Document Number: DSP0226

Date: 2014-06-16

Version: 1.2.0b

Web Services for Management (WS-Management) Specification

Document Type: Specification

Document Status: Work in Progress

Document Language: en-US

CONTENTS

Figures

Tables

Foreword

The *Web Services for Management (WS-Management) Specification* (DSP0226) was prepared by the WS-Management sub-group of the WBEM Infrastructure & Protocols Working Group.

This International Standard makes use of functionality similar to the following W3C Recommendations:

- Web Services Eventing (WS-Eventing)
- Web Services Transfer (WS-Transfer)
- Web Services Enumeration (WS-Enumeration)

These W3C Recommendations were not available at the time WS-Management was defined, and similar functionality was incorporated directly into provisions of the WS-Management specification. Future revisions of WS-Management might incorporate these functions by External Reference to these W3C Recommendations

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

Acknowledgements

The authors wish to acknowledge the following people.

Editors:

- Keith Bankston - Microsoft
- Larry Lamers - VMware

Authors:

- Akhil Arora – Sun Microsystems
- Vince Brunssen – IBM
- Nathan Burkhart – Microsoft
- Mark Carlson – Sun Microsystems
- Josh Cohen – Microsoft
- Doug Davis – IBM
- Jim Davis – WBEM Solutions
- Tony Dicenzo – Oracle
- Mike Dutch – Symantec
- Zulah Eckert – BEA Systems
- George Ericson – EMC
- Wassim Fayed – Microsoft
- Chris Ferris – IBM
- Bob Freund – Hitachi Ltd.
- Eugene Golovinsky – BMC Software
- Yasuhiro Hagiwara – NEC
- Steve Hand – Olocity
- Jackson He – Intel
- David Hines – Intel
- Reiji Inohara – NEC
- Christane Kämpfe – Fujitsu-Siemens Computers
- Paul Knight – Nortel Networks
- Vincent Kowalski – BMC Software
- Heather Kreger – IBM

- Vishwa Kumbalimutt – Microsoft
- Sunil Kunisetty – Oracle
- Richard Landau – Dell
- Paul Lipton – CA
- James Martin – Intel
- Raymond McCollum – Microsoft
- Milan Milenkovic – Intel
- Jeff Mischkinsky – Oracle
- Paul Montgomery – AMD
- Jishnu Mukurji – HP
- Bryan Murray – HP
- Alexander Nosov – Microsoft
- Abhay Padlia – Novell
- Gilbert Pilz – Oracle
- Roger Reich – Symantec
- Brian Reistad – Microsoft
- Larry Russon – Novell
- Tom Rutt – Fujitsu Ltd.
- Jeffrey Schlimmer – Microsoft
- Dr. Hemal Shah – Broadcom
- Sharon Smith – Intel
- Enoch Suen – Dell
- Vijay Tewari – Intel
- William Vambenepe – HP
- Andrea Westerinen – CA, Inc.
- Kirk Wilson – CA, Inc.
- Dr. Jerry Xie – Intel

Contributors:

- Paul C. Allen – Microsoft
- Rodrigo Bomfim – Microsoft
- Don Box – Microsoft
- Jerry Duke – Intel
- David Filani – Intel
- Kirill Gavrylyuk – Microsoft
- Omri Gazitt – Microsoft
- Frank Gorishek – AMD
- Lawson Guthrie – Intel
- Arvind Kumar – Intel
- Brad Lovering – Microsoft
- Pat Maynard – Intel
- Steve Millet – Microsoft
- Matthew Senft – Microsoft
- Barry Shilmover – Microsoft
- Tom Slaight – Intel
- Marvin Theimer – Microsoft
- Dave Tobias – AMD
- John Tollefsrud – Sun
- Anders Vinberg – Microsoft
- Megan Wallent – Microsoft

-

Web Services for Management (WS-Management) Specification

1 Scope

The *Web Services for Management (WS-Management) Specification* describes a Web services protocol based on SOAP for use in management-specific domains. These domains include the management of entities such as PCs, servers, devices, Web services and other applications, and other manageable entities. Services can expose only a WS-Management interface or compose the WS-Management service interface with some of the many other Web service specifications.

A crucial application for these services is in the area of systems management. To promote interoperability between management applications and managed resources, this specification identifies a core set of Web service specifications and usage requirements that expose a common set of operations central to all systems management. This includes the ability to do the following:

- Get, put (update), create, and delete individual resource instances, such as settings and dynamic values
- Enumerate the contents of containers and collections, such as large tables and logs
- Subscribe to events emitted by managed resources
- Execute specific management methods with strongly typed input and output parameters

In each of these areas of scope, this specification defines minimal implementation requirements for conformant Web service implementations. An implementation is free to extend beyond this set of operations, and to choose not to support one or more of the preceding areas of functionality if that functionality is not appropriate to the target device or system.

This specification intends to meet the following requirements:

- Constrain Web services protocols and formats so that Web services can be implemented with a small footprint in both hardware and software management services.
- Define minimum requirements for compliance without constraining richer implementations.
- Ensure backward compatibility and interoperability with WS-Management version 1.0 and 1.1.
- Ensure composability with other Web services specifications.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IETF RFC 2616, R. Fielding et al, *Hypertext Transfer Protocol (HTTP 1.1)*, June 1999, http://tools.ietf.org/html/rfc2616

IETF, RFC 3986, T. Berners-Lee et al, *Uniform Resource Identifiers (URI): Generic Syntax*, August 1998, http://tools.ietf.org/html/rfc3986

IETF, RFC 4122, P. Leach et al, *A Universally Unique Identifier (UUID) URN Namespace*, July 2005, http://tools.ietf.org/html/rfc4122

IETF RFC 4178, L. Zhu et al, *The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism*, October 2005, http://tools.ietf.org/html/rfc4178

IETF, RFC 4559, K. Jaganathan et al, *SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows*, June 2006, http://www.ietf.org/rfc/rfc4559.txt

IETF RFC 5646, A. Phillips et al, *Tags for Identifying Languages*, September 2009, http://tools.ietf.org/rfc/rfc5646.txt

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*, http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

OASIS, A. Nadalin et al, *Web Services Security Username Token Profile 1.0*, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

OASIS, A. Nadalin et al, *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

OASIS, S. Anderson et al, *Web Services Trust Language (WS-Trust)*, December 2005, http://schemas.xmlsoap.org/ws/2005/02/trust

The Unicode Consortium, *The Unicode Standard Version 3.0*, January 2000, http://www.unicode.org/book/u2.html

The Unicode Consortium, *Byte Order Mark (BOM) FAQ*, http://www.unicode.org/faq/utf_bom.html#BOM

W3C, M. Gudgin, et al, *SOAP Version 1.2 Part 1: Messaging Framework*, June 2003, http://www.w3.org/TR/soap12-part1/

W3C, M. Gudgin, et al, *SOAP Version 1.2 Part 2: Adjuncts*, June 2003, http://www.w3.org/TR/2003/REC-soap12-part2-20030624

W3C, M. Gudgin, et al, *SOAP Message Transmission Optimization Mechanism (MTOM)*, November 2004, http://www.w3.org/TR/2004/PR-soap12-mtom-20041116/

W3C, J. Clark et al, *XML Path Language Version 1.0 (XPath 1.0)*, November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

W3C, J. Cowan et al, *XML Information Set Second Edition (XML Infoset)*, February 2004, http://www.w3.org/TR/2004/REC-xml-infoset-20040204/

W3C, H. Thompson et al, *XML Schema Part 1: Structures (XML Schema 1)*, October 2004, http://www.w3.org/TR/xmlschema-1/

W3C, P. Biron et al, *XML Schema Part 2: Datatypes (XML Schema 2)*, October 2004, http://www.w3.org/TR/xmlschema-2/

ISO/IEC 40240:2011 Information technology -- W3C Web Services Addressing 1.0 – Core, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=58365

ISO/IEC 40250:2011 Information technology -- W3C Web Services Addressing 1.0 -- SOAP Binding, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=58375
ISO/IEC 40260:2011 Information technology -- W3C Web Services Addressing 1.0 – Metadata, http://www.iso.org/iso/catalogue_detail?csnumber=58385 W3C, *Extensible Markup Language (XML) 1.0*, W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

W3C, *Namespaces in XML*, W3C Recommendation, January 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114/

W3C, E. Christensen et al, *Web Services Description Language Version 1.1 (WSDL/1.1)*, March 2001, http://www.w3.org/TR/wsdl

W3C, S. Boag et al, XQuery 1.0: An XML Query Language (XQuery 1.0), January 2007, http://www.w3.org/TR/2007/REC-xquery-20070123/

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply. The fact that a normative term such as "shall", “shall not”, “should”, “should not”, “may”, or “need not” may be used in text which does not have an associated rule number does not mean that the text is not normative.

3.1

can

used for statements of possibility and capability, whether material, physical, or causal

3.2

cannot

used for statements of possibility and capability, whether material, physical, or causal

3.3

conditional

indicates requirements to be followed strictly to conform to the document when the specified conditions are met

3.4

mandatory

indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.5

may

indicates a course of action permissible within the limits of the document

3.6

need not

indicates a course of action permissible within the limits of the document

3.7

optional

indicates a course of action permissible within the limits of the document

3.8

shall

indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.9

shall not

indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.10

should

indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.11

should not

indicates that a certain possibility or course of action is deprecated but not prohibited

3.12

client

the application that uses the Web services defined in this document to access the management service

3.13

consumer

the Web service that is requesting the data enumeration from the data source

3.14

data source

a Web service that supports traversal using enumeration contexts via the Enumerate operation defined in this specification

3.15

delivery mode

the mechanism by which notification messages are delivered from the source to the sink

3.16

enumeration context

a session context that represents a specific traversal through a logical sequence of XML element information items using the Pull operation defined in this specification

3.17

event sink

a Web service that receives notifications

3.18

event source

a Web service that sends notifications and accepts requests to create subscriptions

3.19

managed resource

an entity that can be of interest to an administrator

It may be a physical object, such as a laptop computer or a printer, or an abstract entity, such as a service.

3.20

notification

a message sent to indicate that an event has occurred

3.21

push mode

a delivery mechanism where the source sends event messages to the sink as individual, unsolicited SOAP messages

3.22
resource

a Web service that is addressable by an endpoint reference and accessed using the operations defined in this specification. This resource can be represented by an XML document. The XML document may be a representation of managed resource

3.23
resource class

an abstract representation (type) of a managed resource

A resource class defines the representation of management-related operations and properties. An example of a resource class is the description of operations and properties for a set of laptop computers.

3.24
resource factory

a Web service that is capable of creating new resources using the Create operation defined in this specification

3.25
resource instance

an instantiation of a resource class

An example is the set of management-related operations and property values for a specific laptop computer.

3.26
selector

a resource-relative name and value pair that acts as an instance-level discriminant when used with the WS-Management default addressing model

A selector is essentially a filter or "key" that identifies the desired instance of the resource. A selector may not be present when service-specific addressing models are used.

The relationship of services to resource classes and instances is as follows:

- A service consists of one or more resource classes.
- A resource class may contain zero or more instances.

If more than one instance for a resource class exists, they are isolated or identified through parts of the SOAP address for the resource, such as the ResourceURI and SelectorSet fields in the default addressing model.

3.27
service

an application that provides management services to clients by exposing the Web services defined in this document

Typically, a service is equivalent to the network "listener," is associated with a physical transport address, and is essentially a type of manageability access point.

3.28
subscriber

a Web service that sends requests to create, renew, and/or delete subscriptions

3.29
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

4 Symbols and Abbreviated Terms

The following symbols and abbreviations are used in this document.

4.1
BNF
Backus-Naur Form (http://foldoc.org/foldoc/?Backus-Naur+Form)

4.2
BOM
byte-order mark

4.3
CQL
CIM Query Language

4.4
EPR
Endpoint Reference

4.5
GSSAPI
Generic Security Services Application Program Interface

4.6
SOAP
Simple Object Access Protocol

4.7
SPNEGO
Simple and Protected GSSAPI Negotiation Mechanism

4.8
SQL
Structured Query Language

4.9
URI
Uniform Resource Identifier

4.10
URL
Uniform Resource Locator

4.11
UTF
UCS Transformation Format

4.12
UUID
Universally Unique Identifier

4.13
WSDL

Web Services Description Language

4.14

WS-Man

Web Services Management

5 Addressing

WS-Management relies on a SOAP-based addressing mechanism (like the one defined in 5.1) to define references to other Web service endpoints and to define some of the headers used in SOAP messages. This addressing mechanism is semantically equivalent and fully wire-compatible with the version of WS-Addressing referenced in WS-Management 1.0. Therefore, this change to WS-Management is fully backward compatible with existing WS-Management implementations.

Clause 5.2 specifies how more than one addressing version may be used with WS-Management, such as the version defined in 5.1 or the W3C Recommendation version of addressing. In this specification, unless explicitly referring to a particular version, the term "Addressing" refers generically to either version of addressing as defined in 5.2.

Multiple addressing models may be used with any of the addressing versions described in 5.2. Implementations may implement any of the following addressing models:

- basic addressing as defined in 5.1
- the Default Addressing Model as defined in 5.4.2
- new addressing models that are not defined in this specification. These addressing models may impose additional restrictions or requirements for addressing.

5.1 Management Addressing

The features defined in this clause provide a transport-neutral mechanism to address Web services and messages. Specifically, this clause defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. This enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

5.1.1 Introduction

This clause defines two interoperable constructs, endpoint references and message information headers, that convey information that is typically provided by transport protocols and messaging systems. These constructs normalize this underlying information into a uniform format that can be processed independently of transport or application.

A Web service endpoint is an entity, processor, or resource that can be referenced and can be targeted for Web service messages. Endpoint references convey the information needed to identify and reference a Web service endpoint, and they may be used in several different ways:

- Endpoint references are suitable for conveying the information needed to access a Web service endpoint.
- Endpoint references are also used to provide addresses for individual messages sent to and from Web services.

To deal with the latter use case, this clause defines a family of message information headers that allows uniform addressing of messages independent of underlying transport. These message information headers convey end-to-end message characteristics including addressing for source and destination endpoints as well as message identity.

EXAMPLE: The following example illustrates the use of these mechanisms in a SOAP 1.2 message being sent from http://business456.example/client1 to http://fabrikam123.example/Purchasing.

Lines (002) to (014) represent the header of the SOAP message where the mechanisms defined in this clause are used. The body is represented by lines (015) to (017).

Lines (003) to (013) contain the message information header blocks. Specifically, lines (003) to (005) specify the identifier for this message, lines (006) to (008) specify the endpoint from where the message originated, and lines (009) to (011) specify the endpoint to which replies to this message should be sent as an Endpoint Reference. Line (012) specifies the address URI of the ultimate receiver of this message. Line (013) specifies an Action URI identifying expected semantics.

```
(001) <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
                xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(002)   <S:Header>
(003)    <wsa:MessageID>
(004)      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
(005)    </wsa:MessageID>
(006)    <wsa:From>
(007)      <wsa:Address>http://business456.example/client1</wsa:Address>
(008)    </wsa:From>
(009)    <wsa:ReplyTo>
(010)      <wsa:Address>http://business456.example/client1</wsa:Address>
(011)    </wsa:ReplyTo>
(012)    <wsa:To>http://fabrikam123.example/Purchasing</wsa:To>
(013)    <wsa:Action>http://fabrikam123.example/SubmitPO</wsa:Action>
(014)   </S:Header>
(015)   <S:Body>
(016)     ...
(017)   </S:Body>
(018) </S:Envelope>
```

5.1.2 Endpoint References

This clause defines the syntax of an Endpoint Reference (EPR).

5.1.2.1 Format of Endpoint References

This clause defines an XML representation for an endpoint reference as both an XML type (wsa:EndpointReferenceType) and as an XML element (<wsa:EndpointReference>).

The wsa:EndpointReferenceType type is used wherever a Web service endpoint is referenced. The following describes the contents of this type:

```
<wsa:EndpointReference>
    <wsa:Address>xs:anyURI</wsa:Address>
    <wsa:ReferenceProperties>... </wsa:ReferenceProperties> ?
    <wsa:ReferenceParameters>... </wsa:ReferenceParameters> ?
    <wsa:PortType>xs:QName</wsa:PortType> ?
    <wsa:ServiceName PortName="xs:NCName"?>xs:QName</wsa:ServiceName> ?
    <wsp:Policy> ... </wsp:Policy>*
</wsa:EndpointReference>
```

The following describes the attributes and elements listed in the preceding schema overview:

wsa:EndpointReference

This represents some element of type wsa:EndpointReferenceType. This example uses the predefined <wsa:EndpointReference> element, but any element of type wsa:EndpointReferenceType may be used.

wsa:EndpointReference/wsa:Address

This required element (of type xs:anyURI) specifies the address URI that identifies the endpoint. This address may be a logical address or identifier for the service endpoint.

wsa:EndpointReference/wsa:ReferenceProperties/

This optional element contains any number of individual reference properties that are associated with the endpoint to facilitate a particular interaction. Reference properties are XML elements that are required to properly interact with the endpoint. Reference properties are provided by the issuer of the endpoint reference and are otherwise assumed to be opaque to consuming applications.

NOTE: The use of reference properties is deprecated; reference parameters should be used instead.

wsa:EndpointReference/wsa:ReferenceProperties/{any}

Each child element of ReferenceProperties represents an individual reference property.

wsa:EndpointReference/wsa:ReferenceParameters/

This optional element contains any number of individual parameters that are associated with the endpoint to facilitate a particular interaction. Reference parameters are XML elements that are required to properly interact with the endpoint. Reference parameters are also provided by the issuer of the endpoint reference and are otherwise assumed to be opaque to consuming applications.

See 5.4 for some WS-Management-specific reference parameters.

wsa:EndpointReference/wsa:ReferenceParameters/{any}

Each child element of ReferenceParameters represents an individual reference parameter.

wsa:EndpointReference/wsa:PortType

This optional element (of type xs:QName) specifies the value of the primary portType of the endpoint being conveyed.

NOTE: The use of wsa:PortType is deprecated.

wsa:EndpointReference/wsa:ServiceName

This optional element (of type xs:QName) specifies the <wsdl:service> definition that contains a WSDL description of the endpoint being referenced. The service name provides a link to a full description of the service endpoint. An optional non-qualified name identifies the specific port in the service that corresponds to the endpoint.

NOTE: The use of wsa:ServiceName is deprecated.

wsa:EndpointReference/wsa:ServiceName/@PortName

This optional attribute (of type xs:NCName) specifies the name of the <wsdl:port> definition that corresponds to the endpoint being referenced.

wsa:EndpointReference/wsp:Policy

This optional element specifies a policy that is relevant to the interaction with the endpoint.

NOTE: The use of wsp:Policy is deprecated.

wsa:EndpointReference/{any}

This is an extensibility mechanism to allow additional elements to be specified.

wsa:EndpointReference/@{any}

This is an extensibility mechanism to allow additional attributes to be specified.

EXAMPLE: The following example illustrates an endpoint reference. This element references the URI "http://www.fabrikam123.example/acct":

```
<wsa:EndpointReference xmlns:wsa="..." xmlns:fabrikam="...">
   <wsa:Address>http://www.fabrikam123.example/acct</wsa:Address>
</wsa:EndpointReference>
```

5.1.2.2 Binding Endpoint References

When a message needs to be addressed to the endpoint, the information contained in the endpoint reference is mapped to the message according to a transformation that is dependent on the protocol and data representation used to send the message. Protocol-specific mappings (or bindings) define how the information in the endpoint reference is copied to message and protocol fields. This clause defines the SOAP binding for endpoint references. This mapping may be explicitly replaced by other bindings (defined as WSDL bindings or as policies); however, in the absence of an applicable policy stating that a different mapping is to be used, the SOAP binding defined here is assumed to apply. To ensure interoperability with a broad range of devices, all conformant implementations shall support the SOAP binding.

The SOAP binding for endpoint references is defined by the following two rules:

R5.1.2.2-1: The wsa:Address element in the endpoint reference shall be copied in the wsa:To header field of the SOAP message.

R5.1.2.2-2: Each Reference Property and Reference Parameter element becomes a header block in the SOAP message. The elements of each Reference Property or Reference Parameter (including all of its child elements, attributes, and in-scope namespaces) shall be added as a header block in the new message.

EXAMPLE: The following example shows how the default SOAP binding for endpoint references is used to construct a message addressed to the endpoint:

```
<wsa:EndpointReference xmlns:wsa="..." xmlns:fabrikam="...">
   <wsa:Address>http://www.fabrikam123.example/acct</wsa:Address>
   <wsa:ReferenceParameters>
       <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
       <fabrikam:ShoppingCart>ABCDEFG</fabrikam:ShoppingCart>
   </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

According to the mapping rules stated before, the address value is copied in the "To" header and the "CustomerKey" element should be copied literally as a header in a SOAP message addressed to this endpoint. The SOAP message would look as follows:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
         xmlns:wsa="..." xmlns:fabrikam="... ">
   <S:Header>
     ...
    <wsa:To>http://www.fabrikam123.example/acct</wsa:To>
    <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
    <fabrikam:ShoppingCart>ABCDEFG</fabrikam:ShoppingCart>
     ...
   </S:Header>
   <S:Body>
     ...
   </S:Body>
```

```
</S:Envelope>
```

5.1.3 Message Information Headers

This clause defines the syntax of a message information header.

The message information headers collectively augment a message with the headers shown in Figure 1. These headers enable the identification and location of the endpoints involved in an interaction. The basic interaction pattern from which all others are composed is "one way". In this pattern a source sends a message to a destination without any further definition of the interaction.

"Request Reply" is a common interaction pattern that consists of an initial message sent by a source endpoint (the request) and a subsequent message sent from the destination of the request back to the source (the reply). A reply can be an application message, a fault, or any other message.

The message information header blocks provide end-to-end characteristics of a message that can be easily secured as a unit. The information in these headers is immutable and not intended to be modified along the message path.

Figure 1 shows the contents of the message information header blocks:

```
<wsa:MessageID> xs:anyURI </wsa:MessageID>
<wsa:RelatesTo RelationshipType="..."?>xs:anyURI</wsa:RelatesTo>
<wsa:To>xs:anyURI</wsa:To>
<wsa:Action>xs:anyURI</wsa:Action>
<wsa:From>endpoint-reference</wsa:From>
<wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
<wsa:FaultTo>endpoint-reference</wsa:FaultTo>
```

Figure 1 – Message Information Header Blocks

The following describes the attributes and elements listed in Figure 1:

wsa:MessageID

This optional element (of type xs:anyURI) uniquely identifies this message in time and space. This element shall be present if wsa:ReplyTo or wsa:FaultTo is present. No two messages with a distinct application intent may share a wsa:MessageID value. A message may be retransmitted for any purpose (including communications failure) and may use the same wsa:MessageID value. The value of this header is an opaque URI whose interpretation beyond equivalence is not defined in this specification. If a reply is expected, this property shall be present.

wsa:RelatesTo

This optional (repeating) element indicates how this message relates to another message, in the form of a URI-QName pair. The child of this element (which is of type xs:anyURI) contains the wsa:MessageID of the related message or the following well-known URI that means "unspecified message":

```
http://schemas.xmlsoap.org/ws/2004/08/addressing/id/unspecified
```

A reply message shall contain a wsa:RelatesTo header consisting of wsa:Reply and the wsa:MessageID value of the request message.

wsa:RelatesTo/@RelationshipType

This optional attribute (of type xs:QName) conveys the relationship type as a QName. When absent, the implied value of this attribute is wsa:Reply.

This specification has one predefined relationship type, as shown in Table 1:

Table 1 – Relationship Type

QName	Description
wsa:Reply	Indicates that this is a reply to the message identified by the URI.

wsa:ReplyTo

This optional element (of type wsa:EndpointReferenceType) provides an endpoint reference that identifies the intended receiver for replies to this message. This element shall be present if a reply is expected. If this element is present, wsa:MessageID shall be present. If a reply is expected, a message shall contain a wsa:ReplyTo header. The sender shall use the contents of the wsa:ReplyTo to formulate the reply message as defined in 5.1.3.1. If the wsa:ReplyTo header is absent, the contents of the wsa:From header may be used to formulate a message to the source. This header may be absent if the message has no meaningful reply.

wsa:From

This optional element (of type wsa:EndpointReferenceType) provides a reference to the endpoint where the message originated.

wsa:FaultTo

This optional element (of type wsa:EndpointReferenceType) provides an endpoint reference that identifies the intended receiver for faults related to this message. If this element is present, wsa:MessageID shall be present. When formulating a fault message as defined in 5.1.3.1, the sender shall use the contents of this header to formulate the fault message. If this header is absent, the sender should use the contents of the wsa:ReplyTo header to formulate the fault message. If both the wsa:FaultTo and wsa:ReplyTo header are absent, the sender may use the contents of the wsa:From header to formulate the fault message.

wsa:To

This required element (of type xs:anyURI) provides the address of the intended receiver of this message.

wsa:Action

This required element (of type xs:anyURI) uniquely identifies the semantics implied by this message. It is recommended that the value of this header be a URI identifying an input, output, or fault message within a WSDL port type. An action may be explicitly or implicitly associated with the corresponding WSDL definition. Finally, if in addition to the wsa:Action header, a SOAP Action URI is encoded in a request, the URI of the SOAP Action shall either be the same as the one specified by the wsa:Action header, or set to "".

The dispatching of incoming messages is based on two message properties. The mandatory wsa:To and wsa:Action header identify the target processing location and the verb or intent of the message.

Due to the range of network technologies currently in wide-spread use (for example, NAT, DHCP, and firewalls), many deployments cannot assign a meaningful global URI to a given endpoint. To allow these "anonymous" endpoints to initiate message exchange patterns and receive replies, Addressing defines the following well-known URI for use by endpoints that cannot have a stable, resolvable URI:

```
http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
```

Requests whose wsa:ReplyTo, wsa:From and/or wsa:FaultTo headers use this address shall provide some out-of-band mechanism for delivering replies or faults (for example, returning the reply on the same transport connection). This mechanism may be a simple request/reply transport protocol (for example, HTTP GET or POST). This URI may be used as the wsa:To header for reply messages and should not be used as the wsa:To header in other circumstances.

5.1.3.1 Formulating a Reply Message

The reply to an Addressing compliant request message shall be constructed according to the rules defined in this clause.

