Christensen, et al, "Web Services Description Language (WSDL) 1.1," March 2001.
[XML Infoset]	J. Cowan, et al, "XML Information Set," October 2001.
[XML Schema, Part 1]	H. Thompson, et al, "XML Schema Part 1: Structures," May 2001.
[XML Schema, Part 2]	P. Biron, et al, "XML Schema Part 2: Datatypes," May 2001.
[XPath 1.0]	J. Clark, et al, "XML Path Language (XPath) Version 1.0," November 1999.

Appendix I – Service Metadata for Eventing

In order to obtain the event-related metadata that describes a service, the mechanisms described in WS-MetadataExchange should be used. The GetMetadata operation defined there allows WSDL and policy information to be retrieved. The WSDL will contain annotations that identify a service as an event source and that identify those messages that describe notification messages. The policy will specify the delivery modes and filter types supported by the event source.

To indicate that notification and solicit-response operations within a WSDL 1.1 portType are events exposed by an event source, this specification defines an @wse:EventSource attribute to annotate the portType for the event source. The normative outline for the @wse:EventSource attribute is:

<wsd:definitions ...>

```
  <wsdl:import
    namespace='http://schemas.xmlsoap.org/ws/2004/08/eventing'
    location='http://schemas.xmlsoap.org/ws/2004/08/eventing.wsdl' />


 [<wsdl:portType [wse:EventSource='xs:boolean']? >
   [<wsdl:operation ...>
     [
      [<wsdl:input .../>] |
      [<wsdl:output .../>] |
      [<wsdl:input .../> <wsdl:output .../>] |
      [<wsdl:output .../> <wsdl:input .../>]
      ]
    </wsdl:operation>]+
   </wsdl:portType>]*
</wsdl:definitions>
```

The following describes additional, normative constraints on the outline listed above:

/wsdl:definitions/wsdl:portType/@wse:EventSource
If omitted, implied value is false.

/wsdl:definitions/wsdl:portType/@wse:EventSource='true'
Indicates the portType supports the Subscribe operation and indicates that notification and solicit-response operations of the portType are events exposed by a service with a port bound to this portType.

Other components of the outline above are not further constrained by this specification.

For example, here is the WSDL 1.1 for a hypothetical storm warning service that exposes a wind report event.

```
<wsdl:definitions
    targetNamespace="http://www.example.org/oceanwatch"
    xmlns:tns="http://www.example.org/oceanwatch"
    xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
    xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
    xmlns:xs='http://www.w3.org/2001/XMLSchema' >
 <wsdl:import
    namespace='http://schemas.xmlsoap.org/ws/2004/08/eventing'
    location='http://schemas.xmlsoap.org/ws/2004/08/eventing.wsdl' />
 <wsdl:types>
  <xs:schema
    targetNamespace="http://www.example.org/oceanwatch"
    elementFormDefault="qualified"
    blockDefault="#all" >
    <xs:element name="WindReport" >
     <xs:complexType>
      <xs:sequence>
      <xs:element name="Date" type="xs:string" />
      <xs:element name="Time" type="xs:string" />
      <xs:element name="Speed" type="xs:string" />
      <xs:element name="Location" type="xs:string" />
      <xs:element name="County" type="xs:string" />
      <xs:element name="State" type="xs:string" />
      <xs:element name="Lat" type="xs:string" />
      <xs:element name="Long" type="xs:string" />
      <xs:element name="Comments" type="xs:string" />
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   </xs:schema>
```

```
 </wsdl:types>
 <wsdl:message name='WindMsg' >
   <wsdl:part name='body' element='tns:WindReport' />
 </wsdl:message>
 <wsdl:portType name='Warnings' wse:EventSource='true' >
  <wsdl:operation name='WindOp' >
   <wsdl:output message='tns:WindMsg' />
  </wsdl:operation>
 </wsdl:portType>
</wsdl:definitions>
```

As described here, to subscribe to events exposed by an event source, a subscribing endpoint sends a Subscribe message to the endpoint reference for the event source. If the Subscribe does not include a filter, the event sink should expect to receive events defined by notification operations within the portType and should expect to receive and respond to events defined by solicit-response operations within the portType.