EXAMPLE 1: The following example illustrates a request message using message information header blocks in a SOAP 1.2 message:

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:f123="http://www.fabrikam123.example/svc53">
  <S:Header>
  <wsa:MessageID>uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff
    </wsa:MessageID>
    <wsa:ReplyTo>
    <wsa:Address>http://business456.example/client1</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To S:mustUnderstand="1">mailto:joe@fabrikam123.example</wsa:To>
    <wsa:Action>http://fabrikam123.example/mail/Delete</wsa:Action>
  </S:Header>
  <S:Body>
    <f123:Delete>
       <maxCount>42</maxCount>
    </f123:Delete>
  </S:Body>
</S:Envelope>
```

EXAMPLE 2: The following example illustrates a reply message using message information header blocks in a SOAP 1.2 message:

```
<S:Envelope
  xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:f123="http://www.fabrikam123.example/svc53">
  <S:Header>
    <wsa:MessageID>
      uuid:aaaabbbb-cccc-dddd-eeee-wwwwwwwwwww
    </wsa:MessageID>
    <wsa:RelatesTo>
      uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff
    </wsa:RelatesTo>
    <wsa:To>
      http://business456.example/client1
    </wsa:To>
    <wsa:Action>http://fabrikam123.example/mail/DeleteAck</wsa:Action>
  </S:Header>
  <S:Body>
    <f123:DeleteAck/>
  </S:Body>
</S:Envelope>
```

5.1.3.2 Associating Action with WSDL Operations

Addressing defines two mechanisms, explicit association and default action pattern, to associate an action with input, output, and fault elements within a WSDL port type.

5.1.3.2.1 Explicit Association

The action may be explicitly associated using the wsa:Action attribute.

EXAMPLE: Consider the following WSDL excerpt:

```
<definitions targetNamespace="http://example.com/stockquote" ...>
  ...
  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetTradePricesInput"
            wsa:Action="http://example.com/GetQuote"/>
      <output message="tns:GetTradePricesOutput"
            wsa:Action="http://example.com/Quote"/>
    </operation>
  </portType>
  ...
</definitions>
```

The action for the input of the GetLastTradePrice operation within the StockQuotePortType is explicitly defined to be http://example.com/GetQuote. The action for the output of this same operation is http://example.com/Quote.

5.1.3.2.2 Default Action Pattern

In the absence of the wsa:Action attribute, the following pattern is used to construct a default action for inputs and outputs. The general form of an action URI is as follows:

```
targetNamespace/portTypeName/(inputName|outputNname)
```

The "/" is a literal character to be included in the action. The values of the properties are as follows:

- *targetNamespace* is the target namespace (/definition/@targetNamespace). If *target namespace* ends with a "/" an additional "/" is not added.
- *portTypeName* is the name of the port type (/definition/portType/@name).
- (*inputName*|*outputName*) is the name of the element as defined in Section 2.4.5 of WSDL 1.1.

For fault messages, this pattern is not applied. Instead, the following URI is the default action URI for fault messages:

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

EXAMPLE: Consider the following WSDL excerpt:

```
<definitions targetNamespace="http://example.com/stockquote" ...>
  ...
  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetTradePricesInput" name="GetQuote"/>
      <output message="tns:GetTradePricesOutput" name="Quote"/>
    </operation>
  </portType>
  ...
</definitions>
```

targetNamespace = http://example.com/stockquote *portTypeName* = StockQuotePortType

inputName = GetQuote

outputName = Quote

Applying the preceding pattern with these values produces the following:

input action = http://example.com/stockquote/StockQuotePortType/GetQuote

output action = http://example.com/stockquote/StockQuotePortType/Quote

WSDL defines rules for a default input or output name if the name attribute is not present. Consider the following example:

EXAMPLE: The following is a WSDL excerpt:

```
<definitions targetNamespace="http://example.com/stockquote" ...>
  ...
  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetTradePricesInput"/>
      <output message="tns:GetTradePricesOutput"/>
    </operation>
  </portType>
  ...
</definitions>
```

targetNamespace = http://example.com/stockquote

portTypeName = StockQuotePortType

According to the rules defined in 2.4.5 of WSDL, if the name attribute is absent for the input of a request response operation, the default value is the name of the operation with "Request" appended.

inputName = GetLastTradePriceRequest

Likewise, the output defaults to the operation name with "Response" appended.

outputName = GetLastTradePriceResponse

Applying the previous pattern with these values produces the following:

input action = http://example.com/stockquote/StockQuotePortType/GetLastTradePriceRequest

output action = http://example.com/stockquote/StockQuotePortType/GetLastTradePriceResponse

5.2 Versions of Addressing

To maintain compatibility with implementations of previous versions of WS-Management, this protocol accommodates messages formatted by those previous versions. However, WS-Management 1.2 and 1.1 also allow for the optional use of the WS-Addressing W3C Recommendation.

The following abbreviations are used for clarity and brevity.

- "WSMA" refers to the version of Management Addressing as specified in 5.1.
- "WSA-Rec" refers to the WS-Addressing W3C Recommendation.
- "WS-Man 1.0" refers to the *WS-Management Specification* 1.0 and implementations compatible with that specification.

- "WS-Man 1.2" refers to this specification and implementations compatible with this specification.
- “Addressing Anonymous URI” refers to the anonymous URI that is defined by the version of Addressing currently in use. The anonymous URI defined by WSA-Rec is http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous. The anonymous URI defined by WSMA is http://www.w3.org/2005/08/addressing/anonymous.

NOTE: Some information in this clause is implementation advice to clients on algorithms for efficient communication with unknown services. This informative advice should not be construed to place normative requirements on the behavior of compliant clients or services.

5.2.1 Technical Differences

The WSMA and WSA-Rec specifications reference different XML namespaces. An endpoint sending Web service messages shall use, for the Addressing SOAP headers, one namespace or the other; a receiving endpoint may recognize one namespace or both namespaces. Existing implementations of WS-Man 1.0 are limited to recognizing only the WSMA namespace. Interactions between WS-Man 1.0 and WS-Man 1.2 or 1.1 implementations will have to allow for these limitations.

5.3 Requirements for Compatibility

To maximize interoperability of WS-Management implementations, WS-Man 1.0 , WS-Man 1.1, and WS-Man 1.2 clients and services need to be able to exchange messages. These requirements are summarized in Table 2.

Table 2 – Interoperability Requirements

Interoperability Requirements between WS-Management Versions	WS-Man 1.0 Service	WS-Man 1.1 Service	WS-Man 1.2 Service
WS-Man 1.0 client	It works.	WS-Man 1.0 client needs to be able to access WS-Man 1.1 service, but some negotiation might be needed.	It works, but some negotiations might be needed.
WS-Man 1.1 client	WSMan 1.1 client needs to be able to access 1.0 service.	It works, but some negotiations might be needed.	It works, but some negotiations might be needed.
WS-Man 1.2 client	It works, but some negotiations are required.	It works, but some negotiations are required.	It works.

Homogeneous pairings of compliant clients and services (that is, a version 1.0 client with a version 1.0 service, or a version 1.2 client with a version 1.2 service) can exchange messages in accordance with their respective specifications. To ensure reliable communications, heterogeneous pairings need to meet certain requirements and implement certain sequencing strategies.

In particular, clients and services that implement WS-Man 1.0 can use only WSMA in any exchanges; therefore, all exchanges with version 1.0 endpoints use only WSMA. This conclusion is summarized in Table 3.

Table 3 – WSA Versions in Exchanges

Interoperable Version of Addressing	WS-Man 1.0 Service	WS-Man 1.1 or WS-Man 1.2 Service
WS-Man 1.0 client	WSMA	WSMA
WS-Man 1.1 or WS-Man 1.2 client	WSMA	WSMA or WSA-Rec

5.3.1 Discovery or Negotiation

If it is possible for a client to determine the capabilities of the service with respect to WSA, such discovery is more efficient than negotiating the WSA version. For instance, if a service supports Identify, then a client can determine in advance the WS-Man protocol, as well as an Addressing version or versions supported by the service. For this reason, support of Identify is mandatory in this specification when WSA-Rec is used.

Identify would be used as follows:

- The client sends the service an Identify message.
- If the service does not support Identify, the client can conclude that the service is a WS-Man 1.0 implementation and only supports WSMA.
- If the service successfully processes the Identify message, the client examines the versions of Addressing by looking at the AddressingVersionURI element (as defined in clause 11), if present, and can choose the appropriate version.
- If the Identify response message does not contain any Addressing versions, then there is no way for the client to know which version of Addressing to use and it would need to use one of the strategies described in 5.3.2.

In any case, to avoid unnecessary re-discovery or re-negotiation, a WS-Man 1.1 or 1.2 client should retain information about the capabilities of service endpoints where practical.

5.3.2 Client Negotiation Strategies

A compliant WS-Man 1.0 client will use only WSMA in message exchanges. A WS-Man 1.1 or WS-Man 1.2 client, however, may use either WSMA or WSA-Rec in message exchanges. If a WS-Man client does not know the WSA version capabilities of a service, it may use different strategies when initially contacting the service. The client may begin a message exchange with either version of WSA, using WSA-Rec or WSMA in the request message. The message exchange would proceed as follows:

- Strategy type 1: A client sends the request using WSA-Rec. The WSA-Rec SOAP headers need to be marked with a mustUnderstand=”1” attribute to ensure that a fault will be generated if the receiver does not support the WSA-Rec version of Addressing. The client can then retry the operation using WSMA.
- Strategy type 2: A client sends the request using WSMA. Both WS-Man 1.0 services and WS-Man versions 1.1 and later services respond to the request using WSMA.

5.3.3 Initiating Message Exchanges

Outgoing messages initiated by a WS-Man implementation need to use the same version of Addressing that was used in the Endpoint Reference to which those messages are being sent. For example, if a Subscribe request message uses WSA-Rec in the SOAP headers (for example, for the wsa:To and wsa:ReplyTo), but uses WSMA for the NotifyTo EPR, then the Subscribe response will be sent using WSA-Rec, but the events will be sent using WSMA.

5.3.4 Normative Rules

R5.3.4-1: If a WS-Man service supports WSA-Rec, then it shall also support the Identify operation.

R5.3.4-2: A WS-Man service version 1.1 or later shall support WSMA and should support WSA-Rec.

R5.3.4-3: A WS-Man implementation that is version 1.1 or later shall send messages to endpoints using the same version of Addressing used in the Endpoint Reference of the destination endpoint (see 5.2).

R5.3.4-4: Within a single SOAP message, a WS-Man implementation shall use the same version of Addressing for all Addressing SOAP headers.

Because WS-Man version 1.1 or later allows for either version of Addressing to be used, R5.3.4-4 removes the possibility of mixing the two versions for the WSA SOAP headers, but it does not disallow Endpoint References that might appear elsewhere in the message to be of a different version.

In order to provide a migration path from the WSMA to WSA-Rec, the schema of certain messages allows for either version's EndpointReferenceType to be used. While the schema itself is written in a very generic way (that is, using an xs:any) allowing any arbitrary XML to appear, implementations shall restrict the contents of this element to one of the EndpointReference Types.

NOTE: This allows existing WS-Man 1.0 implementations to be compliant, while providing newer implementations a migration path. In this spirit, newer implementations are strongly encouraged to support both versions of Addressing.

5.4 Use of Addressing in WS-Management

This clause describes the use of Endpoint References regardless of whether an implementation uses WS-Management Addressing (see 5.1) or the W3C Recommendation version of WS-Addressing.

Addressing (either addressing type) endpoint references (EPRs) are used to convey information needed to address a Web service endpoint. WS-Management defines a default addressing model that can optionally be used in EPRs.

5.4.1 Use of Endpoint References

WS-Management uses EPRs as the addressing mechanism for individual resource instances. WS-Management also defines a default addressing model for use in addressing resources. In cases where this default addressing model is not appropriate, such as in systems with well-established addressing models or with EPRs retrieved from a discovery service, services may use those service-specific addressing models if they are based on either addressing version supported by WS-Management.

R5.4.1-1: All messages that are addressed to a resource class or instance that is referenced by an EPR must follow the Addressing rules for representing content from the EPR (the address and reference parameters) in the SOAP message. This rule also applies to continuation messages such as Pull or Release, which continue an operation begun in a previous message. Even though such messages contain contextual information that binds them to a previous operation, the information from the EPR is still required in the message to help route it to the correct handler.

Rule R5.4.1-1 clarifies that messages such as Pull or Renew still require a full EPR. For Pull, for example, this EPR would be the same as the original Enumerate, even though EnumerateResponse returns a context object that would seem to obviate the need for the EPR. The EPR is still required to route the message properly. Similarly, the Renew request uses the SubscriptionManager EPR received in the SubscribeResponse.

When a service includes an EPR in a response message, it must be willing to accept subsequent request messages targeted to that EPR for the same individual managed resource. Clients are not required to process or enhance EPRs given to them by the service before using them to address a managed resource.

R5.4.1-2: An EPR returned by a service shall be acceptable to that service to refer to the same managed resource.

R5.4.1-3: All EPRs returned by a service, whether expressed using the WS-Management default addressing model (see 5.4.2) or any other addressing model, shall be valid as long as the managed resource exists.

5.4.2 WS-Management Default Addressing Model

WS-Management defines a default addressing model for resources. A service is not required to use this addressing model, but it is suitable for many new implementations and can increase the chances of successful interoperation between clients and services.

This document uses examples of this addressing model that contain its component parts, the ResourceURI and SelectorSet SOAP headers. This specification is independent of the actual data model and does not define the structure of the ResourceURI or the set of values for selectors for a given resource. These may be vendor specific or defined by other specifications.

Description and use of this addressing model in this specification do not indicate that support for this addressing model is a requirement for a conformant service.

All of the normative text, examples, and conformance rules in 5.4.2 and 5.4.2.2 presume that the service is based on the default addressing model. In cases where this addressing model is not in use, these rules do not apply.

The default addressing model uses a representation of an EPR that is a tuple of the following SOAP headers:

- **wsa:To** (required): the transport address of the service
- **wsman:ResourceURI** (required if the default addressing model is used): the URI of the resource class representation or instance representation
- **wsman:SelectorSet** (optional): a header that identifies or "selects" the resource instance to be accessed if more than one instance of a resource class exists

The wsman:ResourceURI value needs to be marked with an s:mustUnderstand attribute set to "true" in all messages that use the default addressing model. Otherwise, a service that does not understand this addressing model might inadvertently return a resource that was not requested by the client.

The WS-Management default addressing model is defined in the following XML outline for an EPR:

```
(1)  <wsa:EndpointReference>
(2)    <wsa:Address>
(3)      Network address
(4)    </wsa:Address>
(5)    <wsa:ReferenceParameters>
(6)      <wsman:ResourceURI> resource URI </wsman:ResourceURI>
(7)      <wsman:SelectorSet>
(8)        <wsman:Selector Name="selector-name"> *
(9)          Selector-value
```

```
      </wsman:Selector>
    </wsman:SelectorSet> ?
  </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsa:Address

the URI of the transport address

wsa:ReferenceParameters/wsman:ResourceURI

the URI of the resource class or instance to be accessed

Typically, this URI represents the resource class, but it may represent the instance. The combination of this URI and the wsa:To URI form the full address of the resource class or instance.

wsa:ReferenceParameters/wsman:SelectorSet:

the optional set of selectors as described in 5.4.2.2

These values are used to select an instance if the ResourceURI identifies a multi-instanced target.

When the default addressing model is used in a SOAP message, Addressing specifies that translations take place and the headers are flattened out.

EXAMPLE: The following is an example EPR definition:

```
<wsa:EndpointReference>
  <wsa:Address> Address </wsa:Address>
  <wsa:ReferenceParameters xmlns:wsman="...">
    <wsman:ResourceURI>resURI</wsman:ResourceURI>
    <wsman:SelectorSet>
      <wsman:Selector Name="Selector-name">
        Selector-value
      </wsman:Selector>
    </wsman:SelectorSet>
  </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

This address definition is translated as follows when used in a SOAP message. wsa:Address becomes wsa:To, and the reference parameters are unwrapped and juxtaposed. The following example shows a sample SOAP message using WSMA:

```
<s:Envelope xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <s:Header>
    <wsa:To> Address </wsa:To>
    <wsa:Action> Action URI </wsa:Action>
    <wsman:ResourceURI s:mustUnderstand="true">resURI</wsman:ResourceURI>
    <wsman:SelectorSet>
      <wsman:Selector Name="Selector-name">
        Selector-value
      </wsman:Selector>
    </wsman:SelectorSet>
    ...
  </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

The following message shows a sample SOAP message using WS-Rec:

```
<s:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing ">
```

```
 <s:Header>
   <wsa:To s:mustUnderstand="true"> Address </wsa:To>
   <wsa:Action s:mustUnderstand="true"> Action URI </wsa:Action>
   <wsman:ResourceURI s:mustUnderstand="true"
      wsa:isReferenceParameter="true">resURI</wsman:ResourceURI>
   <wsman:SelectorSet wsa:isReferenceParameter="true">
     <wsman:Selector Name="Selector-name">
       Selector-value
       </wsman:Selector>
   </wsman:SelectorSet>
   ...
  </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

In both cases, the wsa:To, wsman:ResourceURI, and wsman:SelectorSet elements work together to *reference* the resource instance to be managed, but the actual *method* or *operation* to be executed against this resource is indicated by the wsa:Action header.

EXAMPLE: The following is an example of Addressing headers based on the default addressing model in an actual message:

```
<s:Envelope
  xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
  <s:Header>
    ...
    <wsa:To>http://123.99.222.36/wsman</wsa:To>
    <wsman:ResourceURI s:mustUnderstand="true">
      http://example.org/hardware/2005/02/storage/physDisk
    </wsman:ResourceURI>
    <wsman:SelectorSet>
      <wsman:Selector Name="LUN"> 2 </wsman:Selector>
    </wsman:SelectorSet>
    <wsa:Action> http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
      </wsa:Action>
    <wsa:MessageID> urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91
      </wsa:MessageID>
    ...
  </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

The following definitions apply to the preceding message example:

wsa:To
: the network (or transport-level) address of the service

wsman:ResourceURI
: the ResourceURI of the resource class or resource instance to be accessed

wsman:SelectorSet
: a wrapper for the selectors

wsman:SelectorSet/wsman:Selector
: identifies or selects the resource instance to be accessed, if more than one instance of the resource exists

In this case, the selector is "LUN" (logical unit number), and the selected device is unit number "2".

wsa:Action

identifies which operation is to be carried out against the resource (in this case, a "Get")

wsa:MessageID

identifies this specific message uniquely for tracking and correlation purposes

The format defined in RFC 4122 is often used in the examples in this specification, but it is not required.

5.4.2.1 ResourceURI

The ResourceURI is used to indicate the class resource or instance.

R5.4.2.1-1: The format of the wsman:ResourceURI is unconstrained provided that it meets RFC 3986 requirements.

The format and syntax of the ResourceURI is any valid URI according to RFC 3986. Although there is no default scheme, http: and urn: are common defaults. If http: is used, users may expect to find Web-based documentation of the resource at that address. The wsa:To and the wsman:ResourceURI elements work together to define the actual resource being targeted.

R5.4.2.1-2: Vendor-specific or organization-specific URIs should contain the Internet domain name in the first token sequence after the scheme, such as "example.org" in ResourceURI in the following example.

EXAMPLE:

```
<s:Header>
  <wsa:To> http://123.15.166.67/wsman </wsa:To>
  <wsman:ResourceURI>
    http//schemas.example.org/2005/02/hardware/physDisk
  </wsman:ResourceURI>
  ...
</s:Header>
```

R5.4.2.1-3: When the default addressing model is used, the wsman:ResourceURI reference parameter is required in messages with the following wsa:Action URIs:

http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew
http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release
http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe

The following messages require the EPR to be returned in the SubscriptionManager element of the SubscribeResponse message. The format of the EPR is determined by the service and might or might not include the ResourceURI:

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

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

While the ResourceURI SOAP header is required when the WS-Management default addressing mode is used, it may be short and of a very simple form, such as http://example.com/* or http://example.com/resource.

R5.4.2.1-4: For the request message of custom actions (methods), the ResourceURI header may be present in the message to help route the message to the correct handler.

R5.4.2.1-5: The ResourceURI element should not appear in other messages, such as responses or events, unless the associated EPR includes it in its ReferenceParameters.

In practice, the wsman:ResourceURI element is required only in requests to reference the targeted resource class. Responses are not addressed to a management resource, so the wsman:ResourceURI has no meaning in that context.

R5.4.2.1-6: When the default addressing model is used and the wsman:ResourceURI element is missing or in an incorrect form, the service shall issue a wsa:DestinationUnreachable fault with a detail code of

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI

R5.4.2.1-7: The wsman:ResourceURI element shall be used to indicate only the identity of a resource, and it may not be used to indicate the action being applied to that resource, which is properly expressed using the wsa:Action URI.

Custom WSDL-based methods have both a ResourceURI identity from the perspective of addressing and a wsa:Action URI from the perspective of execution. In many cases, the ResourceURI is simply a pseudonym for the WSDL identity and Port, and the wsa:Action URI is the specific method within that port (or interface) definition.

Although a single URI could theoretically be used alone to define an instance of a multi-instance resource, it is recommended that the wsa:To element be used to locate the WS-Management service, that the wsman:ResourceURI element be used to identify the resource class, and that the wsman:SelectorSet element be used to reference the resource instance. If the resource consists of only a single instance, then the wsman:ResourceURI element alone refers to the single instance.

This usage is not a strict requirement, just a guideline. The service can use distinct selectors for any given operation, even against the same resource class, and may allow or require selectors for the Enumerate operation.

See the recommendations in 7.2 regarding addressing uniformity.

Custom actions have two distinct identities: the ResourceURI, which can identify the WSDL and port (or interface), and the wsa:Action URI, which identifies the specific method. If only one method exists in the interface, in a sense the ResourceURI and wsa:Action URI are identical.

It is not an error to use the wsa:Action URI for the ResourceURI of a custom method, but both are still required in the message for uniform processing on both clients and servers.

EXAMPLE 1: The following action to reset a network card might have the following EPR usage:

```
<s:Header>
  <wsa:To>
    http://1.2.3.4/wsman/
  </wsa:To>
  <wsman:ResourceURI>http://example.org/2005/02/networkcards/reset
    </wsman:ResourceURI>
  <wsa:Action>
    http://example.org/2005/02/networkcards/reset
  </wsa:Action>
  ...
</s:Header>
```

In many cases, the ResourceURI is equivalent to a WSDL name and port, and the wsa:Action URI contains an additional token as a suffix, as in the following example.

EXAMPLE 2:

```
<s:Header>
  <wsa:To>
    http://1.2.3.4/wsman
  </wsa:To>
  <wsman:ResourceURI>http://example.org/2005/02/networkcards
    </wsman:ResourceURI>
  <wsa:Action>
    http://example.org/2005/02/networkcards/reset
  </wsa:Action>
  ...
</s:Header>
```

Finally, the ResourceURI may be completely unrelated to the wsa:Action URI, as in the following example.

EXAMPLE 3:

```
<s:Header>
  <wsa:To>http://1.2.3.4/wsman</wsa:To>
  <wsman:ResourceURI>
    http://example.org/products/management/networkcards
  </wsman:ResourceURI>
  <wsa:Action>
    http://example.org/2005/02/netcards/reset
  </wsa:Action>
  ...
</s:Header>
```

All of these uses are legal.

When used with subscriptions, the EPR described by wsa:Address and wsman:ResourceURI (and optionally the wsman:SelectorSet values) identifies the event source to which the subscription is directed. In many cases, the ResourceURI identifies a real or virtual event log, and the subscription is intended to provide real-time notifications of any new entries added to the log. In many cases, the wsman:SelectorSet element might not be used as part of the EPR.

5.4.2.2 Selectors

In the WS-Management default addressing model, selectors are optional elements used to identify instances within a resource class. For operations such as Get or Put, the selectors are used to identify a single instance of the resource class referenced by the ResourceURI.

In practice, because the ResourceURI often acts as a table or a "class," the SelectorSet element is a discriminant used to identify a specific "row" or "instance." If only one instance of a resource class is implied by the ResourceURI, the SelectorSet can be omitted because the ResourceURI is acting as the full identity of the resource. If more than one selector value is required, the entire set of selectors is interpreted by the service in order to reference the specific instance. The selectors are interpreted as being separated by implied logical AND operators.

In some information domains, the values referenced by the selectors are "keys" that are part of the resource content itself, whereas in other domains the selectors are part of a logical or physical directory system or search space. In these cases, the selectors are used to identify the resource, but are not part of the representation.

R5.4.2.2-1: If a resource has more than one instance, a wsman:SelectorSet element may be used to distinguish which instance is targeted if the WS-Management default addressing model is in use. Any number of wsman:Selector values may appear with the wsman:SelectorSet element, as required to identify the precise instance of the resource class. The service may consider the case of selector names and values (see 13.6), as required by the underlying execution environment.

If the client needs to discover the policy on how the case of selector values is interpreted, the service can provide metadata documents that describe this policy. The format of such metadata is beyond the scope of this specification.

R5.4.2.2-2: All content within the SelectorSet element is to be treated as a single reference parameter with a scope relative to the ResourceURI.

R5.4.2.2-3: A service using the WS-Management default addressing model shall examine all selectors in the message and process them as if they were logically joined by AND. If the set of selectors is incorrect for the targeted resource instance, a wsman:InvalidSelectors fault should be returned to the client with the following detail codes:

- if selectors are missing:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelectors
- if selector values are the wrong types:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch
- if the selector value is of the correct type from the standpoint of XML types, but out of range or otherwise illegal in the specific information domain:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue
- if the name is not a recognized selector name

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelectors

R5.4.2.2-4: The Selector Name attribute shall not be duplicated at the same level of nesting. If this occurs, the service should return a wsman:InvalidSelectors fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DuplicateSelectors

This specification does not mandate the use of selectors. Some implementations may decide to use complex URI schemes in which the ResourceURI itself implicitly identifies the instance.

The format of the SelectorSet element is as follows:

```
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
  <s:Header>
    ...
    <wsa:To>  service transport address </wsa:To>
    <wsman:ResourceURI> ResourceURI </wsman:ResourceURI>
    <wsman:SelectorSet>
      <wsman:Selector Name="name"> value </wsman:Selector> +
    </wsman:SelectorSet> ?
    ...
  </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsman:SelectorSet

    the wrapper for one or more Selector elements required to reference the instance

wsman:SelectorSet/wsman:Selector

    used to describe the selector and its value
    If more than one selector is required, one Selector element exists for each part of the overall selector. The value of this element is the Selector value.

wsman:SelectorSet/wsman:Selector/@Name

    the name of the selector (to be treated in a case-insensitive manner)

The value of a selector may be a nested EPR.

EXAMPLE: In the following example, the selector on line 9 is a part of a SelectorSet that contains a nested EPR (lines 10–18) with its own Address, ResourceURI, and SelectorSet elements:

```
(1)  <s:Envelope
(2)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)     xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(5)   <s:Header>
(6)     ...
(7)     <wsman:SelectorSet>
(8)       <wsman:Selector Name="Primary"> 123 </wsman:Selector>
(9)       <wsman:Selector Name="EPR">
(10)        <wsa:EndpointReference>
(11)          <wsa:Address> address </wsa:Address>
(12)          <wsa:ReferenceParameters>
(13)            <wsman:ResourceURI> resource URI </wsman:ResourceURI>
(14)            <wsman:SelectorSet>
(15)              <wsman:Selector Name="name"> value </wsman:Selector>
(16)            </wsman:SelectorSet>
(17)          </wsa:ReferenceParameters>
(18)        </wsa:EndpointReference>
(19)      </wsman:Selector>
(20)    </wsman:SelectorSet>
(21)    ...
(22)  </s:Header>
(23)  <s:Body> ... </s:Body>
(24) </s:Envelope>
```

R5.4.2.2-5: For those services using the WS-Management default addressing model, the value of a wsman:Selector shall be one of the following values:

- a simple type as defined in the XML schema namespace

  http://www.w3.org/2001/XMLSchema

- a nested wsa:EndpointReference using the WS-Management default addressing model

A service may fault selector usage with wsman:InvalidSelectors if the selector is not a simple type or an EPR.

R5.4.2.2-6: A conformant service may reject any selector or nested selector with a nested EPR whose wsa:Address value is not the same as the primary wsa:To value or is not the Addressing Anonymous URI.

The primary purpose for this nesting mechanism is to allow resources that can answer questions about other resources.

R5.4.2.2-7: A service may fail to process a selector name of more than 2048 characters.

R5.4.2.2-8: A service may fail to process a selector value of more than 4096 characters, including any embedded selectors, and may fail to process a message that contains more than 8096 characters of content in the root SelectorSet element.

5.4.2.3 Faults for Default Addressing Model

When faults related to the information in the addressing model based on the default format are generated, they may contain specific fault detail codes. These detail codes are called out separately in 14.6 and do not apply when service-specific addressing is used.

5.4.3 Service-Specific Endpoint References

Although WS-Management specifies a default addressing model, in some cases this model is not available or appropriate.

R5.4.3-1: A conformant service may not understand the header values used by the WS-Management default addressing model. If this is the case, and if the client marks the wsman:ResourceURI with mustUnderstand="true", the service shall return an s:NotUnderstood fault.

R5.4.3-2: A conformant service may require additional header values to be present that are beyond the scope of this specification.

Services can thus use alternative addressing models for referencing resources with WS-Management. These addressing models might or might not use ResourceURI or SelectorSet elements and still be valid addressing models if they conform to the rules of Addressing.

In addition to a defined alternative addressing model, a service might not explicitly define any addressing model at all and instead use an opaque EPR generated at run-time, which is handled according to the standard rules of Addressing.

When such addressing models are used, the client application has to understand and interoperate with discovery methods for acquiring EPRs that are beyond the scope of this specification.

5.4.4 mustUnderstand

This clause describes the use of the mustUnderstand attribute, regardless of whether an implementation uses WS-Management Addressing (see 5.1) or the W3C Recommendation type of WS-Addressing.

The mustUnderstand attribute for SOAP headers is to be interpreted as a "must comply" instruction in WS-Management. For example, if a SOAP header that is listed as being optional in this specification is tagged with mustUnderstand="true", the service is required to comply or return a fault. To ensure that the service treats a header as optional, the mustUnderstand attribute can be omitted.

If the wsa:Action URI is not understood, the implementation might not know how to process the message. So, for the following elements, the omission or inclusion of mustUnderstand="true" has no real effect on the message in practice, because mustUnderstand is implied:

- wsa:To
- wsa:MessageID
- wsa:RelatesTo
- wsa:Action
- wsa:ReplyTo
- wsa:FaultTo

R5.4.4-1: A conformant service shall process any of the preceding elements identically regardless of whether mustUnderstand="true" is present.

As a corollary, clients can omit mustUnderstand="true" from any of the preceding elements with no change in meaning.

R5.4.4-2: If a service cannot comply with a header marked with mustUnderstand="true", it shall issue an s:NotUnderstood fault.

The goal is for the service to be tolerant of inconsistent mustUnderstand usage by clients when the request is not likely to be misinterpreted.

It is important that clients using the WS-Management default addressing model (ResourceURI and SelectorSet) use mustUnderstand="true" on the wsman:ResourceURI element to ensure that the service is compliant with that addressing model. Implementations that use service-specific addressing models will otherwise potentially ignore these header values and behave inconsistently with the intentions of the client.

5.4.5 wsa:To

This clause describes the use of the Addressing wsa:To header regardless of whether an implementation uses WS-Management Addressing (see 5.1) or the W3C Recommendation version of WS-Addressing.

In request messages, the wsa:To address contains the transport address of the service. In some cases, this address is sufficient to locate the resource. In other cases, the service is a dispatching agent for multiple resources. In these cases, the message typically contains additional headers to allow the service to identify a resource within its scope. For example, when the default addressing model is in use, these additional headers will be the ResourceURI and SelectorSet elements.

NOTE: WS-Management does not preclude multiple listener services from coexisting on the same physical system. Such services would be discovered and distinguished using mechanisms beyond the scope of this specification.

R5.4.5-1: The wsa:To header shall be present in all messages, whether requests, responses, or events. In the absence of other requirements, it is recommended that the network address for resources that require authentication be suffixed by the token sequence */wsman*. If */wsman* is used, unauthenticated access should not be allowed.

```
(1) <wsa:To> http://123.15.166.67/wsman </wsa:To>
```

R5.4.5-2: In the absence of other requirements, it is recommended that the network address for resources that do not require authentication be suffixed by the token sequence */wsman-anon*. If */wsman-anon* is used, authenticated access shall not be required.

```
(1) <wsa:To> http://123.15.166.67/wsman-anon </wsa:To>
```

Including the network transport address in the SOAP message may seem redundant because the network connection would already be established by the client. However, in cases where the message is routed through intermediaries, the network transport address is required so that the intermediaries can examine the message and make the connection to the actual endpoint.

The wsa:To header may encompass any number of tokens required to locate the service and a group of resources within that service.

R5.4.5-3: The service should generate a fault when the wsa:To address cannot be processed due to the following situations::

- If the resource is offline, a wsa:EndpointUnavailable fault is returned with the following detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ResourceOffline
- If the resource cannot be located ("not found"), a wsa:DestinationUnreachable fault is returned.
- If the resource is valid, but internal errors occur, a wsman:InternalError fault is returned.
- If the resource cannot be accessed for security reasons, a wsman:AccessDenied fault is returned.

5.4.6 Other Addressing Headers

This clause describes the use of other Addressing headers, regardless of whether an implementation uses WS-Management Addressing (see 5.1) or the W3C Recommendation version of WS-Addressing.

WS-Management depends on Addressing to describe the rules for use of other Addressing headers.

5.4.6.1 Processing Addressing Headers

The following additional addressing-related header blocks occur in WS-Management messages.

R5.4.6.1-1: A conformant service shall recognize and process the following Addressing header blocks.

- **wsa:To**
- **wsa:ReplyTo** (required when a response is expected)
- **wsa:FaultTo** (optional)
- **wsa:MessageID** (required)
- **wsa:Action** (required)
- **wsa:RelatesTo** (required in responses)

The use of these header blocks is discussed in subsequent clauses.

5.4.6.2 wsa:ReplyTo

WS-Management requires the following usage of wsa:ReplyTo in addressing:

R5.4.6.2-1: A wsa:ReplyTo header shall be present in all request messages when a reply is required. This address shall be either a valid address for a new connection using any transport supported by the service or the Addressing Anonymous URI, which indicates that the reply is to be delivered over the same connection on which the request arrived. If the wsa:ReplyTo header is missing, a wsa:MessageInformationHeaderRequired fault is returned.

Some messages, such as event deliveries, SubscriptionEnd, and so on, do not require a response and may omit a wsa:ReplyTo element.

R5.4.6.2-2: A conformant service may require that all responses be delivered over the same connection on which the request arrives. In this case, the URI discussed in R5.4.6.2-1 shall indicate this. Otherwise, the service shall return a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode

R5.4.6.2-3: When delivering events for which acknowledgement of delivery is required, the sender of the event shall include a wsa:ReplyTo element and observe the usage in 10.8 of this specification.

R5.4.6.2-4: This rule intentionally left blank.

R5.4.6.2-5: This rule intentionally left blank.

Addressing allows clients to include client-defined reference parameters in wsa:ReplyTo headers. Addressing requires that these reference parameters be extracted from requests and placed in the responses by removing the ReferenceParameters wrapper and placing all of the values as top-level SOAP headers in the response, as discussed in 5.1. This allows clients to better correlate responses with the original requests. This step cannot be omitted.

EXAMPLE: In the following example, the header x:someHeader is included in the reply message:

```
<s:Envelope
  xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
  <s:Header>
    ...
    <wsa:To> http://1.2.3.4/wsman </wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
      </wsa:Address>
      <wsa:ReferenceParameters>
        <x:someHeader xmlns:x="..."> user-defined content </x:someHeader>
      </wsa:ReferenceParameters>
    </wsa:ReplyTo>
    ...
  </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

R5.4.6.2-6: If the wsa:ReplyTo address is not usable or is missing, the service should not reply to the request and it should close or terminate the connection according to the rules of the current network transport. In these cases, the service should locally log some type of entry to help locate the client defect later.

5.4.6.3 wsa:FaultTo

WS-Management qualifies the use of wsa:FaultTo as indicated in this clause.

R5.4.6.3-1: A conformant service may support a wsa:FaultTo address that is distinct from the wsa:ReplyTo address. If such a request is made and is not supported by the service, a wsman:UnsupportedFeature fault shall be returned with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode

If both the wsa:FaultTo and wsa:ReplyTo headers are omitted from a request, transport-level mechanisms are typically used to fail the request because the address to which the fault is to be sent is uncertain. In such a case, it is not an error for the service to simply shut down the connection.

R5.4.6.3-2: If wsa:FaultTo is omitted, the service shall return the fault to the wsa:ReplyTo address if a fault occurs.

R5.4.6.3-3: A conformant service may require that all faults be delivered to the client over the same transport or connection on which the request arrives. In this case, the URI shall be the Addressing Anonymous URI. If services do not support separately addressed fault delivery and the wsa:FaultTo is any other address, a wsman:UnsupportedFeature fault shall be returned with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode

NOTE: This specification does not restrict richer implementations from fully supporting wsa:FaultTo.

R5.4.6.3-4: This rule intentionally left blank.

EXAMPLE: In the following example, the header x:someHeader is included in fault messages if they occur:

```
(1)  <s:Envelope
(2)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)     xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(5)   <s:Header>
(6)     ...
(7)     <wsa:To> http://1.2.3.4/wsman </wsa:To>
(8)     <wsa:FaultTo>
(9)       <wsa:Address>
(10)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(11)      </wsa:Address>
(12)      <wsa:ReferenceParameters>
(13)        <x:someHeader xmlns:x="..."> user-defined content </x:someHeader>
(14)      </wsa:ReferenceParameters>
(15)    </wsa:FaultTo>
(16)    ...
(17)  </s:Header>
(18)  <s:Body> ... </s:Body>
(19) </s:Envelope>
```

R5.4.6.3-5: If the wsa:FaultTo address is not usable, the service should not reply to the request. Similarly, if according to WS-Addressing processing rules there is no suitable address to send a fault to, it should not reply and should close the network connection. In these cases, the service should locally log some type of entry to help locate the client defect later.

R5.4.6.3-6: The service shall properly duplicate the wsa:Address of the wsa:FaultTo element in the wsa:To of the reply, even if some of the information is not understood by the service.

This rule applies in cases where the client includes private content suffixes on the HTTP or HTTPS address that the service does not understand. If the service removes this information when constructing the address, the subsequent message might not be correctly processed.

5.4.6.4 wsa:MessageID and wsa:RelatesTo

WS-Management qualifies the use of wsa:MessageID and wsa:RelatesTo as follows:

R5.4.6.4-1: The MessageID and RelatesTo URIs may be of any format, as long as they are valid URIs according to RFC 3986. Two URIs are considered different even if the characters in the URIs differ only by case.

The following two formats are endorsed by this specification. The first is considered a best practice because it is backed by RFC 4122:

urn:uuid:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
or
uuid:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

In these formats, each *x* is an uppercase or lowercase hexadecimal digit (lowercase is required by RFC 4122); there are no spaces or other tokens. The value may be a DCE-style universally unique identifier (UUID) with provable uniqueness properties in this format, however, it is not necessary to have provable uniqueness properties in the URIs used in the wsa:MessageID and wsa:RelatesTo headers.

Regardless of format, the URI should not exceed the maximum defined in R13.1-6.

UUIDs have a numeric meaning as well as a string meaning, and this can lead to confusion. A UUID in lowercase is a different URI from the same UUID in uppercase. This is because URIs are case-sensitive. If a UUID is converted to its decimal equivalent the case of the original characters is lost. WS-Management works with the URI value itself, not the underlying decimal equivalent representation. Services are free to *interpret* the URI in any way, but are not allowed to alter the case usage when repeating the message or any of the MessageID values in subsequent messages.

The RFC 4122 requires the digits to be lowercase, which is the responsibility of the client. The service simply processes the values as URI values and is not required to analyze the URI for correctness or compliance. The service replicates the client usage in the wsa:RelatesTo reply header and is not allowed to alter the case usage.

R5.4.6.4-2: The MessageID should be generated according to any algorithm that ensures that no two MessageIDs are repeated. Because the value is treated as case-sensitive (R5.4.6.4-1), confusion can arise if the same value is reused differing only in case. As a result, the service shall not create or employ MessageID values that differ only in case. For any message transmitted by the service, the MessageID shall not be reused.

The client ensures that MessageID values are not reused in requests. Although services and clients can issue different MessageIDs that differ only in case, the service is not required to detect this difference, nor is it required to analyze the URI for syntactic correctness or repeated use.

R5.4.6.4-3: The RelatesTo element shall be present in all response messages and faults, shall contain the MessageID of the associated request message, and shall match the original in case, being treated as a URI value and not as a binary UUID value.

R5.4.6.4-4: If the MessageID is not parsable or is missing, a wsa:InvalidMessageInformationHeader fault should be returned.

EXAMPLE: The following examples show wsa:MessageID usage:

```
(20)  <wsa:MessageID>
(21)    uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91
(22)  </wsa:MessageID>
(23)
(24)  <wsa:MessageID>
(25)    anotherScheme:ID/12310/1231/16607/25
(26)  </wsa:MessageID>
```

5.4.6.5 wsa:Action

The wsa:Action URI indicates the "operation" being invoked against the resource.

R5.4.6.5-1: The wsa:Action URI shall not be used to identify the specific resource class or instance, but only to identify the operation to use against that resource.

R5.4.6.5-2: For all resource endpoints, a service shall return a wsa:ActionNotSupported fault if a requested action is not supported by the service for the specified resource.

In other words, to model the "Get" of item "Disk", the wsa:Action URI contains the "Get". The wsa:To, and potentially other SOAP headers, indicate *what* is being accessed. When the default addressing model is used, for example, the ResourceURI typically contains the reference to the "Disk" and the SelectorSet identifies which disk. Other service-specific addressing models can factor the identity of the resource in different ways.

Implementations are free to support additional custom methods that combine the notion of "Get" and "Disk" into a single "GetDisk" action if they strive to support the separated form to maximize interoperation. One of the main points behind WS-Management is to unify common methods wherever possible.

R5.4.6.5-3: If a service exposes any of the following types of capabilities, a conformant service shall at least expose that capability using the definitions in Table 4 according to the rules of this specification. The service may optionally expose additional similar functionality using a distinct wsa:Action URI.

Table 4 – wsa:Action URI Descriptions

Action URI	Description
http://schemas.xmlsoap.org/ws/2004/09/transfer/Get	Models any simple single item retrieval
http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse	Response to "Get"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Put	Models an update of an entire item
http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse	Response to "Put"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Create	Models creation of a new item
http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse	Response to "Create"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete	Models the deletion of an item
http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse	Response to "Delete"

Action URI	Description
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate	Begins an enumeration or query
http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse	Response to "Enumerate"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull	Retrieves the next batch of results from enumeration
http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse	Response to "Pull"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew	Renews an enumerator that may have timed out (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse	Response to "Renew" (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus	Gets the status of the enumerator (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse	Response to "GetStatus" (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release	Releases an active enumerator
http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse	Response to "Release"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd	Notifies that an enumerator has terminated (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe	Models a subscription to an event source
http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse	Response to "Subscribe"
http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew	Renews a subscription prior to its expiration
http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse	Response to "Renew"
http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus	Requests the status of a subscription
http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse	Response to "GetStatus"
http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe	Removes an active subscription
http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse	Response to "Unsubscribe"
http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd	Delivers a message to indicate that a subscription has terminated
http://schemas.dmtf.org/wbem/wsman/1/wsman/Events	Delivers batched events based on a subscription
http://schemas.dmtf.org/wbem/wsman/1/wsman/Heartbeat	A pseudo-event that models a heartbeat of an active subscription; delivered when no real events are available, but used to indicate that the event subscription and delivery mechanism is still active
http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents	A pseudo-event that indicates that the real event was dropped
http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack	Used by event subscribers to acknowledge receipt of events; allows event streams to be strictly sequenced
http://schemas.dmtf.org/wbem/wsman/1/wsman/Event	Used for a singleton event that does not define its own action

R5.4.6.5-4: A custom action may be supported if the operation is a custom method whose semantic meaning is not present in the table.

R5.4.6.5-5: All notifications shall contain a unique action URI that identifies the type of the event delivery. For singleton notifications with only one event per message (the delivery mode http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push), the wsa:Action URI defines the event type. For other delivery modes, the Action varies, as described in clause 10.2.7 of this specification.

5.4.6.6 wsa:From

The wsa:From header can be used in any messages, responses, or events to indicate the source. When the same connection is used for both request and reply, this header provides no useful information, but can be useful in cases where the response arrives on a different connection.

R5.4.6.6-1: A conformant service may include a wsa:From address in the message. A conformant service should process any incoming message that has a wsa:From element.

R5.4.6.6-2: A conformant service should not fault any message with a wsa:From element, regardless of whether the mustUnderstand attribute is included.

NOTE: Processing the wsa:From header is trivial because it has no effect on the meaning of the message. The *From* address is primarily for auditing and logging purposes.

6 WS-Management Control Headers

WS-Management defines several SOAP headers that can be used with any operation.

6.1 wsman:OperationTimeout

Most management operations are time-critical due to quality-of-service constraints and obligations. If operations cannot be completed in a specified time, the service returns a fault so that a client can comply with its obligations. The following header value can be supplied with any WS-Management message to indicate that the client expects a response or a fault within the specified time:

```
(1) <wsman:OperationTimeout> xs:duration </wsman:OperationTimeout>
```

R6.1-1: All request messages may contain a wsman:OperationTimeout header element that indicates the maximum amount of time the client is willing to wait for the service to issue a response. The service should interpret the timeout countdown as beginning from the point the message is processed until a response is generated.

R6.1-2: The service should *immediately* issue a wsman:TimedOut fault if the countdown time is exceeded and the operation is not yet complete. If the OperationTimeout value is not valid, a wsa:InvalidMessageInformationHeader fault should be returned.

R6.1-3: If the service does not support user-defined timeouts, a wsman:UnsupportedFeature fault should be returned with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OperationTimeout

R6.1-4: If the wsman:OperationTimeout element is omitted, the service may interpret this omission as an instruction to block indefinitely until a response is available, or it may impose a default timeout.

These rules do not preclude services from supporting infinite or very long timeouts. Because network connections seldom block indefinitely with no traffic occurring, some type of transport timeout is likely. Also the countdown is initiated from the time the message is received, so network latency is not included. If a client needs to discover the range of valid timeouts or defaults, metadata can be retrieved, but the format of such metadata is beyond the scope of this specification.

If the timeout occurs in such a manner that the service has already performed some of the work associated with the request, the service state reaches an anomalous condition. This specification does not attempt to address behavior in this situation. Clearly, services can attempt to undo the effects of any partially complete operations, but this is not always practical. In such cases, the service can keep a local log of requests and operations, which the client can query later.

For example, if a Delete operation is in progress and a timeout occurs, the service decides whether to attempt a rollback or roll-forward of the deletion, even though it issues a wsman:TimedOut fault. The service can elect to include additional information in the fault (see 14.5) regarding its internal policy in this regard. The service can attempt to return to the state that existed before the operation was attempted, but this is not always possible.

R6.1-5: If the mustUnderstand attribute is applied to the wsman:OperationTimeout element and the service understands wsman:OperationTimeout, the service shall observe the requested value or return the fault specified in R6.1-2. The service should attempt to complete the request within the specified time or issue a fault without any further delay.

Clients can always omit the mustUnderstand header for uniform behavior against all implementations. It is not an error for a compliant service to ignore the timeout value or treat it as a hint if mustUnderstand is omitted.

EXAMPLE: The following is an example of a correctly formatted 30-second timeout in the SOAP header:

```
(1)  <wsman:OperationTimeout>PT30S</wsman:OperationTimeout>
```

If the transport timeout occurs before the actual wsman:OperationTimeout, the operation can be treated as specified in 13.3, the same as a failed connection. In practice, the network transport timeout can be configured to be longer than any expected wsman:OperationTimeout.

6.2 wsman:MaxEnvelopeSize

To prevent a response beyond the capability of the client, the request message can contain a restriction on the response size.

The following header value may be supplied with any WS-Management message to indicate that the client expects a response whose total SOAP envelope does not exceed the specified number of octets:

```
(1)  <wsman:MaxEnvelopeSize> xs:positiveInteger </wsman:MaxEnvelopeSize>
```

The limitation is on the entire envelope. Resource-constrained implementations need a reliable figure for the required amount of memory for all SOAP processing, not just the SOAP Body.

R6.2-1: All request messages may contain a wsman:MaxEnvelopeSize header element that indicates the maximum number of octets (not characters) in the entire SOAP envelope in the response. If the service cannot compose a reply within the requested size, it should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize

R6.2-2: If the mustUnderstand attribute is set to “true”, the service shall comply with the request. If the response would exceed the maximum size, the service should return a wsman:EncodingLimit fault. Because a service might execute the operation prior to knowing the response size, the service should undo any effects of the operation before issuing the fault. If the operation cannot be reversed (such as a destructive Put or Delete, or a Create), the service shall indicate that the operation succeeded in the wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess

R6.2-3: If the mustUnderstand attribute is set to “false”, the service may ignore the header.

R6.2-4: Services should reject any MaxEnvelopeSize value less than 8192 octets. This number is the safe minimum in which faults can be reliably encoded for all character sets. If the requested size is less than this, the service should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MinimumEnvelopeLimit

A service might have its own encoding limit independent of what the client specifies, and the same fault applies.

R6.2-5: If the service cannot compose a reply within its own internal limits, the service should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit

The definition of the wsman:MaxEnvelopeSize element in the schema contains a Policy attribute because this element is used for other purposes. This specification does not define a meaning for the Policy attribute when the wsman:MaxEnvelopeSize element is used as a SOAP header.

R6.2-6: Clients should not add the Policy attribute to the wsman:MaxEnvelopeSize element when it is used as a SOAP header. Services should ignore the Policy attribute if it appears in the wsman:MaxEnvelopeSize element when used as a SOAP header.

6.3 wsman:Locale

Management operations often span locales, and many items in responses can require translation. Typically, translation is required for descriptive information, intended for human readers, that is sent back in the response. If the client requires such output to be translated into a specific language, it can employ the optional wsman:Locale header, which makes use of the standard XML attribute xml:lang, as follows:

```
(1)  <wsman:Locale xml:lang="xs:language" s:mustUnderstand="false"/>
```

R6.3-1: If the mustUnderstand attribute is omitted or set to “false”, the service should use this value when composing the response message and adjust any localizable values accordingly. This use is recommended for most cases. The locale is treated as a hint in this case.

R6.3-2: If the mustUnderstand attribute is set to “true”, the service shall ensure that the replies contain localized information where appropriate, or else the service shall issue a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale

A service may always fault if wsman:Locale contains s:mustUnderstand set to “true”, because it may not be able to ensure that the reply is localized.

Some implementations delegate the request to another subsystem for processing, so the service cannot be certain that the localization actually occurred.

R6.3-3: The value of the xml:lang attribute in the wsman:Locale header shall be a valid RFC 5646 language code.

R6.3-4: In any response, event, or singleton message, the service should include the xml:lang attribute in the s:Envelope (or other elements) to signal to the receiver that localized content appears in the body of the message. This attribute may be omitted if no descriptive content appears in the body. Including the xml:lang attribute is not an error, even if no descriptive content occurs.

EXAMPLE:

```
<s:Envelope
    xml:lang="en-us"
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
  <s:Header> ... </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

The xml:lang attribute can appear on any content in the message, although a simpler approach allows the client always to check for the attribute in one place, the s:Envelope wrapper.

R6.3-5: For operations that span multiple message sequences, the wsman:Locale element is processed in the initial message only. It should be ignored in subsequent messages because the first message establishes the required locale. The service may issue a fault if the wsman:Locale is present in subsequent messages and the value is different from that used in the initiating request.

This rule applies primarily to Enumerate and Pull messages. The locale is clearly established during the initial Enumerate request, so changing the locale during the enumeration serves no purpose. The service ignores any wsman:Locale elements in subsequent Pull messages, but the client can ensure that the value does not change between Pull requests. This uniformity enables the client to construct messages more easily.

It is recommended (as established in R6.3-1) that the wsman:Locale element never contain a mustUnderstand attribute. In this way, the client will not receive faults in unexpected places.

6.4 wsman:OptionSet

The OptionSet header is used to pass a set of switches to the service to modify or refine the nature of the request. This facility is intended to help the service observe any context or side effects desired by the client, but *not* to alter the output schema or modify the meaning of the addressing. Options are similar to switches used in command-line shells in that they are service-specific, text-based extensions.

R6.4-1: Any request message may contain a wsman:OptionSet header, which wraps a set of optional switches or controls on the message. These switches help the service compose the desired reply or observe the required side effect.

R6.4-2: The service should not send responses, unacknowledged events, or singleton messages that contain wsman:OptionSet headers unless it is acting in the role of a client to another service. Those headers are intended for request messages to which a subsequent response is expected, including acknowledged events.

R6.4-3: If the mustUnderstand attribute is omitted from the OptionSet block or if it is present with a value of “false”, the service may ignore the entire wsman:OptionSet block. If it is present with a value of “true” and the service does not support wsman:OptionSet, the service shall return a s:NotUnderstood fault.

Services can process an OptionSet block if it is present, but they are not required to understand or process individual options, as shown in R6.4-6. However, if MustComply is set to “true” on any given option, then mustUnderstand needs to be set to "true". Doing so avoids the incongruity of allowing the entire OptionSet block to be ignored while having MustComply on individual options.

R6.4-4: Each resource class may observe its own set of options, and an individual instance of that resource class may further observe its own set of options. Consistent option usage is not required across resource class and instance boundaries. The metadata formats and definitions of options are beyond the scope of this specification and may be service-specific.

R6.4-5: Any number of individual option elements may appear under the wsman:OptionSet wrapper. Option names may be repeated if appropriate. The content shall be a simple string (xs:string). This specification places no restrictions on whether the names or values are to be treated in a case-sensitive or case-insensitive manner. However, case usage shall be retained as the message containing the OptionSet element and its contents are propagated through SOAP intermediaries.

Interpretation of the option with regard to case sensitivity is up to the service and the definition of the specific option because the value might be passed through to real-world subsystems that inconsistently expose case usage. Where interoperation is a concern, the client can omit both mustUnderstand and MustComply attributes.

R6.4-6: Individual option values may be advisory or may be required by the client. The service shall observe and execute any option marked with the MustComply attribute set to "true", or return a wsman:InvalidOptions fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported

Any option not marked with this attribute (or if the attribute is set to "false") is advisory to the service, and the service may ignore it. If any option is marked with MustComply set to "true", then the mustUnderstand attribute shall be used on the entire wsman:OptionSet block.

This capability is required when the service delegates interpretation and execution of the options to another component. In many cases, the SOAP processor cannot know if the option was observed and can only pass it along to the next subsystem.

R6.4-7: Options may optionally contain a Type attribute, which indicates the data type of the content of the Option element. A service may require that this attribute be present on any given option and that it be set to the QName of a valid XML schema data type. Only the standard simple types declared in the http://www.w3.org/2001/XMLSchema namespace are supported in this version of WS-Management.

This rule can help some services distinguish numeric or date/time types from other string values.

R6.4-8: Options should not be used as a replacement for the documented parameterization technique for the message; they should be used only as a modifier for it.

Options are primarily used to establish context or otherwise instruct the service to perform side-band operations while performing the operation, such as turning on logging or tracing.

R6.4-9: The following faults should be returned by the service:

- when options are not supported, **wsman:InvalidOptions** with the following detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported

- when one or more option names are not valid or supported by the specific resource, **wsman:InvalidOptions** with the following detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName

- when the value is not correct for the option name, **wsman:InvalidOptions** with the following detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue

R6.4-10: For operations that span multiple message sequences, the wsman:OptionSet element is processed in the initial message only. It should be ignored in subsequent messages because the first message establishes the required set of options. The service may issue a fault if the wsman:OptionSet is present in subsequent messages and the value is different from that used in the initiating request, or the service may ignore the values of wsman:OptionSet in such messages.

This rule applies primarily to Enumerate and Pull messages. The set of options is established once during the initial Enumerate request, so changing the options during the enumeration would constitute an error.

Options are intended to make operations more efficient or to preprocess output on behalf of the client. For example, the options could indicate to the service that the returned values are to be recomputed and that cached values are not to be used, or that any optional values in the reply may be omitted. Alternately, the options could be used to indicate verbose output within the limits of the XML schema associated with the reply.

Option values are not intended to contain XML. If XML-based input is required, a custom operation with its own wsa:Action URI is the correct model for the operation. This ensures that no backdoor parameters are introduced over well-known message types. For example, when issuing a Subscribe request, the message already defines a technique for passing an event filter to the service, so the option is not used to circumvent this and pass a filter using an alternate method.

EXAMPLE: The following is an example of wsman:OptionSet:

```
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <s:Header>
    ...
    <wsman:OptionSet s:mustUnderstand="true">
      <wsman:Option Name="VerbosityLevel" Type="xs:int">
        3
      </wsman:Option>
      <wsman:Option Name="LogAllRequests" MustComply ="true"/>
    </wsman:OptionSet>
    ...
  </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsman:OptionSet

  used to wrap individual option blocks

  In this example, s:mustUnderstand is set to "true", indicating that the client is requiring the service to process the option block using the given rules.

wsman:OptionSet/wsman:Option/@Name

identifies the option (an xs:string), which may be a simple name or a URI

This name is scoped to the resource to which it applies. The name may be repeated in subsequent elements. The name cannot be blank and can be a short non-colliding URI that is vendor-specific.

wsman:OptionSet/wsman:Option/@MustComply

if set to "true", indicates that the option shall be observed; otherwise, indicates an advisory or a hint

wsman:OptionSet/wsman:Option/@Type

(optional) if present, indicates the data type of the element content, which helps the service to interpret the content

A service may require this attribute to be present on any given option element.

wsman:OptionSet/wsman:Option

the content of the option

The value may be any simple string value. If the option value is empty, the option should be interpreted as logically "true", and the option should be "enabled". The following example enables the "Verbose" option:

```
(1)  <wsman:Option Name="Verbose"/>
```

Options are logically false if they are not present in the message. All other cases require an explicit string to indicate the option value. The reasoning for allowing the same option to repeat is to allow specification of a list of options of the same name.

6.5 wsman:RequestEPR

Some service operations, including "Put", are able to modify the resource representation in such a way that the update results in a logical identity change for the resource, such as the "rename" of a document. In many cases, this modification in turn alters the EPR of that resource after the operation is completed, as EPRs are often dynamically derived from naming values within the resource representation itself. This behavior is common in SOAP implementations that delegate operations to underlying systems.

To provide the client a way to determine when such a change has happened, two SOAP headers are defined to request and return the EPR of a resource instance.

In any WS-Management request message, the following header may appear:

```
(1) <wsman:RequestEPR .../>
```

R6.5-1: A service receiving a message that contains the wsman:RequestEPR header block should return a response that contains a wsman:RequestedEPR header block. This block contains the most recent EPR of the resource being accessed or a status code if the service cannot determine or return the EPR. This EPR reflects any identity changes that may have occurred as a result of the current operation, as set forth in the following behavior. The header block in the corresponding response message has the following format:

```
(1)  <wsman:RequestedEPR ...>
(2)    [ <wsa:EndpointReference>
(3)        wsa:EndpointReferenceType
(4)    </wsa:EndpointReference> |
(5)    <wsman:EPRInvalid/> |
(6)    <wsman:EPRUnknown/> ]
(7)  </wsman:RequestedEPR>
```

The following definitions describe additional, normative constraints on the preceding format:

wsman:RequestedEPR/wsa:EndpointReference

one of three elements that can be returned as a child element of the wsman:RequestedEPR element

The use of this element indicates that the service understood the request to return the EPR of the resource and is including the EPR of the resource. The returned EPR is calculated after all intentional effects or side effects of the associated request message have occurred. The EPR may not have changed as a result of the operation, but the service is still obligated to return it.

wsman:RequestedEPR/wsman:EPRInvalid

one of three elements that can be returned as a child element of the wsman:RequestedEPR element

The use of this element (no value is required) indicates that the service understands the request to return the EPR of the resource but is unable to calculate a full EPR. However, the service is able to determine that this message exchange has modified the resource representation in such a way that any previous references to the resource are no longer valid. When EPRInvalid is returned, the client shall not use the old wsa:EndpointReference in subsequent operations.

wsman:RequestedEPR/wsman:EPRUnknown

one of three elements that can be returned as a child element of the wsman:RequestedEPR element

The use of this element (no value is required) indicates that the service understands the request to return the EPR of the resource but is unable to determine whether existing references to the resource are still valid. When EPRUnknown is returned, the client may attempt to use the old wsa:EndpointReference in subsequent operations. The result of using an old wsa:EndpointReference, however, is unpredictable; a result may be a fault or a successful response.

7 Resource Access

7.1 General

Resource access applies to all synchronous operations regarding getting, setting, and enumerating values. The subclauses in clause 7 define a mechanism for acquiring management-specific XML-based representations of entities using the Web service infrastructure, such as managed resources.

Specifically, two operations are defined for sending and receiving the management representation of a given resource and two operations are defined for creating and deleting a management resource and its corresponding representation. Multi-instance retrieval is achieved using the enumeration messages. This specification does not define any messages or techniques for batched operations, such as batched Get or Delete. All such operations can be sent as a series of single messages.

It should be noted that the state maintenance of a resource is at most subject to the "best efforts" of the hosting server. When a client receives the server's acceptance of a request to create or update a resource, it can reasonably expect that the resource now exists at the confirmed location and with the confirmed representation, but this is not a guarantee, even in the absence of any third parties. The server may change the representation of a resource, may remove a resource entirely, or may bring back a resource that was deleted.

For instance, the server may store resource state information on a disk drive. If that drive crashes and the server recovers state information from a backup tape, changes that occurred after the backup was made would be lost.

A server may have other operational processes that change resource state information. A server may run a background process that examines resources for objectionable content and deletes any such resources it finds. A server may purge resources that have not been accessed for some period of time. A server may apply storage quotas that cause it to occasionally purge resources.

In essence, the confirmation by a service of having processed a request to create, modify, or delete a resource implies a commitment only at the instant that the confirmation was generated. While the usual case should be that resources are long-lived and stable, there are no guarantees, and clients should code defensively.

There is no requirement for uniformity in resource representations between the messages defined in this specification. For example, the representations required by Create or Put may differ from the representation returned by Get, depending on the semantic requirements of the service. Additionally, there is no requirement that the resource content is fixed for any given endpoint reference. The resource content may vary based on environmental factors, such as the security context, time of day, configuration, or the dynamic state of the service.

As per the SOAP processing model, other specifications may define SOAP headers that may be optionally added to request messages to require the transfer of subsets or the application of transformations of the resource associated with the endpoint reference. When the Action URIs defined by this specification are used, such extension specifications must also allow the basic processing models defined herein.

NOTE: The WSDL for the resource access operations (see ANNEX G), as well as the pseudo schema and example message fragments throughout clause 7, is not usable as represented without first replacing the "*resource-specific-GED*" text with the application-defined GED.

EXAMPLE 1: Following is a full example of a hypothetical Get request:

```
(1)  <s:Envelope
(2)      xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)      xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(5)    <s:Header>
(6)      <wsa:To>http://1.2.3.4/wsman/</wsa:To>
(7)      <wsman:ResourceURI>http://example.org/2005/02/physicalDisk
           </wsman:ResourceURI>
(8)      <wsa:ReplyTo>
(9)        <wsa:Address>
(10)         http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(11)       </wsa:Address>
(12)     </wsa:ReplyTo>
(13)     <wsa:Action>
(14)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
(15)     </wsa:Action>
(16)     <wsa:MessageID>
(17)       urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(18)     </wsa:MessageID>
(19)     <wsman:SelectorSet>
(20)       <wsman:Selector Name="LUN"> 2 </wsman:Selector>
(21)     </wsman:SelectorSet>
(22)     <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
(23)   </s:Header>
(24)   <s:Body/>
(25) </s:Envelope>
```

Notice that the wsa:ReplyTo indicates the response is to be sent on the same connection as the request (line 10), the action is a Get (line 14), and the ResourceURI (line 7) and wsman:SelectorSet (line 20) are used to address the requested management information. This example assumes that the

WS-Management default addressing model is in use. The service is expected to complete the operation in 30 seconds or return a fault to the client (line 22).