Editor's Note: We anticipate that this WSDL extension may change in subsequent versions of this specification.

Appendix II – XML Schema

A normative copy of the XML Schema [XML Schema Part 1, Part 2] for this specification may be retrieved by resolving the XML namespace URI for this specification (listed in Section 2.2 XML Namespaces).

A non-normative copy of the XML schema is listed below for convenience.

```
<xs:schema
  targetNamespace="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:tns="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  blockDefault="#all">


 <xs:import namespace="http://www.w3.org/XML/1998/namespace"
   schemaLocation="http://www.w3.org/2001/xml.xsd" />
 <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
   schemaLocation="http://schemas.xmlsoap.org/ws/2004/08/addressing" />


 <!-- Types and global elements -->
 <xs:complexType name="DeliveryType" mixed="true">
  <xs:sequence>
   <xs:any namespace="##any" processContents="lax"
    minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="Mode" type="xs:anyURI" use="optional" />
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>


 <xs:simpleType name="NonNegativeDurationType">
  <xs:restriction base="xs:duration">
   <xs:minInclusive value="P0Y0M0DT0H0M0S" />
  </xs:restriction>
 </xs:simpleType>


 <xs:simpleType name="ExpirationType">
  <xs:union memberTypes="xs:dateTime tns:NonNegativeDurationType" />
   </xs:simpleType>
```

```
<xs:complexType name="FilterType" mixed="true">
 <xs:sequence>
  <xs:any namespace="##other" processContents="lax"
   minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
 <xs:attribute name="Dialect" type="xs:anyURI" use="optional" />
<xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>


<xs:complexType name="LanguageSpecificStringType">
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute ref="xml:lang" />
    <xs:anyAttribute namespace="##other" processContents="lax" />
   </xs:extension>
 </xs:simpleContent>
</xs:complexType>


<xs:element name="NotifyTo" type="wsa:EndpointReferenceType" />


<!-- Subscribe request -->
<xs:element name="Subscribe">
  <xs:complexType>
    <xs:element name="EndTo" type="wsa:EndpointReferenceType"
     minOccurs="0" />
    <xs:element name="Delivery" type="tns:DeliveryType" />
    <xs:element name="Expires" type="tns:ExpirationType"
     minOccurs="0" />
    <xs:element name="Filter" type="tns:FilterType" minOccurs="0" />
    <xs:any namespace="##other" processContents="lax"
     minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>
</xs:element>


<xs:element name="Identifier" type="xs:anyURI" />


<!-- Subscribe response -->
<xs:element name="SubscribeResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="SubscriptionManager"
             type="wsa:EndpointReferenceType" />
   <xs:element name="Expires" type="tns:ExpirationType" />
   <xs:any namespace="##other" processContents="lax"
    minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>
</xs:element>


<!-- Used in a fault if there's an unsupported dialect -->
<xs:element name="SupportedDialect" type="xs:anyURI" />
```

```
<!-- Used in a fault if there's an unsupported delivery mode -->
<xs:element name="SupportedDeliveryMode" type="xs:anyURI" />


<!-- Renew request -->
<xs:element name="Renew">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Expires" type="tns:ExpirationType"
    minOccurs="0" />
   <xs:any namespace="##other" processContents="lax"
    minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>
</xs:element>


<!-- Renew response -->
 <xs:element name="RenewResponse">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Expires" type="tns:ExpirationType"
     minOccurs="0" />
    <xs:any namespace="##other" processContents="lax"
     minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>
</xs:element>


<!-- GetStatus request -->
<xs:element name="GetStatus">
  <xs:complexType>
   <xs:sequence>
    <xs:any namespace="##other" processContents="lax"
     minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>
</xs:element>


<!-- GetStatus response -->
<xs:element name="GetStatusResponse">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Expires" type="tns:ExpirationType"
     minOccurs="0" />
    <xs:any namespace="##other" processContents="lax"
     minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>
</xs:element>


<!-- Unsubscribe request -->
<xs:element name="Unsubscribe">
  <xs:complexType>
   <xs:sequence>
```