Also, the s:Body in a Get request has no content.

EXAMPLE 1 (continued): The following shows a hypothetical response to the preceding hypothetical Get request:

```
(26) <s:Envelope
(27)      xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(28)      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(29)      xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(30)    <s:Header>
(31)      <wsa:To>
(32)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(33)      </wsa:To>
(34)      <wsa:Action s:mustUnderstand="true">
(35)      http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(36)    </wsa:Action>
(37)    <wsa:MessageID s:mustUnderstand="true">
(38)      urn:uuid:217a431c-b071-3301-9bb8-5f538bec89b8
(39)    </wsa:MessageID>
(40)    <wsa:RelatesTo>
(41)      urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(42)    </wsa:RelatesTo>
(43)    </s:Header>
(44)    <s:Body>
(45)      <PhysicalDisk
            xmlns="http://schemas.example.org/2005/02/samples/physDisk">
(46)        <Manufacturer> Acme, Inc. </Manufacturer>
(47)        <Model> 123-SCSI 42 GB Drive </Model>
(48)        <LUN> 2 </LUN>
(49)        <Cylinders> 16384 </Cylinders>
(50)        <Heads> 80 </Heads>
(51)        <Sectors> 63 </Sectors>
(52)        <OctetsPerSector> 512 </OctetsPerSector>
(53)        <BootPartition> 0 </BootPartition>
(54)      </PhysicalDisk>
(55)    </s:Body>
(56) </s:Envelope>
```

Notice that the response uses the wsa:To address (line 32) that the original request had specified in wsa:ReplyTo. Also, the wsa:MessageID for this response is unique (line 38). The wsa:RelatesTo (line 41) contains the UUID of the wsa:MessageID of the original request to allow the client to correlate the response.

The s:Body (lines 44-55) contains the requested resource representation.

The same general approach exists for Delete, except that no content exists in the response s:Body. The Create and Put operations are similar, except that they contain content in the request s:Body to specify the values being created or updated.

7.2 Addressing Uniformity

Where practical, the EPR of the resource can be the same whether a Get, Delete, or Put operation is being used. This is not a strict requirement, but it reduces the education and training required to construct and use WS-Management-aware tools.

Create is a special case, in that the EPR of the newly created resource is often not known until the resource is actually created. For example, although it might be possible to return running process information using a hypothetical *ProcessID* in an addressing header, it is typically not possible to assert the *ProcessID* during the creation phase because the underlying system does not support the concept. Thus, the Create operation would not have the same addressing headers as the corresponding Get or Delete operations.

If the WS-Management default addressing model is in use, it would be typical to use the ResourceURI as a "type" and selector values for "instance" identification. Thus, the same address would be used for Get, Put, and Delete when working with the same instance. When enumerating all instances, the selectors would be omitted and the ResourceURI would be used alone to indicate the "type" of the object being enumerated. The Create operation might also share this usage, or have its own ResourceURI and selector usage (or not even use selectors). This pattern is not a requirement.

Throughout, it is expected that the s:Body of the messages contains XML with correct and valid XML namespaces referring to XML Schemas that can validate the message. Most services and clients do not perform real-time validation of messages in production environments because of performance constraints; however, during debugging or other systems verification, validation might be enabled, and messages without the appropriate XML namespace declarations would be considered invalid.

When performing resource access operations, side effects might occur. For example, deletion of a particular resource by using Delete can result in several other dependent instances disappearing, and a Create operation can result in the logical creation of more than one resource that can be subsequently returned through a Get operation. Similarly, a Put operation can result in a rename of the target instance, a rename of some unrelated instance, or the deletion of some unrelated instance. These side effects are service specific, and this specification makes no statements about the taxonomy and semantics of objects over which these operations apply.

7.3 Get

A Web service operation (Get) is defined for fetching a one-time snapshot of the representation of a resource. A snapshot is a complete XML representation of a resource at the time the service processes the request.

The Get request message shall be of the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
(5)      </wsa:Action>
(6)      <wsa:MessageID>xs:anyURI</wsa:MessageID>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body .../>
(11) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/Get. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

A Get request shall be targeted at the resource whose representation is desired.

There are no body blocks defined by default for a Get Request. As per the SOAP processing model, other specifications may introduce various types of extensions to the semantics of this message that are enabled through headers tagged with s:mustUnderstand="true". Such extensions may define how resource or subsets of it are to be retrieved or transformed prior to retrieval. Specifications that define such extensions shall allow processing the basic Get request message without those extensions. Because the response may not be sent to the original sender, extension specifications should consider adding a corresponding SOAP header value in the response to signal to the receiver that the extension is being used.

Implementations may respond with a fault message using the standard fault codes defined in Addressing (for example, wsa:ActionNotSupported). Other components of the preceding outline are not further constrained by this specification.

If the resource accepts a Get request, it shall reply with a response of the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(5)      </wsa:Action>
(6)      <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body …>
(11)     resource-specific-element
(12)   </s:Body>
(13) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/child

The representation itself shall be the child element of the SOAP:Body element of the response message.

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

The Get operation retrieves resource representations. The message can be targeted to return a complex XML document or to return a single, simple value. The nature and complexity of the representation is not constrained by this specification.

R7.3-1: A conformant service should support Get operations to service metadata requests about the service itself or to verify the result of a previous action or operation.

This statement does not constrain implementations from supplying additional similar methods for resource and metadata retrieval.

R7.3-2: Execution of Get should not in itself have side effects on the value of the resource.

R7.3-3: If an object cannot be retrieved due to locking conditions, simultaneous access, or similar conflicts, a wsman:Concurrency fault should be returned.

In practice, Get is designed to return XML that corresponds to real-world objects. To retrieve individual property values, either the client can postprocess the XML content for the desired value, or the service can support fragment-level access (7.7).

Fault usage is generally as described in clause 14. An inability to locate or access the resource is equivalent to problems with the SOAP message when the EPR is defective. There are no "Get-specific" faults.

7.4 Put

A Web service operation (Put) is defined for updating a resource by providing a replacement representation. A resource may accept updates that provide different XML representations than that returned by the resource; in such a case, the semantics of the update operation is defined by the resource.

The Put request message shall be of the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
(5)      </wsa:Action>
(6)      <wsa:MessageID>xs:anyURI</wsa:MessageID>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body…>
(11)     resource-specific-element
(12)   </s:Body>
(13) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/Put. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/child

The representation to be used for the update shall be the child element of the s:Body element of the request message.

A Put request shall be targeted at the resource whose representation is desired to be replaced. As per the SOAP processing model, other specifications may introduce various types of extensions to this message, which are enabled through headers tagged with s:mustUnderstand="true". Such extensions may require that a full or partial update should be accomplished using symbolic, instruction-based, or other methodologies.

Extension specifications may also define extensions to the original Put request, enabled by optional SOAP headers, which control the nature of the response (see the information about PutResponse later in this clause).

Specifications that define any of these extensions shall allow processing of the Put message without such extensions.

In addition to the standard fault codes defined in Addressing, implementations may use the fault code wsmt:InvalidRepresentation if the presented representation is invalid for the target resource. Other components of the preceding outline are not further constrained by this specification.

A successful Put operation updates the current representation associated with the targeted resource.

If the resource accepts a Put request and performs the requested update, it shall reply with a response of the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse
(5)      </wsa:Action>
(6)      <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body …>
(11)     resource-specific-element ?
(12)   </s:Body>
(13) </s:Envelope>
```

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/child

An implementation of a service shall choose, in advance, whether to return an empty Body or the resulting representation of the resource. This choice shall be explicitly stated in the WSDL, if WSDL is provided.

By default, a service shall return the current representation of the resource as the child of the s:Body element if the updated representation differs from the representation sent in the Put request message.

As an optimization and as a service to the requester, the s:Body element of the response message should be empty if the updated representation does not differ from the representation sent in the Put request message; that is, if the service accepted the new representation verbatim.

Such a response (an empty s:Body) implies that the update request was successful in its entirety (assuming no intervening mutating operations are performed). A service may return the current representation of the resource as the initial child of the s:Body element even in this case, however.

Extension specifications may define extensions to the original Put request, enabled by optional header values, in order to optimize the response. In the absence of such headers, the behavior shall be as previously described. Specifications that define any of these extensions shall allow processing the Put message without such extensions. Because the response may not be sent to the original sender, extension specifications should consider adding a corresponding SOAP header value in the response to signal to the receiver that the extension is being used.

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

If a resource can be updated in its entirety within the constraints of the corresponding XML schema for the resource, the service can support the Put operation.

R7.4-1: A conformant service may support Put.

R7.4-2: If a single resource instance can be updated (within the constraints of its schema) by using a SOAP message, and that resource subsequently can be retrieved using Get, a service should support updating the resource by using Put. The service may additionally export a custom method for updates.

R7.4-3: If a single resource instance contains a mix of modifiable and non-modifiable properties, the Put message may contain values for both the modifiable and non-modifiable properties if the XML content is legal with regard to its XML schema namespace. If the Put message contains values for modifiable properties, the service shall set these properties to these values during the Put operation. If the Put message contains values for non-modifiable properties, the service should ignore those values during the Put operation. If none of the properties are modifiable, the service should return a wsa:ActionNotSupported fault.

This situation typically happens if a Get operation is performed, a value is altered, and the entire updated representation is sent using Put. In this case, any read-only values would still be present.

A complication arises because Put contains the complete new representation for the instance. If the resource schema requires the presence of any given value (minOccurs is not zero), it will be supplied as part of the Put message, even if it is not being altered from its original value.

R7.4-4: If a Put operation specifies a modifiable value as NULL using the xsi:nil attribute, then the service shall set the value to NULL.

If the schema definition includes elements that are optional (minOccurs=0), the Put message can omit these values. Existing implementations provide two different responses when these elements are modifiable (writeable). They either set the omitted element's value to NULL or leave the value unchanged. Given this reality, the following rules apply:

R7.4-5: Any modifiable properties that are optional in the XML schema (that is, minOccurs=”0”) and that are are omitted from the Put message shall either be set to a resource-specific default value or be left unchanged. Setting to a resource specific default value is recommended.

NOTE 1: Elements not set may have their value changed as a result of other constraints.

NOTE 2: The resource-specific default value is outside the scope of this specification.

To update isolated values without having to supply all values, use the fragment-level resource access mechanism described in 7.7.

In short, the s:Body of the Put message complies with the constraints of the associated XML schema.

EXAMPLE 1: For example, assume that Get returns the following information:

```
<s:Body>
 <MyObject xmlns="examples.org/2005/02/MySchema">
  <A> 100 </A>
  <B> 200 </B>
  <C> 100 </C>
 </MyObject>
</s:Body>
```

EXAMPLE 2: The corresponding XML schema has defined A, B, and C as minOccurs=1:

```
<xs:element name="MyObjecct">
 <xs:complexType>
   <xs:sequence>
    <xs:element name="A" type="xs:int" minOccurs="1" maxOccurs="1"/>
    <xs:element name="B" type="xs:int" minOccurs="1" maxOccurs="1"/>
    <xs:element name="C" type="xs:int" minOccurs="1" maxOccurs="1"/>
    ...
   </xs:sequence>
  </xs:complexType>
</xs :element>
```

In this case, the corresponding Put needs to contain all three elements because the schema mandates that all three be present. Even if the only value being updated is <B>, the client has to supply all three values. This usually means that the client first has to issue a Get to preserve the current values of <A> and <C>, change <B> to the desired value, and then write the object using Put. As noted in R7.4-3, the service can ignore attempts to update values that are read-only with regard to the underlying real-world object.

R7.4-6: A conformant service should support Put using the same EPR as a corresponding Get or other messages, unless the Put mechanism for a resource is semantically distinct.

R7.4-7: If the supplied Body does not have the correct content to update the resource, the service should return a wsmt:InvalidRepresentation fault and detail codes as follows:

- if any values in the s:Body are not correct:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues

- if any values in the s:Body are missing:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues

- if the wrong XML schema namespace is used and is not recognized by the service:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace

R7.4-8: If an object cannot be updated because of locking conditions, simultaneous access, or similar conflicts, the service should return a wsman:Concurrency fault.

R7.4-9: A Put operation may result in a change to the EPR for the resource because the values being updated may in turn cause an identity change.

Because WS-Management services typically delegate the Put to underlying subsystems, the service might not always be aware of an identity change. Clients can make use of the mechanism in 6.5 to be informed of EPR changes that may have occurred as a side effect of executing a Put operation.

R7.4-10: It is recommended that the service return the new representation in the Put response in all cases. Knowing whether the actual resulting representation is different from the requested update is often difficult because resource-constrained implementations may have insufficient resources to determine the equivalence of the requested update with the actual resulting representation.

The implication of this rule is that if the new representation is not returned, it precisely matches what was submitted in the Put message. Because implementations can rarely assure this, they can always return the new representation.

R7.4-11: If the success of an operation cannot be reported as described in this clause because of encoding limits or other reasons, and it cannot be reversed, the service should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess

R7.4-12: The Put operation may contain updates of multiple values. The service shall successfully carry out an update of all the specified values or return the fault that was the cause of the error. If any fault is returned, the implication is that 0…*n*-1 values were updated out of *n* possible update values.

7.5 Delete

This specification defines one Web service operation (Delete) for deleting a resource in its entirety.

Extension specifications may define extensions to the Delete request, enabled by optional header values, which specifically control preconditions for the Delete to succeed and which may control the nature or format of the response. Because the response may not be sent to the original sender, extension specifications should consider adding a corresponding SOAP header value in the response to signal to the receiver that the extension is being used.

The Delete request message shall be of the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
(5)      </wsa:Action>
(6)      <wsa:MessageID>xs:anyURI</wsa:MessageID>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body … />
(11) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

A Delete request shall be targeted at the resource to be deleted.

There are no body blocks defined for a Delete Request.

Implementations may respond with a fault message using the standard fault codes defined in Addressing (for example, wsa:ActionNotSupported). Other components of the preceding outline are not further constrained by this specification.

A successful Delete operation invalidates the current representation associated with the targeted resource.

If the resource accepts a Delete request, it shall reply with a response of the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse
(5)      </wsa:Action>
(6)      <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body .../>
(11) </s:Envelope>
```

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

By default, there are no s:Body blocks defined for a Delete response. Specifications that define extensions for use in the original Delete request that control the format of the response shall allow processing the Delete message without such extensions.

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

In general, the addressing can be the same as for a corresponding Get operation for uniformity, but this is not absolutely required.

R7.5-1: A conformant service may support Delete.

R7.5-2: A conformant service should support Delete using the same EPR as a corresponding Get or other messages, unless the deletion mechanism for a resource is semantically distinct.

R7.5-3: If deletion is supported and the corresponding resource can be retrieved using Get, a conformant service should support deletion using Delete. The service may additionally export a custom action for deletion.

R7.5-4: If an object cannot be deleted due to locking conditions, simultaneous access, or similar conflicts, a wsman:Concurrency fault should be returned.

In practice, Delete removes the resource instance from the visibility of the client and is a *logical* deletion.

The operation might result in an actual deletion, such as removal of a row from a database table, or it might simulate deletion by unbinding the representation from the real-world object. Deletion of a "printer," for example, does not result in literal annihilation of the printer, but simply removes it from the access scope of the service, or "unbinds" it from naming tables. WS-Management makes no distinction between literal deletions and logical deletions.

To delete individual property values within an object that, itself, is not to be deleted, either the client can perform a Put, according to section 7.4 or the service can support fragment-level delete (7.7).

Fault usage is generally as described in clause 14. Inability to locate or access the resource is equivalent to problems with the SOAP message when the EPR is defective. There are no "Delete-specific" faults.

7.6 Create

A Web service operation (Create) is defined for creating a resource and providing its initial representation. In some cases, the initial representation may constitute the representation of a logical constructor for the resource and may thus differ structurally from the representation returned by Get or the one required by Put. This difference is because the parameterization requirement for creating a resource is often distinct from the steady-state representation of the resource. Implementations should provide metadata that describes the use of the representation and how it relates to the resource which is created, but such mechanisms are beyond the scope of this specification. The resource factory that receives a Create request allocates a new resource that is initialized from the presented representation. The new resource is assigned a service-determined endpoint reference that is returned in the response message.

The Create request message shall be of the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
(5)      </wsa:Action>
(6)      <wsa:MessageID>xs:anyURI</wsa:MessageID>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body …>
(11)     resource-specific-element
(12)   </s:Body>
(13) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/Create. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/child

The child element of the s:Body element shall not be omitted. The contents of this element are service-specific, and may contain the literal initial resource representation, a representation of the constructor for the resource, or other instructions for creating the resource.

Extension specifications may also define extensions to the original Create request, enabled by optional SOAP headers, which constrain the nature of the response (see information about the CreateResponse later in this clause). Similarly, they may require headers that control the interpretation of the s:Body as part of the resource creation process.

Such specifications shall also allow processing the Create message without such extensions.

A Create request shall be targeted at a resource factory capable of creating the desired new resource. This factory is distinct from the resource being created (which by definition does not exist prior to the successful processing of the Create request message).

In addition to the standard fault codes defined in Addressing, implementations may use the fault code wsmt:InvalidRepresentation if the presented representation is invalid for the target resource.

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

If the resource factory accepts a Create request, it shall reply with a response of the following form:

```
(1) <s:Envelope …>
(2)   <s:Header …>
(3)     <wsa:Action>
(4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
(5)     </wsa:Action>
(6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
(7)     <wsa:To>xs:anyURI</wsa:To>
(8)     …
(9)   </s:Header>
(10)  <s:Body …>
(11)    <wsmt:ResourceCreated>endpoint-reference</wsmt:ResourceCreated>
(12)  </s:Body>
(13)</s:Envelope>
```

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse. If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/wsmt:ResourceCreated

This required element shall contain a resource reference for the newly created resource. This resource reference, represented as an endpoint reference as defined in Addressing, shall identify the resource for future Get, Put, and Delete operations.

Extension specifications may define extensions to the original Create request, enabled by optional header values. These headers may override the default behavior if they are marked with s:mustUnderstand="true". In the absence of such optional headers, the behavior shall be as described in the previous paragraphs. Because the response may not be sent to the original sender, extension specifications should consider adding a corresponding SOAP header value in the response to signal to the receiver that the extension is being used.

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

In general, the addressing is not the same as that used for Get or Delete in that the EPR assigned to a newly created instance for subsequent access is not necessarily part of the XML content used for creating the resource. Because the EPR is usually assigned by the service or one of its underlying systems, the CreateResponse contains the applicable EPR of the newly created instance.

R7.6-1: A conformant service may support Create.

R7.6-2: If a single resource can be created using a SOAP message and that resource can be subsequently retrieved using Get, then a service should support creation of the resource using Create. The service may additionally export a custom method for instance creation.

R7.6-3: If the supplied SOAP Body does not have the correct content for the resource to be created, the service should return a wsmt:InvalidRepresentation fault and detail codes as follows:

- if one or more values in the <s:Body> were not correct:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues

- if one or more values in the <s:Body> were missing:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues

- if the wrong XML schema namespace was used and is not recognized by the service:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace

R7.6-4: A service shall not use Create to modify the value of an existing representation (except as specified in 7.11). If the targeted object already exists, the service should return a wsman:AlreadyExists fault.

The message body for Create is not required to use the same schema as that returned with a Get operation for the resource. Often, the values required to create a resource are different from those retrieved using a Get operation or those used for updates with a Put operation.

If a service needs to support creation of individual values within a representation (fragment-level creation, array insertion, and so on), it can support fragment-level access (7.7).

R7.6-5: The response to a Create message shall contain the new EPR of the created resource in the ResourceCreated element.

R7.6-6: This rule intentionally left blank.

EXAMPLE: The following is a hypothetical example of a response for a newly created virtual drive:

```
<s:Envelope
   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
   xmlns:wsmt="http://schemas.xmlsoap.org/ws/2004/09/transfer">
  <s:Header>
   ...
   <wsa:Action>
     http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
   </wsa:Action>
   ...
  </s:Header>
  <s:Body>
   <wsmt:ResourceCreated>
     <wsa:Address>
```

```
(16)        http://1.2.3.4/wsman/
(17)      </wsa:Address>
(18)      <wsa:ReferenceParameters>
(19)        <wsman:ResourceURI>
(20)          http://example.org/2005/02/virtualDrive
(21)        </wsman:ResourceURI>
(22)        <wsman:SelectorSet>
(23)          <wsman:Selector Name="ID"> F: </wsman:Selector>
(24)        </wsman:SelectorSet>
(25)      </wsa:ReferenceParameters>
(26)    </wsmt:ResourceCreated>
(27)  </s:Body>
(28) </s:Envelope>
```

This example assumes that the default addressing model is in use. The response contains a ResourceCreated block (lines 14-26), which contains the new endpoint reference of the created resource, including its ResourceURI and the SelectorSet. This address would be used to retrieve the resource in a subsequent Get operation.

The service might use a network address that is the same as the <wsa:To> address in the Create request.

R7.6-7: The service may ignore any values in the initial representation that are considered read-only from the point of view of the underlying real-world object.

This rule allows Get, Put, and Create to share the same schema. Put also allows the service to ignore read-only properties during an update.

R7.6-8: If the success of an operation cannot be reported as described in this clause and cannot be reversed, the service should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess

7.7 Fragment-Level Access

Because the resource access mechanism defined in this specification works with entire instances and it can be inconvenient to specify hundreds or thousands of EPRs just to model fragment-level access with full EPRs, WS-Management supports the concept of fragment-level (property) access of resources that are normally accessed through the resource access operations. This access is done through special use of these operations.

Because of the XML schema limitations discussed in 7.6, simply returning a subset of the XML defined for the object being accessed is often incorrect because a subset may violate the XML schema for that fragment. To support resource access of fragments or individual elements of a representation object, several modifications to the basic resource access operations are made.

R7.7-1: A conformant service may support fragment-level access. If the service supports fragment-level access, the service shall not behave as if the normal access operations were in place but shall operate exclusively on the fragments specified. If the service does not support fragment-level access, it shall return a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAccess

R7.7-2: A conformant service that supports fragment-level access shall accept the following SOAP header in all requests and include it in all responses that transport the fragments:

```
<wsman:FragmentTransfer s:mustUnderstand="true">
  xpath to fragment
</wsman:FragmentTransfer>
```

The value of this header is the XPath 1.0 expression that identifies the fragment being transferred with relation to the full representation of the object. If an expression other than XPath 1.0 is used, a Dialect attribute can be added to indicate this, as follows:

```
(4)  <wsman:FragmentTransfer s:mustUnderstand="true"
(5)    Dialect="URIToNewFragmentDialect">
(6)    dialect expression
(7)  </wsman:FragmentTransfer>
```

The client needs to understand that unless the header is marked mustUnderstand="true", the service might process the request while ignoring the header, resulting in unexpected and potentially serious side effects.

XPath is explicitly defined as a dialect due to its importance, but it is not required that implementations support XPath as a fragment dialect. Any other type of language to describe fragment-level access is permitted as long as the Dialect value is set to indicate to the service what dialect is being used.

R7.7-3: For resource access fragment operations that use [XPath 1.0] (Dialect URI of http://www.w3.org/TR/1999/REC-xpath-19991116), the value of the /s:Envelope/s:Header/wsman:FragmentTransfer element is an XPath expression. This XPath expression is evaluated using the following context:

- **Context Node**: the root element of the XML representation of the resource addressed in the request that would be returned as the initial child element of the SOAP Body response if a Get operation was applied against the addressed resource without using fragment access
- **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 the request /s:Envelope/s:Header/wsman:FragmentTransfer element

This rule means that the XPath is to be interpreted relative to the XML representation of the resource and not relative to any of the SOAP content.

For the Enumeration operations, the XPath is interpreted as defined in clause 8, although the output is subsequently wrapped in wsman:XmlFragment wrappers after the XPath is evaluated.

An XPath value can refer to the entire node, so the concept of a fragment includes the entire object, making fragment-level access a proper superset of normal resource access operations.

If the full XPath expression syntax cannot be supported, a common subset for this purpose is described in ANNEX C of this specification. However, in such cases, the Dialect URI is still that of XPath.

R7.7-4: If a service understands fragment access but does not understand the specified fragment Dialect URI or the default dialect, the service shall issue a wsman:FragmentDialectNotSupported fault.

R7.7-5: All resource access messages in either direction of the XML fragments shall be wrapped with a <wsman:XmlFragment> wrapper that contains a definition that suppresses validation and allows any content to pass. A service shall reject any attempt to use wsman:FragmentTransfer unless the s:Body wraps the content using a wsman:XmlFragment wrapper. If any other usage is encountered, the service shall fault the request by using a wsmt:InvalidRepresentation fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFragment

Fragment access can occur at any level, including single element, complex elements, simple values, and attributes. In practice, services typically support only value-level access to elements.

R7.7-6: If fragment-level access is supported, a conformant service should support at least leaf-node, value-level access using an XPath expression that uses the */text()* NodeTest. In this case, the value is not wrapped with XML but is transferred directly as text within the wsman:XmlFragment wrapper.

In essence, the transferred content is whatever an XPath operation over the full XML would produce.

R7.7-7: If fragment-level access is supported but the filter expression exceeds the capability of the service, the service should return a wsman:CannotProcessFilter fault with text explaining why the filter was problematic.

R7.7-8: For all fragment-level operations, partial successes are not permitted. The entire meaning of the XPath expression or other dialect shall be fully observed by the service in all operations, and the entire fragment that is specified shall be successfully transferred in either direction. Otherwise, faults occur as if none of the operation had succeeded.

All faults are the same as for normal, "full" resource access operations.

The following clauses show how the underlying resource access operations change when transferring XML fragments.

7.8 Fragment-Level Get

Fragment-level Get is similar to full Get, except for the wsman:FragmentTransfer header (lines 25-27).

EXAMPLE 1: The following example is drawn from the example in 7.1:

```
(1)  <s:Envelope
(2)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)     xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(5)   <s:Header>
(6)     <wsa:To>
(7)       http://1.2.3.4/wsman
(8)     </wsa:To>
(9)     <wsman:ResourceURI>http://example.org/2005/02/physicalDisk
          </wsman:ResourceURI>
(10)    <wsa:ReplyTo>
(11)      <wsa:Address>
(12)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(13)      </wsa:Address>
(14)    </wsa:ReplyTo>
(15)    <wsa:Action>
(16)      http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
(17)    </wsa:Action>
(18)    <wsa:MessageID>
(19)      urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(20)    </wsa:MessageID>
(21)    <wsman:SelectorSet>
(22)      <wsman:Selector Name="LUN"> 2 </wsman:Selector>
```

```
(23)     </wsman:SelectorSet>
(24)     <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
(25)     <wsman:FragmentTransfer s:mustUnderstand="true">
(26)       Manufacturer
(27)     </wsman:FragmentTransfer>
(28)   </s:Header>
(29)   <s:Body/>
(30) </s:Envelope>
```

In this case, the service executes the specified XPath expression against the representation that would normally have been retrieved, and then return a fragment instead.

EXAMPLE 2: The service repeats the wsman:FragmentTransfer element in the GetResponse (lines 48-50) to reference the fragment and signal that a fragment has been transferred. The response is wrapped in a wsman:XmlFragment wrapper, which suppresses the schema validation that would otherwise apply.

```
(31)   <s:Envelope
(32)       xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(33)       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(34)       xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(35)     <s:Header>
(36)       <wsa:To>
(37)         http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(38)       </wsa:To>
(39)       <wsa:Action s:mustUnderstand="true">
(40)       http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(41)     </wsa:Action>
(42)     <wsa:MessageID s:mustUnderstand="true">
(43)       urn:uuid:1a7e7314-d791-4b4b-3eda-c00f7e833a8c
(44)     </wsa:MessageID>
(45)     <wsa:RelatesTo>
(46)       urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(47)     </wsa:RelatesTo>
(48)     <wsman:FragmentTransfer s:mustUnderstand="true">
(49)       Manufacturer
(50)     </wsman:FragmentTransfer>
(51)   </s:Header>
(52)   <s:Body>
(53)     <wsman:XmlFragment
    xmlns="http://schemas.example.org/2005/02/samples/physDisk">
(54)       <Manufacturer> Acme, Inc. </Manufacturer>
(55)     </wsman:XmlFragment>
(56)   </s:Body>
(57) </s:Envelope>
```

The output (lines 53-55) is like that supplied by a typical XPath processor.

To receive the value in isolation without an XML element wrapper, the client can use XPath techniques such as the text() operator to retrieve just the values.

EXAMPLE 3: The following example request uses text() to get the manufacturer name:

```
(1)  <wsman:FragmentTransfer s:mustUnderstand="true">
(2)    Manufacturer/text()
(3)  </wsman:FragmentTransfer>
```

This request results in the following XML in the response SOAP Body:

```
(1)  <wsman:XmlFragment>
(2)    Acme, Inc.
(3)  </wsman:XmlFragment>
```

7.9 Fragment-Level Put

Fragment-level Put works like regular Put except that it transfers only the part being updated. Although the fragment can be considered part of an instance from the observer's perspective, the referenced fragment is treated as the "instance" during the execution of the operation.

NOTE: Put is *always* an update operation of an existing element, whether a simple element or an array. To create or insert new elements, Create is required.

EXAMPLE 1: Consider the following XML for illustrative purposes:

```
<a>
  <b>
    <c> </c>
    <d> </d>
  </b>
  <e>
    <f> </f>
    <g> </g>
  </e>
</a>
```

Although `<a>` is the entire representation of the resource instance, if the operation references the a/b node during the Put operation, using an XPath expression of “b”, then the content of `<b>` is updated without touching other parts of `<a>`, such as `<e>`. If the client wants to update only `<d>`, then the XPath expression used is “b/d”.

EXAMPLE 2: Continuing from the example in SECTION 7.1, if the client wanted to update the <BootPartition> value from 0 to 1, the following Put fragment could be sent to the service:

```
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
  <s:Header>
    <wsa:To>
      http://1.2.3.4/wsman
    </wsa:To>
    <wsman:ResourceURI>http://example.org/2005/02/physicalDisk
      </wsman:ResourceURI>
    <wsa:ReplyTo>
      <wsa:Address>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:d9726315-bc91-2222-9ed8-c044c9658a87
    </wsa:MessageID>
    <wsman:SelectorSet>
      <wsman:Selector Name="LUN"> 2 </wsman:Selector>
    </wsman:SelectorSet>
    <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
    <wsman:FragmentTransfer s:mustUnderstand="true">
      BootPartition
    </wsman:FragmentTransfer>
  </s:Header>
  <s:Body>
```

```
    <wsman:XmlFragment>
      <BootPartition> 1 </BootPartition>
    </wsman:XmlFragment>
  </s:Body>
</s:Envelope>
```

EXAMPLE 3: The <BootPartition> wrapper is present because the XPath value specifies this. If "BootPartition/text()" were used as the expression, the Body would contain just the value, as in the following example:

```
  <s:Header>
    ...
    <wsman:FragmentTransfer s:mustUnderstand="true">
      BootPartition/text()
    </wsman:FragmentTransfer>
  </s:Header>
  <s:Body>
    <wsman:XmlFragment>
      1
    </wsman:XmlFragment>
  </s:Body>
```

If the corresponding update occurs, the new representation matches, so no s:Body result is expected, although returning it is always legal. If a value does not match what was requested, the service needs to supply only the parts that are different than what is requested. This situation would generally not occur for single values because a failure to honor the new value would result in a wsmt:InvalidRepresentation fault.

EXAMPLE 4: The following is a sample reply:

```
<s:Envelope
      xmlns:s="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
      xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <s:Header>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
      </wsa:To>
      <wsa:Action s:mustUnderstand="true">
      http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse
    </wsa:Action>
    <wsa:MessageID s:mustUnderstand="true">
      urn:uuid:ee7f13b5-0091-430b-9ed8-2e12fbaa8a7e
    </wsa:MessageID>
    <wsa:RelatesTo>
      urn:uuid:d9726315-bc91-2222-9ed8-c044c9658a87
    </wsa:RelatesTo>
    <wsman:FragmentTransfer s:mustUnderstand="true">
      BootPartition/text()
    </wsman:FragmentTransfer>
  </s:Header>
  <s:Body>
    <wsman:XmlFragment>
      1
    </wsman:XmlFragment>
  </s:Body>
</s:Envelope>
```

R7.9-1: This rule intentionally left blank.

R7.9-2: If the service encounters an attempt to update a read-only value using a fragment-level Put operation, it should return a wsa:ActionNotSupported fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ActionMismatch

NOTE: The fragment-level Put operation implies replacement or update and does not insert new values into the representation object. Thus, it is not appropriate to use Put to insert a new value at the end of an array, for example. The entire array can be returned and then updated and replaced (because it is therefore an update of the entire array), but a single operation to insert a new element in the middle or at the end of an array is actually a Create operation.

As stated in 7.4, if the new representation differs from the input, the new representation is to be returned in the response. With fragment-level Put, this rule applies only to the portion of the representation object being written, not the entire object. If a single value is written and accepted, but has side effects on other values in the representation, the entire object is *not* returned.

To set a value to NULL without removing it as an element, use an attribute value of xsi:nil on the element being set to NULL to ensure that the fragment path is adjusted appropriately.

EXAMPLE 5:

```
(73)  <s:Header> ...
(74)    <wsman:FragmentTransfer s:mustUnderstand="true">
(75)      AssetLabel
(76)    </wsman:FragmentTransfer>
(77)    ...
(78)  </Header>
(79)  <s:Body>
(80)    <wsman:XmlFragment xmlns:xsi="www.w3.org/2001/XMLSchema-instance">
(81)      <AssetLabel xsi:nil="true"/>
(82)    </wsman:XmlFragment>
(83)  </s:Body>
```

7.10 Fragment-Level Delete

Fragment-level Delete applies only if the XML schema for the targeted object supports optional elements that can be removed from the representation object, or supports arrays (repeated elements) with varying numbers of elements and the client wants to remove an element in an array. If replacement of an entire array is needed, fragment-level Put can be used. For array access, the XPath array access notation can conveniently be used. To delete a value that is legal to remove (according to the rules of the schema for the object), the wsman:FragmentTransfer expression identifies the item to be removed.

EXAMPLE 1:

```
(1)  <wsman:FragmentTransfer s:mustUnderstand="true">
(2)    VolumeLabel
(3)  </wsman:FragmentTransfer>
```

To set a value to NULL without removing it as an element, use fragment-level Put with a value of xsi:nil.

To delete an array element, use the XPath [ ] operators.

EXAMPLE 2: The following example deletes the second <BlockedIPAddress> element in the representation. (XPath arrays are 1 based.)

```
<wsman:FragmentTransfer s:mustUnderstand="true">
  BlockedIPAddress[2]
</wsman:FragmentTransfer>
```

The <s:Body> is empty for all Delete operations, even with fragment-level access, and all normal faults for Delete apply.

R7.10-1: If a value cannot be deleted because of locking conditions or similar phenomena, the service should return a wsman:AccessDenied fault.

7.11 Fragment-Level Create

Fragment-level Create applies only if the XML schema for the targeted object supports optional elements that are not currently present, or supports arrays with varying numbers of elements and the client wants to insert an element in an array (a repeated element). If entire array replacement is needed, Fragment-level Put can be used. For array access, the XPath array access notation (the [ ] operators) can be used.

NOTE: Create can be used only to add new content, not to update existing content.

To insert a value that can be legally added (according to the rules of the schema for the object), the wsman:FragmentTransfer expression identifies the item to be added.

EXAMPLE 1: For example, assume the following message fragment is sent to a LogicalDisk resource:

```
<wsman:FragmentTransfer s:mustUnderstand="true">
  VolumeLabel
</wsman:FragmentTransfer>
```

EXAMPLE 2: In this case, the <Body> contains both the element and the value:

```
<s:Body>
  <wsman:XmlFragment>
    <VolumeLabel> MyDisk </VolumeLabel>
  </wsman:XmlFragment>
</s:Body>
```

This operation creates a <VolumeLabel> element where none existed before.

EXAMPLE 3: To create the target using the value alone, apply the XPath text() operator to the path, as follows:

```
<wsman:FragmentTransfer s:mustUnderstand="true">
    VolumeLabel/text()
  </wsman:FragmentTransfer>
```

EXAMPLE 4: The body of Create contains the value to be inserted and is the same as for fragment-level Put:

```
<s:Body>
  <wsman:XmlFragment>
    MyDisk
  </wsman:XmlFragment>
</s:Body>
```

To create an array element in the target, the XPath [ ] operator may be used. To insert a new element at the end of the array, the user needs to know the number of elements in the array so that the new index can be used.

EXAMPLE 5: The following message fragment is sent to an InternetServer resource:

```
(17)   <wsman:FragmentTransfer s:mustUnderstand="true">
(18)     BlockedIPAddress[3]
(19)   </wsman:FragmentTransfer>
```

Insertion of a new element within the array is done using the index of the desired location, and the array expands at that location to accommodate the new element. Using Put at this location *overwrites* the existing array element, whereas Create inserts a *new* element, making the array larger.

The body of Create contains the value to be inserted and is the same as for fragment-level Put.

EXAMPLE 6:

```
(20)   <s:Body>
(21)     <wsman:XmlFragment>
(22)       <BlockedIPAddress> 123.12.188.44 </BlockedIPAddress>
(23)     </wsman:XmlFragment>
(24)   </s:Body>
```

This operation adds a third IP address to the <BlockedIPAddress> array (a repeated element), assuming that at least two elements are at that level already.

R7.11-1: A service shall not use fragment-level Create to modify the value of an existing property. If the targeted object and the targeted property already exists, the service should return a wsman:AlreadyExists fault.

R7.11-2: If the Create fails because the result would not conform to the schema in some way, the service should return a wsmt:InvalidRepresentation fault.

As defined in 7.6, the CreateResponse contains the EPR of the created resource. In the case of fragment-level Create, the response additionally contains the wsman:FragmentTransfer block, including the path (line 12), in a SOAP header.

EXAMPLE 7: In the following example, the ResourceCreated EPR continues to refer to the entire object, not just to the fragment.

```
(25)   <s:Envelope
(26)       xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(27)       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(28)       xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
(29)       xmlns:wsmt="http://schemas.xmlsoap.org/ws/2004/09/transfer">
(30)     <s:Header>
(31)       ...
(32)       <wsa:Action>
(33)         http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
(34)     </wsa:Action>
(35)     <wsman:FragmentTransfer s:mustUnderstand="true">
(36)       Path To Fragment
(37)     </wsman:FragmentTransfer>
(38)     ...
(39)   </s:Header>
(40)   <s:Body>
(41)     <wsmt:ResourceCreated>
(42)       <wsa:Address> ... </wsa:Address>
(43)       <wsa:ReferenceParameters>
(44)         <wsman:SelectorSet>
(45)           <wsman:Selector ...> ... </wsman:Selector>
(46)         </wsman:SelectorSet>
(47)       </wsa:ReferenceParameters>
(48)     </wsmt:ResourceCreated>
(49)   </s:Body>
```

```
(50) </s:Envelope>
```

As discussed in 7.6, to remain compatible with WSDL, only the EPR of the item is returned in the SOAP Body, in spite of other options discussed in 7.6.

8 Enumeration of Datasets

8.1 General

This clause defines a set of operations that can be used as a basis for iteration through the members of a management-specific dataset or collection. WS-Management qualifies and extends these operations as described in this clause.

There are numerous applications for which a simple single-request/single-reply metaphor is insufficient for transferring large data sets over SOAP. Applications that do not fit into this simple paradigm include streaming, traversal, query, and enumeration.

This clause defines a simple SOAP-based protocol for enumeration that allows the data source to provide a session abstraction, called an enumeration context, to a consumer that represents a logical cursor through a sequence of data items. The consumer can then request XML element information items using this enumeration context over the span of one or more SOAP messages.

Somewhere, state must be maintained regarding the progress of the iteration. This state may be maintained between requests by the data source being enumerated or by the data consumer. The operations defined in this clause allow the data source to decide, on a request-by-request basis, which party is responsible for maintaining this state for the next request.

In its simplest form, there is a single operation, Pull, which allows a data source, in the context of a specific enumeration, to produce a sequence of XML elements in the body of a SOAP message. Each subsequent Pull operation returns the next N elements in the aggregate sequence.

A data source may provide a custom mechanism for starting a new enumeration. For instance, a data source that provides access to a SQL database may support a SELECT operation that performs a database query and uses an explicit database cursor to iterate through the returned rows. In general, however, it is simpler if all data sources support a single, standard operation to start an enumeration. This specification defines such an operation, Enumerate, which data sources may implement for starting a new enumeration of a data source. The Enumerate operation is used to create new enumeration contexts for subsequent traversal/retrieval. Each Enumerate operation results in a distinct enumeration context, each with its own logical cursor/position.

It should be emphasized that different enumerations of the same data source may produce different results; this may happen even for two enumeration contexts created concurrently by a single consumer using identical Enumerate requests. In general, the consumer of an enumeration should not make any assumptions about the ordering or completeness of the enumeration; the returned data items represent a selection by the data source of items it wishes to present to that consumer at that time in that order, with no guarantee that every available item is returned or that the order in which items is returned has any semantic meaning whatsoever (of course, any specific data source may provide strong guarantees, if so desired). In particular, it should be noted that the very act of enumerating the contents of a data source may modify the contents of the data source; for instance, a queue might be represented as a data source such that items that are returned in a Pull response are removed from the queue.

Enumeration contexts represent a specific traversal through a sequence of XML information items. An Enumerate operation may be used to establish an enumeration context from a data source. A Pull operation is used to fetch information items from a data source according to a specific enumeration context. A Release operation is used to tell a data source that the consumer is abandoning an enumeration context before it has completed the enumeration.

Enumeration contexts are represented as XML data that is opaque to the consumer. Initially, the consumer gets an enumeration context from the data source by means of an Enumerate operation. The consumer then passes that XML data back to the data source in the Pull request. Optionally, the data source may return an updated enumeration context in the Pull response; when present, this new enumeration context should replace the old one on the consumer, and it should be passed to the data source in all future responses until and unless the data source again returns an updated enumeration context.

Consumers should not reuse old enumeration contexts that have been replaced by the data source. Using a replaced enumeration context in a Pull response may yield undefined results, including being ignored or generating a fault.

After the last element in a sequence has been returned, or the enumeration context has expired, the enumeration context is considered invalid and the result of subsequent operations referencing that context is undefined.

Callers may issue a Release operation against a valid enumeration context at any time, which causes the enumeration context to become invalid and allows the data source to free up any resources it may have allocated to the enumeration. Issuing a Release operation prior to reaching the end of the sequence of elements is explicitly allowed; however, no further operations should be issued after a Release.

In addition, the data source may invalidate an enumeration context at any time, as necessary.

If a resource with multiple instances provides a mechanism for enumerating or querying the set of instances, the operations defined in this clause can be used to perform the iteration.

R8.1-1: A service may support the Enumeration operations if enumeration of any kind is supported.

R8.1-2: If simple, unfiltered enumeration of resource instances is exposed through Web services, a conformant service shall support the Enumeration operations to expose this. The service may also support other techniques for enumerating the instances.

R8.1-3: If filtered enumeration (queries) of resource instances is exposed through Web services, a conformant service should support the Enumeration operations to expose this. The service may also support other techniques for enumerating the instances.

This clause indicates that enumeration is a three-part operation:

1) An initial Enumerate message is issued to establish the enumeration context.
2) Pull operations are used to iterate over the result set.
3) When the enumeration iterator is no longer required and not yet exhausted, a Release message is issued to release the enumerator and associated resources.

As with other WS-Management methods, the enumeration can make use of wsman:OptionSet.

R8.1-4: A service may implement wsmen:Renew, wsmen:GetStatus and wsmen:EnumerationEnd messages; however, in constrained environments these are candidates for exclusion. If these messages are not supported, then a wsa:ActionNotSupported fault shall be returned in response to these requests.

R8.1-5: If a service is exposing enumeration, it shall at least support the following messages: Enumerate, Pull, and Release, and their associated responses.

If the service does not support stateful enumerators, the Release is a simple no-op, so it is trivial to implement. (It always succeeds when the operation is valid.) However, it is supported to allow for the uniform construction of clients.

R8.1-6: The Pull and Release operations are a continuation of the original Enumerate operation. The service should enforce the same authentication and authorization throughout the entire sequence of operations and should fault any attempt to change credentials during the sequence.

Some transports such as HTTP might drop or reestablish connections between Enumerate and subsequent Pull operations, or between Pull operations. It is expected that services will allow the enumeration to continue uninterrupted, but for practical reasons some services might require that the same connection be used. This specification establishes no requirements in this regard. However, R8.1-6 establishes that the user credentials do not change during the entire enumeration sequence.

8.2 Enumerate

All data sources shall support some operation that allows an enumeration to be started. A data source may support the Enumerate operation, or it may provide some other mechanism for starting an enumeration and receiving an enumeration context.

The Enumerate operation is initiated by sending an Enumerate request message to the data source. The Enumerate request message shall be of the following form:

```
(1) <s:Envelope …>
(2)   <s:Header …>
(3)     <wsa:Action>
(4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
(5)     </wsa:Action>
(6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
(7)     <wsa:To>xs:anyURI</wsa:To>
(8)     …
(9)   </s:Header>
(10)  <s:Body …>
(11)    <wsmen:Enumerate …>
(12)      <wsmen:EndTo>endpoint-reference</wsmen:EndTo> ?
(13)      <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
(14)      <wsmen:Filter Dialect="xs:anyURI"?> xs:any </wsmen:Filter> ?
(15)      …
(16)    </wsmen:Enumerate>
(17)  </s:Body>
(18)</s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate.

If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/*/wsmen:EndTo

This optional element denotes where to send an EnumerationEnd message if the enumeration is terminated unexpectedly. If present, this element shall be of type wsa:EndpointReferenceType. The default is to not send this message. The endpoint referenced by this EPR shall implement a binding of the "EnumEndEndpoint" portType described in ANNEX H.

/s:Envelope/s:Body/*/wsmen:Expires

Requested expiration time for the enumeration. (No implied value.) The data 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 enumeration's creation time. Both specific times and durations are interpreted based on the data source's clock.

If this element does not appear, then the request is for an enumeration that will not expire. That is, the consumer is requesting the data source to create an enumeration with an indefinite lifetime. If the data source grants such an enumeration, it will terminate when the end of the enumeration is reached, or if the consumer sends a Release request, or by the data 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 data source, then the request shall fail, and the data source may generate a wsmen:InvalidExpirationTime fault indicating that an invalid expiration time was requested.

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

/s:Envelope/s:Body/wsmen:Enumerate/wsmen:Filter

This optional element contains a Boolean predicate in some dialect (see /s:Envelope/s:Body/*/wsmen:Filter/@Dialect) that all elements of interest must satisfy. The resultant enumeration context shall not return elements for which this predicate expression evaluates to the value false. If this element is absent, then the implied value is the expression true(), indicating that no filtering is desired.

If the data source does not support filtering, the request shall fail, and the data source may generate a wsmen:FilteringNotSupported SOAP fault as follows:

If the data source supports filtering but cannot honor the requested filter dialect, the request shall fail, and the data source may generate a wsmen:FilterDialectRequestedUnavailable SOAP fault as follows:

If the data source supports filtering and the requested dialect but cannot process the requested filter content, the request shall fail, and the data source may generate a wsman:CannotProcessFilter SOAP fault as follows:

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

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

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

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

- **Context Node:** any XML element that could be returned as a direct child of the Items element
- **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/*/wsmen:Filter

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

Upon successful processing of an Enumerate request message, a data source is expected to create an enumeration context and return that context in an Enumerate response message, which shall adhere to the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)   http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse
(5)      </wsa:Action>
(6)      <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body …>
(11)     <wsmen:EnumerateResponse …>
(12)       <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
(13)       <wsmen:EnumerationContext>…</wsmen:EnumerationContext>
(14)       …
(15)     </wsmen:EnumerateResponse>
(16)   </s:Body>
(17) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse

If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/*/wsmen:Expires

The expiration time assigned by the data 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 enumeration will not expire. That is, the enumeration has an indefinite lifetime. It will terminate when the end of the enumeration is reached, if the consumer sends a Release request, or by the data source at any time for reasons such as connection termination, resource constraints, or system shut-down.

/s:Envelope/s:Body/wsmen:EnumerateResponse/wsmen:EnumerationContext

The required EnumerationContext element contains the XML representation of the new enumeration context. The consumer is required to pass this XML data in Pull requests for this enumeration context, until and unless a PullResponse message updates the enumeration context.

8.2.1 General

WS-Management qualifies the Enumerate operation as described in this clause.

R8.2.1-1: A conformant service may accept a wsmen:Enumerate message with an EndTo address; however, if EnumerationEnd is not supported, a service may instead issue a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode

R8.2.1-2: A conformant service shall accept an Enumerate message with an Expires timeout or fault with wsman:UnsupportedFeature and the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime

R8.2.1-3: The wsman:Filter element (see 8.3) in the Enumerate body shall be either simple text or a single complex XML element. A conformant service shall not accept mixed content of both text and elements, or multiple peer XML elements under the wsman:Filter element.

Although this use of mixed content is allowed in the general case of Enumerate, it is unnecessarily complex for WS-Management implementations.

A common filter dialect is XPath 1.0 (identified by the Dialect URI http://www.w3.org/TR/1999/REC-xpath-19991116). Resource-constrained implementations might have difficulty exporting full XPath processing and yet still want to use a subset of XPath syntax. As long as the filter expression is a proper subset of the specified dialect, it is legal and can be described using that Dialect value.

No rule mandates the use of XPath or any subset as a filtering dialect. If no Dialect is specified, the default interpretation is that the Filter value is XPath (as specified previously in this clause).

R8.2.1-4: A conformant service may not support the entire syntax and processing power of the specified Filter Dialect. The only requirement is that the specified Filter is syntactically correct within the definition of the Dialect. Subsets are therefore legal. If the specified Filter exceeds the capability of the service, the service should return a wsmen:CannotProcessFilter fault with some text indicating what went wrong.

Some services require filters to function because their search space is so large that simple enumeration is meaningless or impossible.

R8.2.1-5: If a wsman:Filter is required, a conformant service shall fault any request without a wsman:Filter, by using a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired

R8.2.1-6: A conformant service may block, fault (using wsman:Concurrency faults), or allow other concurrent operations on the resource for the duration of the enumeration, and may include or exclude the results of such operations as part of any enumeration still in progress.

If clients execute other operations, such as Create or Delete, while an enumeration is occurring, this specification makes no restrictions on the behavior of the enumeration. The service can include or exclude the results of these operations in real-time, can produce an initial snapshot of the enumeration and execute the Pull requests from this snapshot, or can deny access to other operations while enumerations are in progress.

8.2.2 Enumeration "Count" Option

To give clients an estimate of the number of items in an enumeration, two optional SOAP headers are defined: one for use in the request message to return an approximate count of items in an enumeration sequence, and a corresponding header for use in the response to return this value to the client.

These SOAP headers are defined for use with the Enumerate and Pull messages and their responses. The header used in Enumerate and Pull is as follows:

```
(1)  <s:Header>
(2)    ...
(3)    <wsman:RequestTotalItemsCountEstimate …/>
(4)  </s:Header>
```

The header used by the service to return the value is as follows:

```
(5)  <s:Header>
(6)    ...
(7)    <wsman:TotalItemsCountEstimate>
```

```
(8)      xs:nonNegativeInteger
(9)    </wsman: TotalItemsCountEstimate>
(10) </s:Header>
```

The following definitions provide additional, normative constraints on the preceding headers:

wsman:RequestTotalItemsCountEstimate

when present as a SOAP header on an Enumerate or Pull message, indicates that the client is requesting that the associated response message includes an estimate of the total number of items in the enumeration sequence

This SOAP header does not have any meaning defined by this specification when included with any other messages.

wsman:TotalItemsCountEstimate

when present as a SOAP header on an EnumerateResponse or PullResponse message, indicates the approximate number of items in the enumeration sequence

This is the total number of items and not the remaining number of items in the sequence. This SOAP header does not have any meaning defined by this specification when included with any other messages.

When a service understands the TotalItemsCountEstimate feature but cannot determine the number of items, the service responds with the wsman:TotalItemsCountEstimate element having an xsi:nil attribute with value ‘true’, and having no value, as follows:

```
(1) <wsman:TotalItemsCountEstimate xsi:nil="true"/>
```

R8.2.2-1: A conformant service may support the ability to return an estimate of the number of items in an enumeration sequence. If a service receives an Enumerate or Pull message without the wsman:RequestTotalItemsCountEstimate SOAP header, the service shall not return the wsman:TotalItemsCountEstimate SOAP header on the associated response message.

R8.2.2-2: The value returned in the wsman:TotalItemsCountEstimate SOAP header is only an estimate of the number of items in the sequence. The client should not use the wsman:TotalItemsCountEstimate value for determining an end of enumeration instead of using EndOfSequence.

This mechanism is intended to assist clients in determining the percentage of completion of an enumeration as it progresses. When a service sends a result count estimate after a previous estimate for the same enumeration sequence, the most recent total results count estimate is considered to be the more precise estimate.

8.2.3 Optimization for Enumerations with Small Result Sets

To optimize the number of round-trip messages required to enumerate the items in an enumerable resource, a client can request optimized enumeration behavior. This behavior is useful in cases where the enumeration has such a small number of items that the initial EnumerateResponse could reasonably include the entire result, without the need for a subsequent Pull to retrieve the items. This mechanism can be used even for large enumerations to get the first few results in the initial response.

A client initiates an optimized enumeration by placing the wsman:OptimizeEnumeration element as a child element of the Enumerate element, and can optionally include the wsman:MaxElements element, as follows:

EXAMPLE:

```
(1)  <s:Body>
```

```
(2)   <wsmen:Enumerate>
(3)     ...
(4)     <wsman:OptimizeEnumeration/>
(5)     <wsman:MaxElements>xs:positiveInteger</wsman:MaxElements> ?
(6)   </wsmen:Enumerate>
(7) </s:Body>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsmen:Enumerate/wsman:OptimizeEnumeration

when present as a child of the Enumerate element, indicates that the client is requesting an optimized enumeration

wsmen:Enumerate/wsman:MaxElements

(optional) indicates the maximum number of items the consumer is willing to accept in the EnumerateResponse

It plays the same role as wsmen:Pull/wsmen:MaxElements. When this element is absent, its implied value is 1.

R8.2.3-1: A conformant service may support enumeration optimization. If a service receives the wsman:OptimizeEnumeration element in an Enumerate message and it does not support enumeration optimization, it should ignore the element and complete the enumeration request as if the element were not present.

If the service ignores the element, the client continues with a subsequent Pull as if the option was not in force. The client requires no special mechanisms over what was needed for normal enumeration if the optimization request is ignored.

R8.2.3-2: A conformant service that receives an Enumerate message without the wsman:OptimizeEnumeration element shall not return any enumeration items in the EnumerateResponse message and shall return a EnumerationContext initialized to return the first items when the first Pull message is received.

If the service implements the optimization even if it was not requested, clients unaware of the optimization will incorrectly process the enumeration result.

R8.2.3-3: A conformant service that receives an Enumerate message with the wsman:OptimizeEnumeration element shall not return more elements in the Enumerate response message than requested in the wsman:MaxElements element (or no more than1 item if the wsman:MaxElements element is not present). Implementations may return fewer items based on either the wsman:OperationTimeout SOAP header, wsman:MaxEnvelopeSize SOAP header, or implementation-specific constraints.

When requested by the client, a service implementing the optimized enumeration will respond with the following additional content in an EnumerateResponse message:

```
(1)  <s:Body>
(2)    <wsmen:EnumerateResponse>
(3)      <wsmen:EnumerationContext> ... </wsmen:EnumerationContext>
(4)      <wsman:Items>
(5)        ...same as for wsmen:Items in wsmen:PullResponse
(6)      </wsman:Items> ?
(7)      <wsman:EndOfSequence/> ?
(8)      ...
(9)    </wsmen:EnumerateResponse>
(10) </s:Body>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsman:Items

(optional) contains one or more enumeration-specific elements as would have been encoded for Items in a PullResponse

The service will return no more than wsman:MaxElements elements in this list if wsman:MaxElements is specified in the request message, or one element if wsman:MaxElements was omitted.

wsman:EndOfSequence

(optional) indicates that no more elements are available from this enumeration and that the entire result (even if there are zero elements) is contained within the wsman:Items element

wsmen:EnumerationContext

required context for requesting additional items, if any, in subsequent Pull messages

If the wsman:EndOfSequence is also present, the EnumerationContext cannot be used in a subsequent Pull request. The service should observe the same fault usage that would occur if the EnumerationContext were used in a Pull request after the EndOfSequence element occurred in a PullResponse. Although the EnumerationContext element must be present, no value is required; therefore, in cases where the wsman:EndOfSequence element is present, the value for EnumerationContext can be empty.

EXAMPLE:

```
(1) <s:Body>
(2)   <wsmen:EnumerateResponse>
(3)     <wsmen:EnumerationContext/>
(4)     <wsman:Items>
(5)       Items
(6)     </wsman:Items>
(7)     <wsman:EndOfSequence/>
(8)     ...
(9)   </wsmen:EnumerateResponse>
(10) </s:Body>
```

R8.2.3-4: A conformant service that supports optimized enumeration and is responding with an EnumerateResponse message shall include the wsman:Items element, the wsman:EndOfSequence element, or both in the response as an indication to the client that the optimized enumeration request was understood and honored.

If neither wsman:Items nor wsman:EndOfSequence is in the EnumerateResponse message, the client can continue to use the enumeration message exchanges as defined in 8.2.1.

R8.2.3-5: A conformant service that supports optimized enumeration and has not returned all items of the enumeration sequence in the EnumerateResponse message shall return an EnumerationContext element that is initialized such that a subsequent Pull message will return the set of items after those returned in the EnumerateResponse. If all items of the enumeration sequence have been returned in the EnumerateResponse message, the service should return an empty EnumerationContext element and shall return the wsman:EndOfSequence element in the response.

A client that has requested optimized enumeration can determine if this request was understood and honored by the service by examining the response message.

Clients concerned about the size of the initial response, irrespective of the number of items, can use the wsman:MaxEnvelopeSize mechanism described in 6.2.

8.3 Filter Interpretation

The Filter expression is constrained to be a Boolean predicate. To support ad hoc queries including projections, WS-Management defines a wsman:Filter element of exactly the same form as in the Enumeration filter except that the filter expression is not constrained to be a Boolean predicate. This allows the use of enumeration using existing query languages such as SQL and CQL, which combine predicate and projection information in the same syntax. The use of projections is defined by the filter dialect, not by WS-Management.