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


<!-- count(/s:Envelope/s:Body/*) = 0 for Unsubscribe response -->


 <!-- SubscriptionEnd message -->
 <xs:element name="SubscriptionEnd">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="SubscriptionManager"
              type="wsa:EndpointReferenceType" />
    <xs:element name="Status"
     type="tns:OpenSubscriptionEndCodeType" />
     <xs:element name="Reason" type="tns:LanguageSpecificStringType"
       minOccurs="0" maxOccurs="unbounded" />
      <xs:any namespace="##other" processContents="lax"
       minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
   <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>
 </xs:element>


 <xs:simpleType name="SubscriptionEndCodeType">
  <xs:restriction base="xs:anyURI">
   <xs:enumeration value=
 "http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure" />
    <xs:enumeration value=
 "http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown" />
    <xs:enumeration value=
 "http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCancelling" />
    </xs:restriction>
 </xs:simpleType>


 <xs:simpleType name="OpenSubscriptionEndCodeType">
   <xs:union memberTypes="tns:SubscriptionEndCodeType xs:anyURI" />
 </xs:simpleType>


  <xs:attribute name="EventSource" type="xs:boolean" />
</xs:schema>
```

Appendix III – WSDL

A normative copy of the WSDL [WSDL 1.1] description can be retrieved from the following address:

http://schemas.xmlsoap.org/ws/2004/08/eventing/eventing.wsdl

A non-normative copy of the WSDL description is listed below for convenience.

```
<wsdl:definitions
    targetNamespace='http://schemas.xmlsoap.org/ws/2004/08/eventing'
    xmlns:wse='http://schemas.xmlsoap.org/ws/2004/08/eventing'
    xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
    xmlns:xs='http://www.w3.org/2001/XMLSchema' >
```

```
<wsdl:types>
 <xs:schema
  targetNamespace='http://schemas.xmlsoap.org/ws/2004/08/eventing'>
   <xs:include schemaLocation='eventing.xsd' />
  </xs:schema>
</wsdl:types>

<wsdl:message name='SubscribeMsg' >
 <wsdl:part name='body' element='wse:Subscribe' />
</wsdl:message>
<wsdl:message name='SubscribeResponseMsg' >
  <wsdl:part name='body' element='wse:SubscribeResponse' />
</wsdl:message>


<wsdl:message name='RenewMsg' >
 <wsdl:part name='body' element='wse:Renew' />
</wsdl:message>
<wsdl:message name='RenewResponseMsg' >
 <wsdl:part name='body' element='wse:RenewResponse' />
</wsdl:message>


<wsdl:message name='GetStatusMsg' >
 <wsdl:part name='body' element='wse:GetStatus' />
</wsdl:message>
<wsdl:message name='GetStatusResponseMsg' >
  <wsdl:part name='body' element='wse:GetStatusResponse' />
</wsdl:message>


<wsdl:message name='UnsubscribeMsg' >
 <wsdl:part name='body' element='wse:Unsubscribe' />
</wsdl:message>
<wsdl:message name='UnsubscribeResponseMsg' />


<wsdl:message name='SubscriptionEnd' >
 <wsdl:part name='body' element='wse:SubscriptionEnd' />
</wsdl:message>


<wsdl:portType name='EventSource' >
 <wsdl:operation name='SubscribeOp' >
   <wsdl:input message='wse:SubscribeMsg' />
   <wsdl:output message='wse:SubscribeResponseMsg' />
  </wsdl:operation>
  <wsdl:operation name='SubscriptionEnd' >
   <wsdl:output message="wse:SubscriptionEnd' />
 </wsdl:operation>
</wsdl:portType>


<wsdl:portType name='SubscriptionManager' >
 <wsdl:operation name='RenewOp' >
  <wsdl:input message='wse:RenewMsg' />
  <wsdl:output message='wse:RenewResponseMsg' />
 </wsdl:operation>
 <wsdl:operation name='GetStatusOp' >
  <wsdl:input message='wse:GetStatusMsg' />
   <wsdl:output message='wse:GetStatusResponseMsg' />
  </wsdl:operation>
```

```
  <wsdl:operation name='UnsubscribeOp' >
  <wsdl:input message='wse:UnsubscribeMsg' />
   <wsdl:output message='wse:UnsubscribeResponseMsg' />
   </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```