```
(1) <wsman:Filter Dialect="xs:anyURI"?> xs:any </wsman:Filter>
```

The Dialect attribute is optional. When not specified, it has the following implied value:

http://www.w3.org/TR/1999/REC-xpath-19991116

This dialect allows any full XPath expression or subset to be used.

The wsman:Filter element is a child of the Enumerate element.

If the filter dialect used for the Enumerate message is XPath 1.0, the context node is the same as that specified in 8.1.

R8.3-1: If a service supports filtered enumeration using Filter, it shall also support filtering using wsman:Filter. This rule allows client stacks to always pick the wsman XML namespace for the Filter element. Even though a service supports wsman:Filter, it is not required to support projections.

R8.3-2: If a service supports filtered enumeration using wsman:Filter, it should also support filtering using Filter.

R8.3-3: If an Enumerate request contains both Filter and wsman:Filter, the service shall return a wsmen:CannotProcessFilter fault.

Filters are generally intended to select entire XML document representations. However, most query languages have both filtering and compositional capabilities in that they can return subsets of the original representation, or perform complex operations on the original representation and return something entirely new.

This specification places no restriction on the capabilities of the service, but services may elect to provide only simple filtering capability and no compositional capabilities. In general, filtering dialects fall into the following simple hierarchy:

1) simple enumeration with no filtering
2) filtered enumeration with no representation change (within the capabilities of XPath, for example)
3) filtered enumeration in which a subset of each item is selected (within the capabilities of XPath, for example)
4) composition of new output (XQuery), including simple projection

Most services fall into the first or second category. However, if a service wants to support fragment-level enumeration to complement fragment-level access (7.7), the service can implement category 3 as well. Only rarely do services implement category 4.

XPath 1.0 can be used simply for filtering, or it can be used to send back subsets of the representation (or even the values without XML wrappers). In cases where the result is not just filtered but also "altered," the technique in 8.6 applies.

If full XPath cannot be supported, a common subset for this purpose is described in D.3 of this specification.

EXAMPLE 1: Following is a typical example of the use of XPath in a filter. Assume that each item in the enumeration to be delivered has the following XML content:

```
<s:Body>
  ...
  <wsmen:Items>
    <DiskInfo xmlns="...">
      <LogicalDisk>C:</LogicalDisk>
      <CurrentMegabytes>12</CurrentMegabytes>
      <BackupDrive> true </BackupDrive>
    </DiskInfo>
    ...
  </wsmen:Items>
</s:Body>
```

The anchor point for the XPath evaluation is at the first element of each item within the Items wrapper, and it does not reference the s:Body or Items elements. The XPath expression is evaluated as if each item in the Items block were a separate document.

EXAMPLE 2: When used for simple document processing, the following four XPath expressions "select" the entire DiskInfo node:

```
/
/DiskInfo
../DiskInfo
.
```

If used as a "filter," this XPath expression does not filter out any instances and is the same as selecting all instances, or omitting the filter entirely. However, using the following syntax, the XPath expression selects the XML node only if the test expression in brackets evaluates to logical "true":

```
../DiskInfo[LogicalDisk="C:"]
```

In this case, the item is selected only if it refers to disk drive "C:"; otherwise the XML node is not selected. This XPath expression filters out all DiskInfo instances for other drives.

EXAMPLE 3: Full XPath implementations may support more complex test expressions, as follows:

```
../DiskInfo[CurrentMegabytes>"10" and CurrentMegabytes <"200"]
```

This action selects only drives with free space within the range of values specified.

In essence, the XML form of the event passes logically through the XPath processor to see if it would be selected. If so, it is delivered in the enumeration. If not, the item is discarded and not delivered as part of the enumeration.

See the related clause (10.2.2) on filtering over subscriptions.

8.4 Pull

The Pull operation is initiated by sending a Pull request message to the data source. The Pull request message shall be of the following form:

```
(1) <s:Envelope …>
(2)   <s:Header …>
(3)     <wsa:Action>
(4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
(5)     </wsa:Action>
(6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
(7)     <wsa:ReplyTo>wsa:EndpointReference</wsa:ReplyTo>
(8)     <wsa:To>xs:anyURI</wsa:To>
(9)     …
(10)  </s:Header>
(11)  <s:Body …>
(12)    <wsmen:Pull …>
(13)      <wsmen:EnumerationContext>…</wsmen:EnumerationContext>
(14)      <wsmen:MaxTime>xs:duration</wsmen:MaxTime> ?
(15)      <wsmen:MaxElements>xs:long</wsmen:MaxElements> ?
(16)      <wsmen:MaxCharacters>xs:long</wsmen:MaxCharacters> ?
(17)      …
(18)    </wsmen:Pull>
(19)  </s:Body>
(20)</s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull

If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/wsmen:Pull/wsmen:EnumerationContext

This required element contains the XML data that represents the current enumeration context.

If the enumeration context is not valid, because it has been replaced in the response to another Pull request, it has completed (EndOfSequence has been returned in a Pull response), it has been Released, it has expired, or the data source has had to invalidate the context, then the data source should fail the request, and may generate a wsmen:InvalidEnumerationContext fault.

The data source may not be able to determine that an enumeration context is not valid, especially if all of the state associated with the enumeration is kept in the enumeration context and refreshed on every PullResponse.

/s:Envelope/s:Body/wsmen:Pull/wsmen:MaxTime

This optional element (of type xs:duration) indicates the maximum amount of time the initiator is willing to allow the data source to assemble the Pull response. When this element is absent, the data source is not required to limit the amount of time it takes to assemble the Pull response.

This is useful with data sources that accumulate elements over time and package them into a single Pull response.

/s:Envelope/s:Body/wsmen:Pull/wsmen:MaxElements

This optional element (of type xs:long) indicates the number of items (child elements of Items in the Pull response) the consumer is willing to accept. When this element is absent, its implied value is 1. Implementations shall not return more than this number of elements in the Pull response message. Implementations may return fewer than this number based on either the MaxTime timeout, the MaxCharacters size limit, or implementation-specific constraints.

/s:Envelope/s:Body/wsmen:Pull/wsmen:MaxCharacters

This optional element (of type xs:long) indicates the maximum size of the returned elements, in Unicode characters, that the initiator is willing to accept. When this element is absent, the data source is not required to limit the number of characters in the Pull response. Implementations shall not return a Pull response message whose Items element is larger than MaxCharacters. Implementations may return a smaller message based on the MaxTime timeout, the MaxElements limit, or implementation-specific constraints.

Even if a Pull request contains a MaxCharacters element, the consumer shall be prepared to receive a Pull response that contains more data characters than specified, as XML canonicalization or alternate XML serialization algorithms may change the size of the representation.

It may happen that the next item the data source would return to the consumer is larger than MaxCharacters. In this case, the data source may skip the item, or may return an abbreviated representation of the item that fits inside MaxCharacters. If the data source skips the item, it may return it as part of the response to a future Pull request with a larger value of MaxCharacters, or it may omit it entirely from the enumeration. If the oversize item is the last item to be returned for this enumeration context and the data source skips it, it shall include the EndOfSequence item in the Pull response and invalidate the enumeration context; that is, it may not return zero items but not consider the enumeration completed. See the discussion of EndOfSequence later in this clause.

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

Upon receipt of a Pull request message, the data source may wait as long as it deems necessary (but not longer than the value of the MaxTime element, if present) to produce a message for delivery to the consumer. The data source shall recognize the MaxTime element and return the wsmen:TimedOut fault if no elements are available prior to the request message's deadline.

However, this fault should not cause the enumeration context to become invalid (of course, the data source may invalidate the enumeration context for other reasons). That is, the requestor should be able to issue additional Pull requests using this enumeration context after receiving this fault.

Upon successful processing of a Pull request message, a data source is expected to return a Pull response message, which shall adhere to the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse
(5)      </wsa:Action>
(6)      <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body …>
(11)     <wsmen:PullResponse …>
(12)       <wsmen:EnumerationContext>…</wsmen:EnumerationContext> ?
(13)       <wsmen:Items> ?
(14)         <xs:any> enumeration-specific element </xs:any> +
(15)       </wsmen:Items>
(16)       <wsmen:EndOfSequence/> ?
(17)       …
(18)     </wsmen:PullResponse>
(19)   </s:Body>
(20) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse

If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/wsmen:PullResponse/wsmen:EnumerationContext

The optional EnumerationContext element, if present, contains a new XML representation of the current enumeration context. The consumer is required to replace the prior representation with the contents of this element.

/s:Envelope/s:Body/wsmen:PullResponse/wsmen:Items/any

The optional Items element contains one or more enumeration-specific elements, one for each element being returned.

/s:Envelope/s:Body/wsmen:PullResponse/wsmen:EndOfSequence

This optional element indicates that no more elements are available from this enumeration. Additionally, once this element is returned in a Pull response message, subsequent Pull requests using that enumeration context should generate an InvalidEnumerationContext fault message; in any case, they shall not return a valid PullResponse.

At least one of Items or EndOfSequence shall appear. It is possible for both to appear if items are returned and the sequence is exhausted. Similarly, EnumerationContext and EndOfSequence shall not both appear; neither may appear, or one without the other, but not both in the same PullResponse.

The consumer should not issue additional Pull request messages after a Pull response containing an EndOfSequence element has been returned. Similarly, upon receipt of a Pull response containing an EndOfSequence element, the consumer should not issue a Release operation to signal that the enumeration context is no longer needed.

If the consumer does issue a Pull or Release on an invalid enumeration context, the result is undefined: the data source may ignore the request or may return an InvalidEnumerationContext fault, as described previously in this clause, or may take some other action.

Because Pull allows the client to specify a wide range of batching and timing parameters, it is often advisable for the client to know the valid ranges ahead of time. This information can be exported from the service in the form of metadata, which is beyond the scope of this specification. No message-based negotiation is available for discovering the valid ranges of the parameters.

Because wsman:MaxEnvelopeSize can be requested for any response in WS-Management, it is used in the Pull message instead of MaxCharacters, which is generally redundant and preferably is omitted. However, if wsman:MaxEnvelopeSize is present, it has the following characteristics:

R8.4-1: If a service is exposing enumeration operations and supports Pull with the MaxCharacters element, the service should implement MaxCharacters as a general guideline or hint, but may ignore it if wsman:MaxEnvelopeSize is present, because it takes precedence. The service should not fault in the case of a conflict but should observe the wsman:MaxEnvelopeSize value.

R8.4-2: If a service is exposing enumeration operations and supports Pull with the MaxCharacters element, and a single response element would cause the limit to be exceeded, the service may return the single element in violation of the hint. However, the service shall not violate wsman:MaxEnvelopeSize in any case.

A service can send a PullResponse with fewer elements to ensure that the wsman:MaxEnvelopeSize is not exceeded. However, if a single item would cause this to be exceeded, then the rules from 6.2 apply.

In general, MaxCharacters is a hint, and wsman:MaxEnvelopeSize is a strict rule.

R8.4-3: If any fault occurs during a Pull, a compliant service should allow the client to retry Pull with other parameters, such as a larger limit or with no limit, and attempt to retrieve the items. The service should not cancel the enumeration as a whole, but retain enough context to be able to retry if the client so wishes. However, the service may cancel the enumeration outright if an error occurs with an InvalidEnumerationContext fault.

If a fault occurs with a Pull request, the service generally does not need to cancel the entire enumeration, but it can simply freeze the cursor and allow the client to try again.

The EnumerationContext from only the latest response is considered to be valid. Although the service can return the same EnumerationContext values with each Pull, it is not required to do so and can in fact change the EnumerationContext unpredictably.

R8.4-4: A conformant service may ignore MaxTime if wsman:OperationTimeout is also specified, as wsman:OperationTimeout takes precedence. These elements have precisely the same meaning and may be used interchangeably. If both are used, the service should observe only the wsman:OperationTimeout element.

Clients can use wsman:OperationTimeout and wsman:MaxEnvelopeSize rather than MaxTime and MaxCharacters to allow for uniform message construction.

Any fault issued for Pull applies to the Pull message itself, not the underlying enumeration that is in progress. The most recent EnumerationContext is still considered valid, and if the service allows a retry of the most recent Pull message, the client can continue. However, the service can terminate early upon encountering any kind of problem (as specified in R8.4-7).

R8.4-5: This rule intentionally left blank.

If no content is available, the enumerator is still considered active and the Pull message can be retried.

R8.4-6: If a service cannot populate the PullResponse with any items before the timeout, it should return a wsman:TimedOut fault to indicate that true timeout conditions occurred and that the client is not likely to succeed by simply issuing another Pull message. If the service is only waiting for results at the point of the timeout, it should return a response with no items and an updated EnumerationContext, which may have changed, even though no items were returned, as follows:

```
(1) <s:Body>
(2)   <wsmen:PullResponse>
(3)     <wsmen:EnumerationContext> ...possibly updated...
    </wsmen:EnumerationContext>
(4)     <wsmen:Items/>
(5)   </wsmen:PullResponse>
(6) </s:Body>
```

An empty Items block is essentially a directive from the service to try again. If the service faults with a wsman:TimedOut fault, it implies that a retry is not likely to succeed. Typically, the service knows which one to return based on its internal state. For example, on the very first Pull message, if the service is waiting for another component, a wsman:TimedOut fault could be likely. If the enumeration is continuing with no problem and after 50 requests a particular Pull message times out, the service can simply send back zero items in the expectation that the client can continue with another Pull message.

R8.4-7: The service may terminate the entire enumeration early at any time, in which case an InvalidEnumerationContext fault is returned. No further operations are possible, including Release. In specific cases, such as internal errors or responses that are too large, other faults may also be returned. In all such cases, the service should invalidate the enumeration context as well.

R8.4-8: If the EndOfSequence marker occurs in the PullResponse message, the EnumerationContext element shall be omitted, as the enumeration has completed. The client cannot subsequently issue a Release message.

Normally, the end of an enumeration in all cases is reported by the EndOfSequence element being present in the PullResponse content, not through faults. If the client attempts to enumerate past the end of an enumeration, an InvalidEnumerationContext fault is returned. The client need not issue a Release message if the EndOfSequence actually occurs because the enumeration is then completed and the enumeration context is invalid.

R8.4-9: If no MaxElements element is specified, the batch size is 1.

R8.4-10: If the value of MaxElements is larger than the service supports, the service may ignore the value and use any default maximum of its own.

The service can export its maximum MaxElements value in metadata, but the format and location of such metadata is beyond the scope of this specification.

R8.4-11: The EnumerationContext element shall be present in all Pull requests, even if the service uses a constant value for the lifetime of the enumeration sequence.

8.5 Release

The Release operation is initiated by sending a Release request message to the data source. The Release request message shall be of the following form:

```
<s:Envelope …>
  <s:Header …>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release
    </wsa:Action>
    <wsa:MessageID>xs:anyURI</wsa:MessageID>
    <wsa:ReplyTo>wsa:EndpointReference</wsa:ReplyTo>
    <wsa:To>xs:anyURI</wsa:To>
    …
  </s:Header>
  <s:Body …>
    <wsmen:Release …>
      <wsmen:EnumerationContext>…</wsmen:EnumerationContext>
      …
    </wsmen:Release>
  </s:Body>
</s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release

If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

/s:Envelope/s:Body/wsmen:Release/wsmen:EnumerationContext

This required element contains the XML data that represents the enumeration context being abandoned.

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

Upon successful processing of a Release request message, a data source is expected to return a Release response message, which shall adhere to the following form:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)   http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse
(5)      </wsa:Action>
(6)      <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
(7)      <wsa:To>xs:anyURI</wsa:To>
(8)      …
(9)    </s:Header>
(10)   <s:Body />
(11) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

This required element shall contain the value:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse

If a SOAP Action URI is also present in the underlying transport, its value shall convey the same value.

Release is used only to perform an early cancellation of the enumeration. In cases in which it is not actually needed, the implementation can expose a dummy implementation that always succeeds. This promotes uniform client-side messaging.

R8.5-1: The service shall recognize and process the Release message if the enumeration is terminated early. If an EndOfSequence marker occurs in a PullResponse message, the enumerator is already completed and a Release message cannot be issued because no up-to-date EnumerationContext exists.

R8.5-2: The client may fail to deliver the Release message in a timely fashion or may never send it. A conformant service may terminate the enumeration after a suitable idle time has expired, and any attempt to reuse the enumeration context shall result in an InvalidEnumerationContext fault.

R8.5-3: This rule intentionally left blank.

R8.5-4: The service may accept a Release message asynchronously to any Pull requests already in progress and cancel the enumeration. The service may refuse such an asynchronous request and fault it with a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AsynchronousRequest

The service may also queue or block the request and serialize it so that it is processed after the Pull message.

In most cases, it is desirable to be able to asynchronously cancel an outstanding Pull message. This capability requires the service to be able to receive the Release message asynchronously while still processing a pending Pull message. Further, it requires that the EnumerationContext element contain information that is constant between Pull operations.

NOTE: If the value of EnumerationContext is a simple increasing integer, Release always uses a previous value, so the service may consider it to be invalid. If the EnumerationContext element contains a value that is constant across Pull requests (as well as any other information that the service might need), the service can more easily implement the cancellation.

8.6 Ad-Hoc Queries and Fragment-Level Enumerations

As discussed in 7.7, it is desirable that clients be able to access subsets of a representation. This is especially important in the area of query processing, where users routinely want to execute XPath or XQuery operations over the representation to receive ad-hoc results.

Because SOAP messages need to conform to known schemas, and ad-hoc queries return results that are dynamically generated and might conform to no schema, the wsman:XmlFragment wrapper from 7.7 is used to wrap the responses.

R8.6-1: The service may support ad-hoc compositional queries, projections, or enumerations of fragments of the representation objects by supplying a suitable dialect in the wsman:Filter. The resulting set of Items in the PullResponse element (or EnumerateResponse element if OptimizedEnumeration is used) should be wrapped with wsman:XmlFragment wrappers as follows:

```
(1) <s:Body>
(2)    <wsmen:PullResponse>
(3)      <wsmen:EnumerationContext> ..possibly updated..
   </wsmen:EnumerationContext>
(4)    <wsmen:Items>
(5)      <wsman:XmlFragment>
(6)        XML content
(7)      </wsman:XmlFragment>
(8)      <wsman:XmlFragment>
(9)        XML content
(10)       </wsman:XmlFragment>
(11)       ...
(12)     </wsmen:Items>
(13)   </wsmen:PullResponse>
(14) </s:Body>
```

The schema for wsman:XmlFragment contains a directive to suppress schema validation, allowing a validating parser to accept ad-hoc content produced by the query processor acting behind the enumeration.

XPath 1.0 and XQuery 1.0 already support returning subsets or compositions of representations, so they are suitable for use in this regard.

R8.6-2: If the service does not support fragment-level enumeration, it should return a wsmen:FilterDialectRequestedUnavailable fault, the same as for any other unsupported dialect.

The XPath expression used for filtering is still as described in the Enumeration clauses (see 8.2, 8.2.2, 8.2.3). The wsman:XmlFragment wrappers are applied after the XPath is evaluated to prevent schema violations if the XPath selects node sets that are fragments and not legal according to the original schema.

8.7 Enumeration of EPRs

Typically, inferring the EPR of an enumerated object simply by inspection is not possible. In many cases, it is desirable to enumerate the EPRs of objects rather than the objects themselves. Such EPRs can be usable in subsequent Get or Delete requests, for example. Similarly, it is often desirable to enumerate both the objects and the associated EPRs.

The default behavior for Enumerate is as defined in 8.1. However, WS-Management provides an additional extension for controlling the output of the enumeration.

R8.7-1: A service may optionally support the wsman:EnumerationMode modifier element with a value of *EnumerateEPR*, which returns only the EPRs of the objects as the result of the enumeration.

EXAMPLE 1:

```
<s:Envelope ...>
  <s:Header>
    ...
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
    </wsa:Action>
    ...
  </s:Header>
  <s:Body>
    <wsmen:Enumerate>
      <wsman:Filter Dialect="...">  filter </wsman:Filter>
      <wsman:EnumerationMode> EnumerateEPR </wsman:EnumerationMode>
      ...
    </wsmen:Enumerate>
  </s:Body>
</s:Envelope>
```

EXAMPLE 2: The hypothetical response would appear as in the following example:

```
<s:Body>
  <wsmen:PullResponse>
    <wsmen:Items>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      ...
    </wsmen:Items>
  </wsmen:PullResponse>
</s:Body>
```

The filter, if any, is still applied to the enumeration, but the response contains only the EPRs of the items that would have been returned. These EPRs are intended for use in subsequent Get operations.

R8.7-2: A service may optionally support the wsman:EnumerationMode modifier with the value of *EnumerateObjectAndEPR*. If present, the enumerated objects are wrapped in a wsman:Item element that juxtaposes two XML representations: the payload representation followed by the associated wsa:EndpointReference.

EXAMPLE 3: The wsman:EnumerationMode example appears as follows:

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

```
(4)      http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
(5)    </wsa:Action>
(6)  </s:Header>
(7)  <s:Body>
(8)    <wsmen:Enumerate>
(9)      <wsman:Filter Dialect="...">  filter </wsman:Filter>
(10)       <wsman:EnumerationMode> EnumerateObjectAndEPR
   </wsman:EnumerationMode>
(11)       ...
(12)   </wsmen:Enumerate>
(13) </s:Body>
```

EXAMPLE 4: The response appears as follows:

```
(1)  <s:Body>
(2)    <wsmen:PullResponse>
(3)      <wsmen:Items>
(4)        <wsman:Item>
(5)          <PayloadObject xmlns="..."> ... </PayloadObject>  <!-- Object -->
(6)          <wsa:EndpointReference> ... </wsa:EndpointReference>   <!-- EPR -->
(7)        </wsman:Item>
(8)        <wsman:Item>
(9)          <PayloadObject xmlns="..."> ... </PayloadObject>  <!-- Object -->
(10)           <wsa:EndpointReference> ... </wsa:EndpointReference> <!-- EPR -->
(11)       </wsman:Item>
(12)       ...
(13)     </wsmen:Items>
(14)   </wsmen:PullResponse>
(15) </s:Body>
```

In the preceding example, each item is wrapped in a wsman:Item wrapper (line 8), which itself contains the representation object (line 9) followed by its EPR (line 10). As many wsman:Item objects may be present as is consistent with other encoding limitations.

R8.7-3: If a service does not support the wsman:EnumerationMode modifier, it shall return a fault of wsman:UnsupportedFeature with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EnumerationMode

8.8 Renew

To renew an enumeration, the consumer sends a request of the following form to the data source:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew
(5)      </wsa:Action>
(6)      <wsa:MessageID>xs:anyURI</wsa:MessageID>
(7)      <wsa:FaultTo>endpoint-reference</wsa:FaultTo> ?
(8)      <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
(9)      <wsa:To>xs:anyURI</wsa:To>
(10)     …
(11)   </s:Header>
(12)   <s:Body …>
(13)     <wsmen:Renew …>
(14)       <wsmen:EnumerationContext>…</wsmen:EnumerationContext>
(15)       <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
(16)       …
(17)     </wsmen:Renew>
```

```
  </s:Body>
</s:Envelope>
```

Components of the preceding outline are additionally constrained as for a request to create an enumeration with the following addition(s):

/s:Envelope/s:Body/*/wsmen:EnumerationContext

This required element contains the XML data that represents the current enumeration context.

If the enumeration context is not valid, either because it has been replaced in the response to another Pull request, or because it has completed (EndOfSequence has been returned in a Pull response), or because it has been Released, or because it has expired, or because the data source has had to invalidate the context, then the data source should fail the request, and may generate a wsmen:InvalidEnumerationContext fault.

The data source may not be able to determine that an enumeration context is not valid, especially if all of the state associated with the enumeration is kept in the enumeration context and refreshed on every PullResponse.

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

If the data source accepts a request to renew an enumeration, it shall reply with a response of the following form:

```
<s:Envelope …>
  <s:Header …>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse
    </wsa:Action>
    <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
    <wsa:To>xs:anyURI</wsa:To>
    …
  </s:Header>
  <s:Body …>
    <wsmen:RenewResponse …>
      <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
      <wsmen:EnumerationContext>…</wsmen:EnumerationContext> ?
      …
    </wsmen:RenewResponse>
  </s:Body>
</s:Envelope>
```

Components of the preceding outline listed are constrained as for a response to an Enumerate request with the following addition:

/s:Envelope/s:Body/wsmen:RenewResponse/wsmen:Expires

If the requested expiration is a duration, then the implied start of that duration is the time when the data source starts processing the Renew request.

/s:Envelope/s:Body/wsmen:RenewResponse/wsmen:EnumerationContext

This element is optional in this response.

If the data source chooses not to renew this enumeration, the request shall fail, and the data source should generate a wsmen:UnableToRenew fault indicating that the renewal was not accepted.

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

8.9 GetStatus

To get the status of an enumeration, the subscriber sends a request of the following form to the data source:

```
<s:Envelope …>
  <s:Header …>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus
    </wsa:Action>
    <wsa:MessageID>xs:anyURI</wsa:MessageID>
    <wsa:FaultTo>endpoint-reference</wsa:FaultTo> ?
    <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
    <wsa:To>xs:anyURI</wsa:To>
    …
  </s:Header>
  <s:Body …>
    <wsmen:GetStatus …>
      <wsmen:EnumerationContext>…</wsmen:EnumerationContext> ?
      …
    </wsmen:GetStatus>
  </s:Body>
</s:Envelope>
```

Components of the preceding outline are additionally constrained as for a request to renew an enumeration. Other components of the preceding outline are not further constrained by this specification.

If the enumeration is valid and has not expired, the data source shall reply with a response of the following form:

```
<s:Envelope …>
  <s:Header …>
    <wsa:Action>
 http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse
    </wsa:Action>
    <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
    <wsa:To>xs:anyURI</wsa:To>
    …
  </s:Header>
  <s:Body …>
    <wsmen:GetStatusResponse …>
      <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
      …
    </wsmen:GetStatusResponse>
  </s:Body>
</s:Envelope>
```

Components of the preceding outline are constrained as for a response to a Renew request. Other components of the preceding outline are not further constrained by this specification.

8.10 EnumerationEnd

If the data source terminates an enumeration unexpectedly, the data source should send an EnumerationEnd SOAP message to the endpoint reference indicated when the enumeration was created. The message shall be of the following form:

```
<s:Envelope …>
  <s:Header …>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd
```

```
(5)      </wsa:Action>
(6)      <wsa:To>xs:anyURI</wsa:To>
(7)      …
(8)    </s:Header>
(9)    <s:Body …>
(10)     <wsmen:EnumerationEnd …>
(11)       <wsmen:EnumerationContext>…</wsmen:EnumerationContext>
(12)       <wsmen:Code>
(13)         [
(14)  http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceShuttingDown
(15)  | http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceCancelling
(16)         ]
(17)       </wsmen:Code>
(18)       <wsmen:Reason xml:lang="language identifier" >
(19)         xs:string
(20)       </wsmen:Reason> ?
(21)       …
(22)     </wsmen:EnumerationEnd>
(23)   </s:Body>
(24) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Body/wsmen:Release/wsmen:EnumerationContext

This required element contains the XML data that represents the enumeration context being terminated. It is recommended that consumers DO NOT attempt to compare this element against any collection of wsmen:EnumerationContext elements for purposes of correlation, because that requires the ability to compare arbitrary XML elements. If consumers wish to correlate this message against their outstanding contexts, it is recommend that they use the reference parameters of the /wsmen:Enumerate/wsmen:EndTo EPR.

/s:Envelope/s:Body/wsmen:EnumerationEnd/wsmen:Code = "http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceShuttingDown"

This value shall be used if the data source terminated the enumeration because the source is being shut down in a controlled manner; that is, if the data source is being shut down but has the opportunity to send an EnumerationEnd message before it exits.

/s:Envelope/s:Body/wsmen:EnumerationEnd/wsmen:Code = "http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceCancelling"

This value shall be used if the data source terminated the enumeration for some other reason before it expired.

/s:Envelope/s:Body/wsmen:EnumerationEnd/wsmen:Reason

This optional element contains text, in the language specified by the @xml:lang attribute, describing the reason for the unexpected enumeration termination.

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

9 Custom Actions (Methods)

Custom actions, or "methods," are ordinary SOAP messages with unique Actions. An implementation can support resource-specific methods in any form, subject to the addressing model and restrictions described in clause 5 of this specification.

R9-1: A conformant service may expose any custom actions or methods.

R9-2: If custom methods are exported, Addressing rules, as described elsewhere in this specification, shall be observed, and each custom method shall have a unique wsa:Action.

R9-3: If a request does not contain the correct parameters for the custom action, the service may return a wsman:InvalidParameter fault. Fault details for incorrect type and incorrect name may also be included.

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch	(incorrect type)
http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName	(incorrect name)

As defined by Addressing, the Action URI is used to describe the semantics of the operation and the wsa:To element describes the destination of the message. A custom method thus has a dedicated Addressing Action URI.

Because options are a parameterization technique for message types that are not user-extensible, such as the resource access operations, they are not appropriate for use as a custom method or combined with a custom method. Custom operations defined in a WSDL document define any required parameters and thus expose naming and type checking in a stringent way. Mixing wsman:OptionSet with a strongly typed WSDL operation is likely to lead to confusion.

10 Notifications (Eventing)

10.1 General

Management infrastructures often want to receive messages when events occur in remote management 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 changes over time. This specification defines a set of operations for one management Web service (called a "subscriber") to register interest (called a "subscription") with another management 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.

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 (for example, [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.

If a managed entity emits events, it can publish those events using this publish-and-subscribe mechanism and paradigms.

R10.1-1: If a resource can emit events and allows clients to subscribe to and receive notification messages, it shall do so by implementing the operations as specified in this clause.

R10.1-2: If the eventing mechanism as described in this clause is supported, the wsme:Subscribe, wsme:Renew, and wsme:Unsubscribe messages shall be supported. The wsme:SubscriptionEnd message is optional. The wsme:GetStatus message in a constrained environment is a candidate for exclusion. If this message is not supported, then a wsa:ActionNotSupported fault shall be returned in response to this request.

10.2 Subscribe

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 includes 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 ReferenceParameters) 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 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.

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 …>
    <wsme:Subscribe …>
      <wsme:EndTo>endpoint-reference</wsme:EndTo> ?
      <wsme:Delivery Mode="xs:anyURI"? >xs:any</wsme:Delivery>
      <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
      <wsme:Filter Dialect="xs:anyURI"? > xs:any </wsme:Filter> ?
      …
    </wsme:Subscribe>
  </s:Body>
</s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Header/wsa:Action

If a SOAP Action URI is used in the binding for SOAP, the value indicated herein shall be used for that URI.

/s:Envelope/s:Body/*/wsme:EndTo

Where to send a SubscriptionEnd message if the subscription is terminated unexpectedly. If present, this element shall be of type wsa:EndpointReferenceType. The default is not to send this message. The endpoint referenced by this EPR shall implement a binding of the "EndToEndpoint" portType described in ANNEX I.

/s:Envelope/s:Body/*/wsme:Delivery

A delivery destination for notification messages, using some delivery mode.

/s:Envelope/s:Body/*/wsme: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.

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

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

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

/s:Envelope/s:Body/*/wsme: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 shall fail, and the event source may generate a 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 an UnsupportedExpirationType fault indicating that an unsupported expiration type was requested.

/s:Envelope/s:Body/*/wsme:Filter

A Boolean expression in some dialect, either as a string or as an XML fragment. If the expression evaluates to false for a notification, the notification shall 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 shall fail, and the event source may generate a wsme:FilteringNotSupported fault indicating that filtering is not supported.

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

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

Value of /s:Envelope/s:Body/*/wsme: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/*/wsme:Filter

Other message information headers defined by Addressing may be included in the request and response messages, according to the usage and semantics defined in Addressing.

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

If the event source accepts a request to create a subscription, it shall 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 …>
    <wsme:SubscribeResponse …>
      <wsme:SubscriptionManager>
        wsa:EndpointReferenceType
      </wsme:SubscriptionManager>
      <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires>
      …
    </wsme:SubscribeResponse>
  </s:Body>
</s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/S:Header/wsa:RelatesTo

Shall be the value of the wsa:MessageID of the corresponding request.

/s:Envelope/s:Body/*/wsme: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, Identifier of type xs:anyURI, that may be used as a distinguishing reference parameter if desired by the event source.

/s:Envelope/s:Body/*/wsme: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 preceding outline are not further constrained by this specification.

If the event source chooses not to accept a subscription, the request shall fail, and the event source may generate a wsme:EventSourceUnableToProcess fault indicating that the request was not accepted.

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/wsme:Subscribe/wsme:NotifyTo/wsa:ReferenceParameters elements; per Addressing, the event source shall include each such literal SOAP header block in every notification sent to the endpoint addressed by /s:Envelope/s:Body/wsme:Subscribe/wsme:NotifyTo.

10.2.1 General

WS-Management uses Subscribe substantially as documented here, except that the WS-Management default addressing model is incorporated as described in 5.1.

R10.2.1-1: The identity of the event source shall be based on the Addressing EPR.

R10.2.1-2: If the service cannot support the requested addressing, it should return a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode

Verifying that the address is usable allows errors to be detected at the time the subscription is created. For example, if the address cannot be reached due to firewall configuration and the service can detect this, telling the client allows for it to be corrected immediately.

R10.2.1-3: Because many delivery modes require a separate connection to deliver the event, the service should comply with the security profiles defined in clause 11 of this specification, if HTTP or HTTPS is used to deliver events. If no security is specified, the service may attempt to use default security mechanisms, or return a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress

Because clients might need to have client-side context sent back with each event delivery, the NotifyTo address in the Delivery block can be used for this purpose. This NotifyTo EPR can contain any number of client-defined reference parameters.

R10.2.1-4: A service may validate the address by attempting a connection while the Subscribe request is being processed to ensure delivery can occur successfully. If the service determines that the address is not valid or permissions cannot be acquired, it should emit a wsman:EventDeliverToUnusable fault.

This situation can occur when the address is incorrect or when the event source cannot acquire permissions to deliver events properly.

R10.2.1-5: Any reference parameters supplied in the NotifyTo address shall be included with each event delivery as top-level headers as specified 5.4. If EndTo is supported, this behavior applies as well.

When the default addressing model is used by the service, the ResourceURI is often used to reference the logical event source, and selector values can additionally be used to indicate a real or virtual log within the scope of that source, or might even be used to limit the types or groups of events available. This action can logically overlap with the Filter mechanism in the subscription body itself, so due consideration should be given to the interplay among the address of the event source, the types of events it can publish, and the subscription-level filtering.

If a client needs to have events delivered to more than one destination, more than one subscription is required.

R10.2.1-6: If the events contain localized content, the service should accept a subscription with a wsman:Locale block acting as a hint (see 6.3) within the Delivery block of the Subscribe message. The language is encoded in an xml:lang attribute using RFC 5646 language codes.

The service attempts to localize any descriptive content to the specified language when delivering such events, which is outlined as follows:

```
(1)  <wsme:Subscribe>
(2)    <wsme:Delivery>
(3)      <wsme:NotifyTo>  ... </wsme:NotifyTo>
(4)      <wsman:Locale xml:lang="language-code"/>
(5)    </wsme:Delivery>
(6)  </wsme:Subscribe>
```

NOTE: In this context, the wsman:Locale element (defined in 6.3) is not a SOAP header and mustUnderstand cannot be used.

R10.2.1-7: The service should accept a subscription with a wsman:ContentEncoding block within the Delivery block of the Subscribe message. This block acts as a hint to indicate how the delivered events are to be encoded. The two standard xs:language tokens defined for this purpose are "UTF-8" or "UTF-16", although other encoding formats may be specified if necessary. The service should attempt to encode the events using the requested language token, as in the following example:

EXAMPLE:

```
(1) <wsme:Subscribe>
(2)  <wsme:Delivery>
(3)    ...
```

```
(4)    <wsme:NotifyTo>  ... </wsme:NotifyTo>
(5)    <wsman:ContentEncoding> UTF-16 </wsman:ContentEncoding>
(6)  </wsme:Delivery>
(7) </wsme:Subscribe>
```

10.2.2 Filtering

Filter expression is constrained to be a Boolean predicate. To support ad hoc queries including projections, WS-Management defines a wsman:Filter element of exactly the same form as what is used in the Subscribe operation except that the filter expression is not constrained to be a Boolean predicate. This allows the use of subscriptions using existing query languages such as SQL and CQL, which combine predicate and projection information in the same syntax. The use of projections is defined by the filter dialect, not by WS-Management.

If the filter dialect for either Filter or wsman:Filter used for the Subscribe message is http://www.w3.org/TR/1999/REC-xpath-19991116 (the default dialect in both cases), the context node is the SOAP Envelope element.

WS-Management defines the wsman:Filter element as a child of the Subscribe element.

WS-Management defines the wsman:Filter element to allow projections, which is outlined as follows:

```
(1)  <wsman:Filter Dialect="xs:anyURI"?> xs:any </wsman:Filter>
```

The Dialect attribute is optional. When not specified, it has the following implied value:

http://www.w3.org/TR/1999/REC-xpath-19991116

This dialect allows any full XPath expression or subset to be used.

R10.2.2-1: If a service supports filtered subscriptions using Filter, it shall also support filtering using wsman:Filter. This rule allows client stacks to always pick the wsman XML namespace for the Filter element. Even though a service supports wsman:Filter, it is not required to support projections.

R10.2.2-2: If a service supports filtered subscriptions using wsman:Filter, it should also support filtering using Filter.

R10.2.2-3: If a Subscribe request contains both Filter and wsman:Filter, the service shall return a wsa:InvalidMessage fault.

To allow eventing filter expressions to be defined independently of the delivery mode, WS-Management defines a new filter dialect that is the same as previously defined except that the context node is defined as the element that would be returned as the first child of the SOAP Body element if the Push delivery mode were used. The URI for this filter dialect is:

http://schemas.dmtf.org/wbem/wsman/1/wsman/filter/eventRootXPath

The context node for this expression is as follows:

- **Context Node**: any XML element that could be returned as a direct child of the s:Body element if the delivery mode was Push
- **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/wsme:Subscribe/wsman:Filter

R10.2.2-4: Services should support this filter dialect when they want to use an XPath-based filter, rather than the default filter dialect defined in 10.2.1.

The considerations described in 8.3 regarding the XPath 1.0 filter dialect also apply to the preceding eventing filter.

Resource-constrained implementations might have difficulty providing full XPath processing and yet still want to use a subset of XPath syntax. This does not require the addition of a new dialect if the expression specified in the filter is a true XPath expression. The use of the filter dialect URI does not imply that the service supports the entire specification for that dialect, only that the expression conforms to the rules of that dialect. Most services use XPath only for filtering, but they will not support the composition of new XML or removing portions of XML that would result in the XML fragment violating the schema of the event.

EXAMPLE 1: A typical example of the use of XPath in a subscription follows. Assume that each event that would be delivered has the following XML content:

```
(1)  <s:Body>
(2)    <LowDiskSpaceEvent xmlns="...">
(3)      <LogicalDisk>C:</LogicalDisk>
(4)      <CurrentMegabytes>12</CurrentMegabytes>
(5)      <Megabytes24HoursAgo>17</Megabytes24HoursAgo>
(6)    </LowDiskSpaceEvent>
(7)  </s:Body>
```

The event is wholly contained within the s:Body of the SOAP message. The anchor point for the XPath evaluation is the first element of each event, and it does not reference the <s:Body> element as such. The XPath expression is evaluated as if the event content were a separate XML document.

EXAMPLE 2: When used for simple document processing, the following four XPath expressions "select" the entire <LowDiskSpaceEvent> node:

```
(8)  /
(9)  /LowDiskSpaceEvent
(10)  ../LowDiskSpaceEvent
(11)  .
```

If used as a "filter", this XPath expression does not filter out any instances and is the same as selecting all instances of the event, or omitting the filter entirely.

EXAMPLE 3: However, using the following syntax, the XPath expression selects the XML node only if the test expression in brackets evaluates to logical "true":

```
(1)  ../LowDiskSpaceEvent[LogicalDisk="C:"]
```

In this case, the event is selected if it refers to disk drive "C:"; otherwise the XML node is not selected. This XPath expression would filter out all <LowDiskSpaceEvent> events for other drives.

EXAMPLE 4: Full XPath implementations may support more complex test expressions:

```
(1)  ../LowDiskSpaceEvent[LogicalDisk="C:" and CurrentMegabytes < "20"]
```

In essence, the XML form of the event is logically passed through the XPath processor to see if it would be selected. If so, it is delivered as an event. If not, the event is discarded and not delivered to the subscriber.

XPath 1.0 can be used simply for filtering or to send back subsets of the representation (or even the values without XML wrappers). In cases where the result is not just filtered but is "altered," the technique in 8.6 applies.

If full XPath cannot be supported, a common subset for this purpose is described in ANNEX D of this specification.

R10.2.2-5: The wsman:Filter element shall contain either simple text or a single XML element of a single or complex type. A service should reject any filter with mixed content or multiple peer XML elements using a wsme:EventSourceUnableToProcess fault.

R10.2.2-6: A conformant service may not support the entire syntax and processing power of the specified filter dialect. The only requirement is that the specified filter is syntactically correct within the definition of the dialect. Subsets are therefore legal. If the specified filter exceeds the capability of the service, the service should return a wsman:CannotProcessFilter fault with text explaining why the filter was problematic.

R10.2.2-7: If a service requires complex initialization parameters in addition to the filter, these should be part of the wsman:Filter block because they logically form part of the filter initialization, even if some of the parameters are not strictly used in the filtering process. In this case, a unique dialect URI shall be devised for the event source and the schema and usage published.

R10.2.2-8: If the service supports composition of new XML or filtering to the point where the resultant event would not conform to the original schema for that event, the event delivery should be wrapped in the same way as content for the fragment-level access operations (see 7.7).

Events, regardless of how they are filtered or reduced, need to conform to some kind of XML schema definition when they are actually delivered. Simply sending out unwrapped XML fragments during delivery is not legal.

R10.2.2-9: If the service requires specific initialization XML in addition to the filter to formulate a subscription, this initialization XML shall form part of the filter body and be documented as part of the filter dialect.

This rule promotes a consistent location for initialization content, which may be logically seen as part of the filter. The filter XML schema is more understandable if it separates the initialization and filtering parts into separate XML elements.

For information about filtering over enumerations, see 8.3.

10.2.3 Connection Retries

Due to the nature of event delivery, the subscriber might not be reachable at event-time. Rather than terminate all subscriptions immediately, typically the service attempts to connect several times with suitable timeouts before giving up.

R10.2.3-1: A service may observe any connection retry policy or allow the subscriber to define it by including the following wsman:ConnectionRetry element in a subscription. If the service does not accept the wsman:ConnectionRetry element, it should return a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DeliveryRetries

This only applies to failures to *connect* and does not include replay of actual SOAP deliveries.

```
(1)  <wsme:Subscribe>
(2)    <wsme:Delivery>
(3)      <wsme:NotifyTo>  ... </wsme:NotifyTo>
```

```
(4)      <wsman:ConnectionRetry Total="count"> xs:duration
   </wsman:ConnectionRetry>
(5)    </wsme:Delivery>
(6)  </wsme:Subscribe>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsman:ConnectionRetry
  an xs:duration for how long to wait between retries while trying to connect

wsman:ConnectionRetry/@Total
  how many retries to attempt, observing the specified interval between the attempts

R10.2.3-2: If the retry counts are exhausted, the subscription should be considered abnormally terminated.

The retry mechanism applies only to attempts to connect. Failures to deliver on an established connection can result in terminating the connection according to the rules of the transport in use, and terminating the subscription. Other Web services mechanisms can be used to synthesize reliable delivery or safe replay of the actual deliveries.

10.2.4 SubscribeResponse

The service returns any service-specific reference parameters in the SubscriptionManager EPR, and these are included by the subscriber (client) later when issuing Unsubscribe and Renew messages.

R10.2.4-1: In SubscribeResponse, the service may specify any EPR for the SubscriptionManager. However, it is recommended that the address contain the same wsa:To address as the original Subscribe request and differ only in other parts of the address, such as the reference parameters.

R10.2.4-2: A conformant service may not return the Expires field in the response, but, as specified in 10.2, this implies that the subscription does not expire until explicitly canceled.

10.2.5 Heartbeats

A typical problem with event subscriptions is a situation in which no event traffic occurs. It is difficult for clients to know whether no events matching the subscription have occurred or whether the subscription has simply failed and the client was not able to receive any notification.

Because of this, WS-Management defines a "heartbeat" pseudo-event that can be sent periodically for any subscription. This event is sent if no regular events occur so that the client knows the subscription is still active. If the heartbeat event does not arrive, the client knows that connectivity is bad or that the subscription has expired, and it can take corrective action.

The heartbeat event is sent *in place of* the events that would have occurred and is *never* intermixed with "real" events. In all modes, including batched, it occurs alone.

To request heartbeat events as part of a subscription, the Subscribe request has an additional field in the Delivery section:

```
(1)  <wsme:Delivery>
(2)    ...
(3)    <wsman:Heartbeats> xs:duration </wsman:Heartbeats>
(4)    ...
(5)  </wsme:Delivery>
```

wsman:Heartbeats specifies that heartbeat events are added to the event stream at the specified interval.

R10.2.5-1: A service should support heartbeat events. If the service does not support them, it shall return a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats

Heartbeats apply to all delivery modes.

Heartbeats apply to "pull" mode deliveries as well, in that they are a hint to the publisher about how often to expect a Pull request. The service can refuse to deliver events if the client does not regularly call back at the heartbeat interval. If no events are available at the heartbeat interval, the service simply includes a heartbeat event as the result of the Pull.

R10.2.5-2: While a subscription with heartbeats is active, the service shall ensure that either real events or heartbeats are sent out within the specified wsman:Heartbeat interval. The service may send out heartbeats at this interval in addition to the events, as long as the heartbeat events are sent separately (not batched with other events). The goal is to ensure that some kind of event traffic always occurs within the heartbeat interval.

R10.2.5-3: A conformant service may send out heartbeats at earlier intervals than specified in the subscription. However, the events should not be intermixed with other events when batching delivery modes are used. Typically, heartbeats are sent out *only when no real events occur.* A service may fail to produce heartbeats at the specified interval if real events have been delivered.

R10.2.5-4: A conformant service shall not send out heartbeats asynchronously to any event deliveries already in progress. They shall be delivered in sequence like any other events, although they are delivered alone as single events or as the only event in a batch.

In practice, heartbeat events are based on a countdown timer. If no events occur, the heartbeat is sent out alone. However, every time a real event is delivered, the heartbeat countdown timer is reset. If a steady stream of events occurs, heartbeats might never be delivered.

Heartbeats need to be acknowledged like any other event if one of the acknowledged delivery modes is in effect.

The client assumes that the subscription is no longer active if no heartbeats are received within the specified interval, so the service can proceed to cancel the subscription and send any requested SubscriptionEnd messages, because the client will likely resubscribe shortly. Used in combination with bookmarks (see 10.2.6), heartbeats can achieve highly reliable delivery with known latency behavior.

The heartbeat event itself is simply an event message with no body and is identified by its wsa:Action URI as follows:

```
(1)  <s:Envelope ...>
(2)    <s:Header>
(3)      <wsa:To> .... </wsa:To>
(4)      <wsa:Action s:mustUnderstand="true">
(5)        http://schemas.dmtf.org/wbem/wsman/1/wsman/Heartbeat
(6)      </wsa:Action>
(7)      ...
(8)    </s:Header>
(9)    <s:Body/>
(10) </s:Envelope>
```

10.2.6 Bookmarks

Reliable delivery of events is difficult to achieve, so management subscribers need to have a way to be certain of receiving all events from a source. When subscriptions expire or when deliveries fail, windows of time can occur in which the client cannot be certain whether critical events have occurred. Rather than using a highly complex, transacted delivery model, WS-Management defines a simple mechanism for ensuring that all events are delivered or that dropped events can be detected.

This mechanism requires event sources to be backed by logs, whether short-term or long-term. The client subscribes in the same way as a normal Subscribe operation, and specifies that bookmarks are to be used. The service then sends a new bookmark with each event delivery, which the client is responsible for persisting. This bookmark is essentially a context or a pointer to the logical event stream location that matches the subscription filter. As each new delivery occurs, the client updates the bookmark in its own space. If the subscription expires or is terminated unexpectedly, the client can subscribe again, using the last known bookmark. In essence, the subscription filter identifies the desired set of events, and the bookmark tells the service where to start in the log. The client may then pick up where it left off.

This mechanism is immune to transaction problems, because the client can simply start from any of several recent bookmarks. The only requirement for the service is to have some type of persistent log in which to apply the bookmark. If the submitted bookmark is too old (temporally or positionally within the log), the service can fault the request, and at least the client reliably knows that events have been dropped.

R10.2.6-1: A conformant service may support the WS-Management bookmark mechanism. If the service does not support bookmarks, it should return a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Bookmarks

To request bookmark services, the client includes the wsman:SendBookmarks element in the Subscribe request as follows:

```
<s:Body>
  <wsme:Subscribe>
    <wsme:Delivery>
      ...
    </wsme:Delivery>
    <wsman:SendBookmarks/>
  </wsme:Subscribe>
</s:Body>
```

wsman:SendBookmarks instructs the service to send a bookmark with each event delivery. Bookmarks apply to all delivery modes.

The bookmark is a token that represents an abstract pointer in the event stream, but whether it points to the last delivered event or the last event plus one (the upcoming event) makes no difference because the token is supplied to the same implementation during a subsequent Subscribe operation. The service can thus attach any service-specific meaning and structure to the bookmark with no change to the client.

If bookmarks are requested, each event delivery contains a new bookmark value as a SOAP header, as shown in the following outline. The format of the bookmark is entirely determined by the service and is treated as an opaque value by the client.

```
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
```

```
(5)   <s:Header>
(6)     <wsa:To s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
(7)     ...
(8)     <wsman:Bookmark> xs:any </wsman:Bookmark>
(9)     ...
(10)  </s:Header>
(11)  <s:Body>
(12)    ...event content...
(13)  </s:Body>
(14) </s:Envelope>
```

wsman:Bookmark contains XML content supplied by the service that indicates the logical position of this event or event batch in the event stream implied by the subscription.

R10.2.6-2: If bookmarks are supported, the wsman:Bookmark element content shall be either simple text or a single complex XML element. A conformant service shall not accept mixed content of both text and elements, or multiple peer XML elements, under the wsman:Bookmark element.

R10.2.6-3: If bookmarks are supported, the service shall use a wsman:Bookmark element in the header to send an updated bookmark with each event delivery. Bookmarks accompany only event deliveries and are not part of any SubscriptionEnd message.

After the subscription has terminated, for whatever reason, a subsequent Subscribe message on the part of the client can include the bookmark in the subscription request. The service then knows where to start.

The last-known bookmark received by the client is added to the Subscribe message as a new block, positioned after the child elements of Subscribe, as in the following outline:

```
(1)  <s:Body>
(2)    <wsme:Subscribe>
(3)      <wsme:Delivery>  ... </wsme:Delivery>
(4)      <wsme:Expires> ... </wsme:Expires>
(5)      <wsman:Filter> ... </wsman:Filter>
(6)      <wsman:Bookmark>
(7)        ...last known bookmark from a previous delivery...
(8)      </wsman:Bookmark>
(9)      <wsman:SendBookmarks/>
(10)   </wsme:Subscribe>
(11) </s:Body>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsman:Bookmark

arbitrary XML content previously supplied by the service as a wsman:Bookmark during event deliveries from a previous subscription

wsman:SendBookmarks

an instruction to continue delivering updated bookmarks with each event delivery

R10.2.6-4: The bookmark is a pointer to the last event delivery or batched delivery. The service shall resume delivery at the first event or events after the event represented by the bookmark. The service shall not replay events associated with the bookmark or skip any events since the bookmark.

R10.2.6-5: The service may support a short queue of previous bookmarks, allowing the subscriber to start using any of several previous bookmarks. If bookmarks are supported, the service is required only to support the most recent bookmark for which delivery had apparently succeeded.

R10.2.6-6: If the bookmark cannot be honored, the service shall fault with a wsman:InvalidBookmark fault with one of the following detail codes:

- bookmark has expired (the source is not able to back up and replay from that point):

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired

- format is unknown:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFormat

If multiple new subscriptions are made using a previous bookmark, the service can allow multiple reuse or may limit bookmarks to a single subscriber, and can even restrict how long bookmarks can be used before becoming invalid.

The following predefined, reserved bookmark value indicates that the subscription starts at the earliest possible point in the event stream backed by the publisher:

http://schemas.dmtf.org/wbem/wsman/1/wsman/bookmark/earliest

If a subscription is received with this bookmark, the event source replays all possible events that match the filter and any events that subsequently occur for that event source. The absence of any bookmark means "begin at the next available event".

R10.2.6-7: A conformant service may support the reserved bookmark http://schemas.dmtf.org/wbem/wsman/1/wsman/bookmark/earliest and not support any other type of bookmark. If the http://schemas.dmtf.org/wbem/wsman/1/wsman/bookmark/earliest bookmark is supported, the event source should send all previous and future events that match the filter starting with the earliest such event.

10.2.7 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 require event messages to be wrapped in a standard "event" SOAP envelope, while others 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.

A WS-Management implementation can support a variety of event delivery modes.

In essence, delivery consists of the following items:

- a delivery mode (how events are packaged)
- an address (the transport and network location)
- an authentication profile to use when connecting or delivering the events (security)

The standard security profiles are discussed in clause 12 and may be required for subscriptions if the service needs hints or other indications of which security model to use at event-time.

If the delivery mode is supported but not actually usable due to firewall configuration, the service can return a wsme:DeliveryModeRequestedUnavailable fault with additional detail to this effect.

R10.2.7-1: For any given transport, a conformant service should support at least one of the following delivery modes to interoperate with standard clients:

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

http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck

http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull

The delivery mode does *not* imply any specific transport.

Modes describe SOAP message behavior and are unrelated to the transport that is in use. A delivery mode implies a specific SOAP message format, so a message that deviates from that format requires a new delivery mode.

R10.2.7-2: The NotifyTo address in the Subscribe message shall support only a single delivery mode.

This requirement is for the client because the service cannot verify whether this statement is true. If this requirement is not observed by the client, the service might not operate correctly. If the subscriber supports multiple delivery modes, the NotifyTo address needs to be differentiated in some way, such as by adding an additional reference parameter.

10.2.8 Event Action URI

Typically, each event type has its own wsa:Action URI to quickly identify and route the event. If an event type does not define its own wsa:Action URI, the following URI can be used as a default:

http://schemas.dmtf.org/wbem/wsman/1/wsman/Event

This URI can be used in cases where event types are inferred in real-time from other sources and not published as Web service events, and thus do not have a designated wsa:Action URI. This specification places no restrictions on the wsa:Action URI for events. More specific URIs can act as a reliable dispatching point. In many cases, a fixed schema can serve to model many different types of events, in which case the event "ID" is simply a field in the XML content of the event. The URI in this case might reflect the schema and be undifferentiated for all of the various event IDs that might occur or it might reflect the specific event by suffixing the event ID to the wsa:Action URI. This specification places no restrictions on the granularity of the URI, but careful consideration of these issues is part of designing the URIs for events.

10.2.9 Delivery Sequencing and Acknowledgement

The delivery mode indicates how the service will exchange events with interested parties. This clause describes delivery modes in detail.

10.2.9.1 General

For some event types, ordered and acknowledged delivery is important, but for other types of events the order of arrival is not significant. WS-Management defines four standard delivery modes:

- http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push

  With this mode, each SOAP message has only one event and no acknowledgement or SOAP response. The service can deliver events for the subscription asynchronously without regard to any events already in transit. This mode is useful when the order of events does not matter, such as with events containing running totals in which each new event can replace the previous one completely and the time stamp is sufficient for identifying the most recent event.

- http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck

  With this mode, each SOAP message has only one event, but each event is acknowledged before another is sent. The service queues all undelivered events for the subscription and delivers each new event only after the previous one has been acknowledged.

- http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

  With this mode, each SOAP message can have many events, but each batch is acknowledged before another is sent. The service queues all events for the subscription and delivers them in that order, maintaining the order in the batches.

- http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull

  With this mode, each SOAP message can have many events, but each batch is acknowledged. Because the receiver uses Pull to synchronously retrieve the events, acknowledgement is implicit. The order of delivery is maintained.

Ordering of events across subscriptions is not implied.

The acknowledgement model is discussed in 10.8.

10.2.9.2 Push Mode

The standard delivery mode is http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push, in which each delivery consists of a single event. No acknowledgement occurs, so the delivery cannot be faulted to cancel the subscription.

Therefore, subscriptions made with this delivery mode can have short durations to prevent a situation in which deliveries cannot be stopped if the SubscriptionManager content from the SubscribeResponse information is corrupted or lost.

To promote fast routing of events, the required wsa:Action URI in each event message can be distinct for each event type, regardless of how strongly typed the event body is.

R10.2.9.2-1: A service may support the http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push delivery mode.

R10.2.9.2-2: To precisely control how to deal with events that are too large, the service may accept the following additional instruction in a subscription:

```
(1)  <wsme:Delivery>
(2)    <wsme:NotifyTo> ... </wsme:NotifyTo>
(3)    ...
(4)    <wsman:MaxEnvelopeSize Policy="enumConstant">
```

```
(5)      xs:positiveInteger
(6)    </wsman:MaxEnvelopeSize>
(7)    ...
(8)  </wsme:Delivery>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsme:Delivery/wsman:MaxEnvelopeSize
    the maximum number of octets for the entire SOAP envelope in a single event delivery

wsme:Delivery/wsman:MaxEnvelopeSize/@Policy
    an optional value with one of the following enumeration values:

- **CancelSubscription:** cancel on the first oversized event
- **Skip:** silently skip oversized events
- **Notify:** notify the subscriber that events were dropped as specified in 10.9

R10.2.9.2-3: If wsman:MaxEnvelopeSize is requested, the service shall not send an event body larger than the specified limit. The default behavior is to notify the subscriber as specified in 10.9, unless otherwise instructed in the subscription, and to attempt to continue delivery. If the event exceeds any internal default maximums, the service should also attempt to notify as specified in 10.9 rather than terminate the subscription, unless otherwise specified in the subscription. If wsman:MaxEnvelopeSize is too large for the service, the service shall return a wsman:EncodingLimit fault with the following detail code:

    http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize

In the absence of any other Policy instructions, services are to deliver notifications of dropped events to subscribers, as specified in 10.9.

10.2.9.3 PushWithAck Mode

This delivery mode is identical to the standard "Push" mode except that each delivery is acknowledged. Each delivery still has one event, and the wsa:Action element indicates the event type. However, a SOAP-based acknowledgement occurs as described in 10.7.

The delivery mode URI is:

    http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck

In every other respect except the delivery mode URI, this mode is identical to Push mode as described in 10.2.9.2.

R10.2.9.3-1: A service should support the http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck delivery mode. If the delivery mode is not supported, the service should return a fault of wsme:DeliveryModeRequestedUnavailable.

10.2.9.4 Batched Delivery Mode

Batching events is an effective way to minimize event traffic from a high-volume event source without sacrificing event timeliness. WS-Management defines a custom event delivery mode that allows an event source to bundle multiple outgoing event messages into a single SOAP envelope. Delivery is always acknowledged, using the model defined in 10.7.

R10.2.9.4-1: A service may support the http://schemas.dmtf.org/wbem/wsman/1/wsman/Events delivery mode. If the delivery mode is not supported, the service should return a fault of wsme:DeliveryModeRequestedUnavailable.

For this delivery mode, the Delivery element has the following format:

```
(1)  <wsme:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
(2)    <wsme:NotifyTo>
(3)      wsa:EndpointReferenceType
(4)    </wsme:NotifyTo>
(5)    <wsman:MaxElements> xs:positiveInteger </wsman:MaxElements> ?
(6)    <wsman:MaxTime> xs:duration </wsman:MaxTime> ?
(7)    <wsman:MaxEnvelopeSize Policy="enumConstant">
(8)      xs:positiveInteger
(9)    </wsman:MaxEnvelopeSize> ?
(10) </wsme:Delivery>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsme:Delivery/@Mode

required attribute that shall be defined as

http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

wsme:Delivery/wsme:NotifyTo

required element that shall contain the EPR to which event messages are to be sent for this subscription

wsme:Delivery/wsman:MaxElements

optional element that contains a positive integer that indicates the maximum number of event bodies to batch into a single SOAP envelope

The resource shall not deliver more than this number of items in a single delivery, although it may deliver fewer.

wsme:Delivery/wsman:MaxEnvelopeSize

optional element that contains a positive integer that indicates the maximum number of octets in the SOAP envelope used to deliver the events

wsman:MaxEnvelopeSize/@Policy

an optional attribute with one of the following enumeration values:

- **CancelSubscription:** cancel on the first oversized event
- **Skip:** silently skip oversized events
- **Notify:** notify the subscriber that events were dropped as specified in 10.9

wsme:Delivery/wsman:MaxTime

optional element that contains a duration that indicates the maximum amount of time the service should allow to elapse while batching Event bodies

This time may not be exceeded between the encoding of the first event in the batch and the dispatching of the batch for delivery. Some publisher implementations may choose more complex schemes in which different events included in the subscription are delivered at different latencies or at different priorities. In such cases, a specific filter dialect can be designed for the purpose and used to describe the instructions to the publisher. In such cases, wsman:MaxTime can be omitted if it is not applicable; if present, however, it serves as an override of anything defined within the filter.

In the absence of any other instructions in any part of the subscription, services are to deliver notifications of dropped events to subscribers, as specified in 10.9.

If a client wants to discover the appropriate values for wsman:MaxElements or wsman:MaxEnvelopeSize, the client can query for service-specific metadata. The format of such metadata is beyond the scope of this particular specification.

R10.2.9.4-2: If batched mode is requested in a Subscribe message, and MaxElements, MaxEnvelopeSize, and MaxTime elements are not present, the service may pick any applicable defaults. The following faults apply:

- If MaxElements is not supported, wsman:UnsupportedFeature is returned with the following fault detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements

- If MaxEnvelopeSize is not supported, wsman:UnsupportedFeature is returned with the following fault detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize

- If MaxTime is not supported, wsman:UnsupportedFeature is returned with the following fault detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime

- If MaxEnvelopeSize/@Policy is not supported, wsman:UnsupportedFeature is returned with the following fault detail code:

  http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy

R10.2.9.4-3: If wsman:MaxEnvelopeSize is requested, the service shall not send an event body larger than the specified limit. The default behavior is to notify the subscriber as specified in 10.9, unless otherwise instructed in the subscription, and to attempt to continue delivery. If the event exceeds any internal default maximums, the service should also attempt notification as specified in 10.9 rather than terminate the subscription, unless otherwise specified in the subscription.

If a subscription has been created using batched mode, all event delivery messages shall have the following format:

```
<s:Envelope ...>
  <s:Header>
    ...
    <wsa:Action>
      http://schemas.dmtf.org/wbem/wsman/1/wsman/Events
    </wsa:Action>
    ...
  </s:Header>
  <s:Body>
    <wsman:Events>
      <wsman:Event Action="event action URI">
        ...event body...
      </wsman:Event> +
    </wsman:Events>
  </s:Body>
</s:Envelope>
```

The following definitions provide additional, normative constraints on the preceding outline:

s:Envelope/s:Header/wsa:Action

required element that shall be defined as

http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

s:Envelope/s:Body/wsman:Events/wsman:Event

required elements that shall contain the body of the corresponding event message, as if wsman:Event were the s:Body element

s:Envelope/s:Body/wsman:Events/wsman:Event/@Action

required attribute that shall contain the wsa:Action URI that would have been used for the contained event message

R10.2.9.4-4: If batched mode is requested, deliveries shall be acknowledged as described in 10.7.

Dropped events (as specified in 10.9) are encoded with any other events.

EXAMPLE: The following example shows batching parameters supplied to a Subscribe operation. The service is instructed to send no more than 10 items per batch, to wait no more than 20 seconds from the time the first event is encoded until the entire batch is dispatched, and to include no more than 8192 octets in the SOAP message.

```
(1)    ...
(2)  <wsme:Delivery
(3)    Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
(4)    <wsme:NotifyTo>
(5)      <wsa:Address>http://2.3.4.5/client</wsa:Address>
(6)    </wsme:NotifyTo>
(7)    <wsman:MaxElements>10</wsman:MaxElements>
(8)    <wsman:MaxTime>PT20S</wsman:MaxTime>
(9)    <wsman:MaxEnvelopeSize>8192</wsman:MaxEnvelopeSize>
(10) </wsme:Delivery>
```

EXAMPLE: Following is an example of batched delivery that conforms to this specification:

```
(1) <s:Envelope
(2)  xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing
(4)  xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
(5)  xmlns:wsme="http://schemas.xmlsoap.org/ws/2004/08/eventing">
(6)  <s:Header>
(7)    <wsa:To s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
(8)    <wsa:Action>
(9)      http://schemas.dmtf.org/wbem/wsman/1/wsman/Events
(10)     </wsa:Action>
(11)     ...
(12)   </s:Header>
(13)   <s:Body>
(14)     <wsman:Events>
(15)       <wsman:Event
(16)         Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(17)       <DiskChange
(18)         xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(19)         <Drive> C: </Drive>
```

```
(20)        <FreeSpace> 802012911 </FreeSpace>
(21)      </DiskChange>
(22)      </wsman:Event>
(23)      <wsman:Event
(24)        Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(25)        <DiskChange
(26)          xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(27)          <Drive> D: </Drive>
(28)          <FreeSpace> 1402012913 </FreeSpace>
(29)        </DiskChange>
(30)      </wsman:Event>
(31)    </wsman:Events>
(32)  </s:Body>
(33) </s:Envelope>
```

The Action URI in line 9 specifies that this is a batch that contains distinct events. The individual event bodies are at lines 15–22 and lines 23–30. The actual Action attribute for the individual events is an attribute of the wsman:Event wrapper.

10.2.9.5 Pull Delivery Mode

In some circumstances, polling for events is an effective way of controlling data flow and balancing timeliness against processing ability. Also, in some cases, network restrictions prevent "push" modes from being used; that is, the service cannot initiate a connection to the subscriber.

WS-Management defines a custom event delivery mode, "pull mode," which allows an event source to maintain a logical queue of event messages received by enumeration. This delivery mode borrows the Pull message to retrieve events from the logical queue. However, all of the other pub/sub operations defined in this clause can continue to be used. (For example, Unsubscribe, rather than Release, is used to cancel a subscription.)

For this delivery mode, the Delivery element has the following format:

```
<wsme:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull">
  ...
</wsme:Delivery>
```

wsme:Delivery/@Mode shall be

http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull

R10.2.9.5-1: A service may support the http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull delivery mode. If pull mode is requested but not supported, the service shall return a fault of wsme:DeliveryModeRequestedUnavailable.

wsman:MaxElements, wsman:MaxEnvelopeSize, and wsman:MaxTime do not apply in the Subscribe message when using this delivery mode because the Pull message contains all of the necessary functionality for controlling the batching and timing of the responses.

R10.2.9.5-2: If a subscription incorrectly specifies parameters that are not compatible with pull mode, the service should issue a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch

R10.2.9.5-3: If pull mode is requested in a Subscribe message and the event source accepts the subscription request, the SubscribeResponse element in the REPLY message shall contain an EnumerationContext element suitable for use in a subsequent Pull operation.

EXAMPLE:

```
(1)  <s:Body ...>
(2)    <wsme:SubscribeResponse ...>
(3)      <wsme:SubscriptionManager>
(4)        wsa:EndpointReferenceType
(5)      </wsme:SubscriptionManager>
(6)      <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires>
(7)      <wsmen:EnumerationContext>...</wsmen:EnumerationContext>
(8)      ...
(9)    </wsme:SubscribeResponse>
(10) </s:Body>
```

The subscriber extracts the EnumerationContext and uses it thereafter in Pull requests.

R10.2.9.5-4: If pull mode is active, Pull messages shall use the EPR of the subscription manager obtained from the SubscribeResponse message. The EPR reference parameters are of a service-specific addressing model, but may use the WS-Management default addressing model if it is suitable.

R10.2.9.5-5: If pull mode is active and a Pull request returns no events (because none have occurred since the last "pull"), the service should return a wsman:TimedOut fault. The EnumerationContext is still considered active, and the subscriber may continue to issue Pull requests with the most recent EnumerationContext for which event deliveries actually occurred.

R10.2.9.5-6: If pull mode is active and a Pull request returns events, the service may return an updated EnumerationContext as specified for Pull, and the subscriber is expected to use the update, if any, in the subsequent Pull, as specified for the Enumeration operations. Bookmarks, if active, may also be returned in the header and shall also be updated by the service.

In practice, the service might not actually change the EnumerationContext, but the client cannot depend on it remaining constant. It is updated conceptually, if not actually.

In pull mode, the Pull request controls the batching. If no defaults are specified, the batch size is 1 and the maximum envelope size and timeouts are service-defined.

R10.2.9.5-7: If pull mode is active, the service shall not return an EndOfSequence element in the event stream because no concept of a "last event" exists in this mode. Rather, the enumeration context should become invalid if the subscription expires or is canceled for any reason.

R10.2.9.5-8: If pull mode is used, the service shall accept the wsman:MaxEnvelopeSize used in the Pull as the limitation on the event size that can be delivered.

The batching properties used in batched mode do not apply to pull mode. The client controls the maximum event size using the normal mechanisms in Pull.

10.3 GetStatus

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

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
(5)      </wsa:Action>
(6)      …
(7)    </s:Header>
```

```
(8)    <s:Body …>
(9)      <wsme:GetStatus …>
(10)       …
(11)     </wsme:GetStatus>
(12)   </s:Body>
(13) </s:Envelope>
```

Components of the preceding outline are additionally constrained as for a request to renew a subscription. Other components of the preceding outline are not further constrained by this specification.

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

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse
(5)      </wsa:Action>
(6)      …
(7)    </s:Header>
(8)    <s:Body …>
(9)      <wsme:GetStatusResponse …>
(10)       <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
(11)       …
(12)     </wsme:GetStatusResponse>
(13)   </s:Body>
(14) </s:Envelope>
```

Components of the preceding outline are constrained as for a response to a renew request. Other components of the preceding outline are not further constrained by this specification.

The wsme:GetStatus message is optional for WS-Management.

R10.3-1: The wse:GetStatus message in a constrained environment is a candidate for exclusion. If this message is not supported, then a wsa:ActionNotSupported fault shall be returned in response to this request.

Heartbeat support may be implemented rather than the wsme:GetStatus message.

10.4 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:

```
(1)  <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe
(5)      </wsa:Action>
(6)      …
(7)    </s:Header>
(8)    <s:Body>
(9)      <wsme:Unsubscribe …>
(10)       …
(11)     </wsme:Unsubscribe>
(12)   </s:Body>
(13) </s:Envelope>
```

Components of the preceding outline are additionally constrained only as for a request to renew a subscription. 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 shall reply with a response of the following form:

```
(1) <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)   http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse
(5)      </wsa:Action>
(6)      <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
(7)      …
(8)    </s:Header>
(9)    <s:Body />
(10) </s:Envelope>
```

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

R10.4-1: If a service supports Subscribe, it shall implement the Unsubscribe message and ensure that event delivery will be terminated if the message is accepted as valid. Delivery of events may occur after responding to the Unsubscribe message as long as the event traffic stops at some point.

R10.4-2: A service may unilaterally cancel a subscription for any reason, including internal timeouts, reconfiguration, or unreliable connectivity.

Clients need to be prepared to receive any events already in transit even though they have issued an Unsubscribe message. Clients have the option to either fault any such deliveries or accept them.

The EPR to use for this message is received from the SubscribeResponse element in the SubscriptionManager element.

10.5 Renew

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

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

```
(1) <s:Envelope …>
(2)    <s:Header …>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew
(5)      </wsa:Action>
(6)      …
(7)    </s:Header>
(8)    <s:Body …>
(9)      <wsme:Renew …>
(10)       <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
(11)       …
(12)     </wsme:Renew>
(13)   </s:Body>
(14) </s:Envelope>
```

Components of the preceding outline are additionally constrained as for a request to create a subscription. Other components of the preceding outline are not further constrained by this specification.

If the subscription manager accepts a request to renew a subscription, it shall 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 …>
    <wsme:RenewResponse …>
      <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
      …
    </wsme:RenewResponse>
  </s:Body>
</s:Envelope>
```

Components of the preceding outline are constrained as for a response to a subscribe request with the following addition(s):

/s:Envelope/s:Body/*/wsme: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 shall fail, and the subscription manager may generate a wsme:UnableToRenew fault indicating that the renewal was not accepted.

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

Processing of the Renew message is required, but it is not required to succeed.

R10.5-1: Although a conformant service shall accept the Renew message as a valid action, the service may always fault the request with a wsme:UnableToRenew fault, forcing the client to subscribe from scratch.

Renew has no effect on deliveries in progress, bookmarks, heartbeats, or other ongoing activity. It simply extends the lifetime of the subscription.

The EPR to use for this message is received from the SubscribeResponse element in the SubscriptionManager element.

10.6 SubscriptionEnd

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. The message shall 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 …>
    <wsme:SubscriptionEnd …>
       <wsme:SubscriptionManager>
         endpoint-reference
```

```
(12)       </wsme:SubscriptionManager>
(13)       <wsme:Status>
(14)         [
(15)     http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure |
(16)     http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown |
(17)     http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCancelling
(18)         ]
(19)       </wsme:Status>
(20)       <wsme:Reason xml:lang="language identifier" >xs:string</wsme:Reason>
?
(21)       …
(22)     </wsme:SubscriptionEnd>
(23)     …
(24)   </s:Body>
(25) </s:Envelope>
```

The following describes additional, normative constraints on the preceding outline:

/s:Envelope/s:Body/*/wsme:SubscriptionManager

Endpoint reference of the subscription manager. It is recommended that event sinks ignore this element as its usage requires the ability to compare EPRs for equality when no such mechanism exists. Event sinks are advised to use reference parameters in the /wsme:Subscribe/wsme:EndTo EPR if they wish to correlate this message against their outstanding subscriptions.

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

This value shall be used if the event source terminated the subscription because of problems delivering notifications.

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

This value shall 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/wsme:SubscriptionEnd/wsme:Status = "http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCancelling"

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

/s:Envelope/s:Body/wsme:SubscriptionEnd/wsme: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 in 5.4 may be included in the message, according to the usage and semantics defined in 5.4.

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

This SubscriptionEnd message is optional for WS-Management. In effect, it is the "last event" for a subscription. Because its primary purpose is to warn a subscriber that a subscription has ended, it is not suitable for use with pull-mode delivery.

R10.6-1: A conformant service may implement the SubscriptionEnd message.

R10.6-2: A conformant service shall not implement the SubscriptionEnd message when event delivery is done using pull mode as defined in 10.2.9.4.

R10.6-3: If SubscriptionEnd is supported, the message shall contain any reference parameters specified by the subscriber in the EndTo address in the original subscription.

R10.6-4: This rule intentionally left blank.

If the service delivers events over the same connection as the Subscribe operation, the client typically knows that a subscription has been terminated because the connection itself closes or terminates.

When the delivery connection is distinct from the subscribe connection, a SubscriptionEnd message is highly recommended; otherwise, the client has no immediate way of knowing that a subscription is no longer active.

10.7 Acknowledgement of Delivery

To ensure that delivery is acknowledged at the application level, the original subscriber can request that the event sink physically acknowledge event deliveries, rather than relying entirely on transport-level guarantees.

In other words, the transport might have accepted delivery of the events but not forwarded them to the actual event sink process, and the service would move on to the next set of events. System failures might result in dropped events. Therefore, a mechanism is needed in which a message-level acknowledgement can occur. This allows acknowledgement to be pushed up to the application level, increasing the reliability of event deliveries.

The client selects acknowledged delivery by selecting a delivery mode in which each event has a response. In this specification, the two acknowledged delivery modes are

- http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck
- http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

R10.7-1: A conformant service may support the PushWithAck or Events delivery mode. However, if either of these delivery modes is requested, to maintain an ordered queue of events, the service shall wait for the acknowledgement from the client before delivering the next event or events that match the subscription.

R10.7-2: If an acknowledged delivery mode is selected for the subscription, the service shall include the following SOAP headers in each event delivery:

```
(1)  <s:Header>
(2)     <wsa:ReplyTo> where to send the acknowledgement </wsa:ReplyTo>
(3)     <wsman:AckRequested/>
(4)     ...
(5)  </s:Header>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsa:ReplyTo

address that shall always be present in the event delivery as a consequence of the presence of wsman:AckRequested

The client extracts this address and sends the acknowledgement to the specified EPR as required by Addressing.

wsman:AckRequested

no content; requires that the subscriber acknowledge all deliveries as described later in this clause

The client then replies to the delivery with an acknowledgement or a fault.

R10.7-3: A service may request receipt acknowledgement by using the wsman:AckRequested block and subsequently expect an http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack message. If this message is not received as a reply, the service may terminate the subscription.

The acknowledgement message format returned by the event sink (receiver) to the event source is identical for all delivery modes. As shown in the following outline, it contains a unique wsa:Action, and the wsa:RelatesTo field is set to the MessageID of the event delivery to which it applies:

```
(1)   <s:Envelope ...>
(2)     <s:Header>
(3)       ...
(4)       <wsa:To> endpoint reference from the event ReplyTo field </wsa:To>
(5)       <wsa:Action> http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack
          </wsa:Action>
(6)       <wsa:RelatesTo> message ID of original event delivery
    </wsa:RelatesTo>
(7)       ...
(8)     </s:Header>
(9)     <s:Body/>
(10)</s:Envelope>
```

The following definitions provide additional, normative constraints on the preceding outline:

s:Envelope/s:Header/wsa:Action

URI that shall be defined as

http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack

s:Envelope/s:Header/wsa:RelatesTo

element that shall contain the wsa:MessageID of the event delivery to which it refers

wsa:RelatesTo is the critical item that ensures that the correct delivery is being acknowledged, and thus it shall not be omitted.

s:Envelope/s:Header/wsa:To

EPR address extracted from the ReplyTo field in the event delivery

All reference parameters shall be extracted and added to the SOAP header as well.

In spite of the request to acknowledge, the event sink can refuse delivery with a fault or fail to respond with the acknowledgement. In this case, the event source can terminate the subscription and send any applicable SubscriptionEnd messages.

If the event sink does not support acknowledgement, it can respond with a wsman:UnsupportedFeature fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack

However, this action is just as difficult as acknowledging the delivery, so most clients can scan for the wsman:AckRequested field and be prepared to acknowledge delivery or fault it.

10.8 Refusal of Delivery

With all acknowledged delivery modes as described in 10.7, an event sink can refuse to take delivery of events, either for security reasons or a policy change. It then responds with a fault rather than an acknowledgement.

In this case, the event source needs to be prepared to end the subscription even though an Unsubscribe message is not issued by the subscriber.

R10.8-1: During event delivery, if the receiver faults the delivery with a wsman:DeliveryRefused fault, the service shall immediately cancel the subscription and may also issue a SubscriptionEnd message to the EndTo endpoint in the original subscription, if supported.

Thus, the receiver can issue the fault as a way to cancel the subscription when it does not have the SubscriptionManager information.

10.9 Dropped Events

Events that cannot be delivered are not to be silently dropped from the event stream, or the subscriber gets a false picture of the event history. WS-Management defines three behaviors for events that cannot be delivered with push modes or that are too large to fit within the delivery constraints requested by the subscriber:

- Terminate the subscription.
- Silently skip such events.
- Send a special event in place of the dropped events.

These options are discussed in 10.2.9.2 and 10.2.9.3.

During delivery, the service might have to drop events for the following reasons:

- The events exceed the maximum size requested by the subscriber.
- The client cannot keep up with the event flow, and there is a backlog.
- The service might have been reconfigured or restarted and the events permanently lost.

In these cases, a service can inform the client that events have been dropped.

R10.9-1: If a service drops events, it should issue an http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents event, which indicates this drop to the client. Any reference parameters specified in the NotifyTo address in the subscription shall also be copied into this message. This event is normal and implicitly considered part of any subscription.

R10.9-2: If an http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents event is issued, it shall take the ordinal position of the original dropped event in the delivery stream. The DroppedEvents event is considered the same as any other event with regard to its location and other behavior (bookmarks, acknowledged delivery, location in batch, and so on). It simply takes the place of the event that was dropped.

EXAMPLE:

```
<s:Envelope ...>
  <s:Header>
    ...subscriber endpoint-reference...

    <wsa:Action>
      http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents
    </wsa:Action>
  </s:Header>
  <s:Body>
    <wsman:DroppedEvents Action="wsa:Action URI of dropped event">
      xs:int
    </wsman:DroppedEvents>
    ...
  </s:Body>
```

```
(15) </s:Envelope>
```

The following definitions provide additional, normative constraints on the preceding outline:

s:Envelope/s:Header/wsa:Action

URI that shall be defined as

http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents

s:Body/wsman:DroppedEvents/@Action

the Action URI of the event that was dropped

s:Body/wsman:DroppedEvents

a positive integer that represents the total number of dropped events since the subscription was created

Renew has no effect on the running total of dropped events. Dropped events are like any other events and can require acknowledgement, affect the bookmark location, and so on.

EXAMPLE: Following is an example of how a dropped event would appear in the middle of a batched event delivery:

```
(1)  <wsman:Events>
(2)    <wsman:Event Action="https://foo.com/someEvent">
(3)      …event body
(4)    </wsman:Event>
(5)    <wsman:Event
(6)      Action="http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents">
(7)    <wsman:DroppedEvents Action="https://foo.com/someEvent">
(8)      1
(9)    </wsman:DroppedEvents>
(10)   </wsman:Event>
(11)   <wsman:Event Action="https://foo.com/someEvent">
(12)     ...event body...
(13)   </wsman:Event>
(14) <wsman:Events>
```

R10.9-3: If a service cannot deliver an event and does not support the http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents event, it should terminate the subscription rather than silently skipping events.

Because this requirement cannot be enforced, and some dropped events are irrelevant when replaced by a subsequent event (running totals, for example), it is not a firm requirement that dropped events are signaled or that they result in a termination of the subscription.

10.10 Access Control

It is important for event sources to properly authorize requests. This is especially true for Subscribe requests, because 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. Be aware that event sources that are not reachable from the Internet have less need to control Subscribe requests.

10.11 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.

10.12 Advertisement of Notifications

An Event Source can choose to advertise the Notification messages that it might send by including a well-defined portType, called "EventSink", in its WSDL. Subscribers can examine this portType to determine which messages they might need to support. Each Notification appears as an independent operation within the portType, as shown in the following example:

EXAMPLE:

```
<wsdl:portType name="EventSink">
  <wsdl:operation name="WeatherReport">
    <wsdl:input message="wr:ThunderStormMessage"
     wsa:Action="urn:weatherReport:ThunderStorm"
     wsam:Action="urn:weatherReport:ThunderStorm" />
   <wsdl:input message="wr:TyphoonMessage"
     wsa:Action="urn:weatherReport:Typhoon"
     wsam:Action="urn:weatherReport:Typhoon" />
  </wsdl:operation>
</wsdl:portType>
```

In the preceding example this Event Source can send two types of Notifications (a ThunderStorm and a Typhoon message).

Unless otherwise noted, Event Sinks should assume that the Notifications will be sent using SOAP1.2 and will use document-literal encoding.

11 Metadata and Discovery

The WS-Management protocol is compatible with many techniques for discovery of resources available through a service.

In addition, this specification defines a simple request-response operation to facilitate the process of establishing communications with a WS-Management service implementation in a variety of network environments without prior knowledge of the protocol version or versions supported by the implementation. This operation is used to discover the presence of a service that is compatible with WS-Management, assuming that a transport address over which the message can be delivered is known. Typically, a simple HTTP address would be used.

To ensure forward compatibility, the message content of this operation is defined in an XML namespace that is separate from the core protocol namespace and that will not change as the protocol evolves. Further, this operation does not depend on any SOAP envelope header or body content other than the types explicitly defined for this operation. In this way, WS-Management clients are assured of the ability to use this operation against all implementations and versions to confirm the presence of WS-Management services without knowing the supported protocol versions or features in advance.

The request message is defined as follows:

```
<s:Envelope
  xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/identity/1/
    wsmanidentity.xsd"
  <s:Header>
    ...
  </s:Header>
  <s:Body>
    <wsmid:Identify>
      ...
    </wsmid:Identify>
  </s:Body>
</s:Envelope>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsmid:Identify

the body of the Identify request operation, which may contain additional vendor-specific extension content, but is otherwise empty

The presence of this body element constitutes the request.

Notice the absence of any Addressing namespace, WS-Management namespace, or other version-specific concepts. This message is compatible only with the basic SOAP specification, and the presence of the wsmid:Identify block in the s:Body is the embodiment of the request operation.

The response message is defined as follows:

```
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/identity/1/
        wsmanidentity.xsd">
    <s:Header>
      ...
    </s:Header>
    <s:Body>
      <wsmid:IdentifyResponse>
        <wsmid:ProtocolVersion> xs:anyURI </wsmid:ProtocolVersion> +
        <wsmid:ProductVendor> xs:string </wsmid:ProductVendor> ?
        <wsmid:ProductVersion> xs:string </wsmid:ProductVersion> ?
        <wsmid:InitiativeSupport>
          <wsmid:InitiativeName> xs:string </wsmid:InitiativeName> ?
          <wsmid:InitiativeVersion> xs:string </wsmid:InitiativeVersion> ?
        </wsmid:InitiativeSupport> ?
        <wsmid:SecurityProfiles>
          <wsmid:SecurityProfileName> xs:anyURI
</wsmid:SecurityProfileName> *
        </wsmid:SecurityProfiles> ?
        <wsmid:AddressingVersionURI> xs:anyURI
</wsmid:AddressingVersionURI> *
        ...
    </wsmid:IdentifyResponse>
  </s:Body>
</s:Envelope>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsmid:IdentifyResponse

the body of the response, which packages metadata about the WS-Management implementation

wsmid:IdentifyResponse/wsmid:ProtocolVersion

a required element or elements, each of which is a URI whose value shall be equal to the core XML namespace that identifies a supported version of the WS-Management specification

One element shall be provided for each supported version of the protocol. Services should also include the XML namespace URI for supported dependent specifications such as Addressing. For example, if a future version of WS-Management supports multiple versions of Addressing, the IdentifyResponse can indicate which of the versions are supported.

wsmid:IdentifyResponse/wsmid:ProductVendor

an optional element that identifies the vendor of the WS-Management service implementation by using a widely recognized name or token, such as the official corporate name of the vendor or its stock symbol

Alternatively, a DNS name, e-mail address, or Web URL may be used.

wsmid:IdentifyResponse/wsmid:ProductVersion

an optional version string for the WS-Management implementation

This specification places no constraints on the format or content of this element.

wsmid:IdentifyResponse/wsmid:InitiativeSupport

an optional element that identifies an initiative supported by the WS-Management implementation.

wsmid:IdentifyResponse/wsmid:InitiativeSupport/wsmid:InitiativeName

an element that identifies the name of an initiative supported by the WS-Management implementation.

wsmid:IdentifyResponse/wsmid:InitiativeSupport/wsmid:InitiativeVersion

an element that identifies the version of an initiative supported by the WS-Management implementation.

In addition, vendor-specific content can follow the preceding standardized elements. After the vendor-specific content, the following elements can follow:

wsmid:IdentifyResponse/wsmid:SecurityProfiles

an optional element that identifies the set of security profiles supported by the WS-Management implementation.

wsmid:IdentifyResponse/wsmid:SecurityProfiles/wsmid:SecurityProfileName

an optional element which is a URI that identifies a security profile supported by the WS-Management implementation.

wsmid:IdentifyResponse/wsmid:AddressingVersionURI

an optional element which is a URI that identifies a version of Addressing supported by the WS-Management implementation.

When a service supports this element, the value shall be the XML Schema namespace URI of the addressing version in use. XML Schema namespaces used in this specification are listed in ANNEX A. A service may support and advertise more than none version of addressing.

R11-1: A WS-Management service should support the wsmid:Identify operation. A service implementation that supports the operation shall do so irrespective of the versions of WS-Management supported by that service. The operation shall be accessible at the same transport-level address at which the resource instances are made accessible.

It is recommended that client applications not include any SOAP header content in the wsmid:Identify operation delivered to the transport address against which the inquiry is being made. If SOAP header elements are present, the s:mustUnderstand attribute on all such elements can be set to "false". Doing otherwise reduces the likelihood of a successful, version-independent response from the service.

R11-2: A service that supports the wsmid:Identify operation shall not require the presence of any SOAP header elements in order to dispatch execution of the request. If a service receives a wsmid:Identify operation that contains unexpected or unsupported header content with the s:mustUnderstand attribute set to "false", the service shall not fault the request and shall process the body of the request as though the header elements were not present.

R11-3: A service that is processing the wsmid:Identify request should not request the presence of any Addressing header values, including the wsa:Action URI.

The entire purpose of this mechanism is to be able to identify the presence of specific versions of WS-Management (and the corresponding dependent protocols) in a version-independent manner.

Because Addressing is not used, the address to which this message is delivered is defined entirely at the transport level and not present in the SOAP content.

If a client does not have any prior knowledge about a service including credentials, it is desirable to allow a service to process an Identify message without requiring authentication.

R11-4: A service that supports the wsmid:Identify operation may expose this operation without requiring client or server authentication in order to process the message. In the absence of other requirements, it is recommended that the network address be suffixed by the token sequence */wsman-anon/identify*.

Services that support unauthenticated wsmid:Identify requests might choose not to reveal descriptive information about protocol, vendor, or other versioning information that could potentially represent or contribute to a vulnerability. To accommodate this scenario, this specification defines a URI that services can use in place of a valid WS-Management protocol version URI. This value can be returned as a value for the wsmid:ProtocolVersion element of the wsmid:IdentifyResponse message.

R11-5: A service supporting an unauthenticated wsmid:Identify message may respond using the following URI for the value of the wsmid:ProtocolVersion element:

http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity/NoAnonymousDisclosure

R11-6: A service that provides unauthenticated access to the wsmid:Identify operation but does not respond to such requests with the WS-Management protocol versions that are supported by the service shall support authenticated access to the wsmid:Identify operation. Such services shall respond to authenticated requests with the WS-Management protocol version identifiers for each version of the WS-Management protocol supported by the service.

12 Security

12.1 General

In general, management operations and responses need to be protected against attacks such as snooping, interception, replay, and modification during transmission. Authenticating the user who has sent a request is also generally necessary so that access control rules can be applied to determine whether to process a request.

This specification establishes the minimum interoperation standards and predefined profiles using transport-level security.

This approach provides the best balance between simple implementations (HTTP and HTTPS stacks are readily available, even for hardware) and the security mechanisms that sit in front of any SOAP message processing, limiting the attack surface.

More sophisticated transport and SOAP-level profiles, published separately from this specification, may be defined and used.

Implementations that expect to interoperate can adopt one or more of the transport and security models defined in this clause and are free to define any additional profiles under different URI-based designators.

12.2 Security Profiles

For this specification, a profile is any arbitrary mix of transport or SOAP behavior that describes a common security need. In some cases, the profile is defined for documentation and metadata purposes, but might not be part of the actual message exchange. Rather, it *describes* the message exchange involved.

Metadata retrieval can be employed to discover which profiles the service supports, and that is beyond the scope of this particular specification.

For all predefined profiles, the transport is responsible for all message integrity, protection, authentication, and security.

This specification makes no assumptions about the security requirements of the applications that use WS-Eventing. However, once those requirements have been satisfied within a given operational context, the addition of WS-Eventing to this operational context cannot undermine the fulfillment of those requirements; the use of WS-Eventing SHOULD NOT create additional attack vectors within an otherwise secure system.

The authentication profiles do not appear in the SOAP traffic, with the exception of the Subscribe message when using any delivery mode that causes a new connection to be created from the event source to the event sink (push and batched modes, for example). When a subscription is created, the authentication technique for event-delivery needs to be specified by the subscriber, because the event sink has to authenticate the event source (acting as publisher) at event delivery-time.

In this specification, security profiles are identified by a URI. As profiles are defined, they can be assigned a URI and published. WS-Management defines a set of standardized security profiles for the common transports HTTP and HTTPS as described in C.3.1.

12.3 Security Considerations for Event Subscriptions

When specifying the NotifyTo address in subscriptions, it is often important to hint to the service about which authentication model to use when delivering the event.

If no hints are present, the service can simply infer from the wsa:To address what needs to be done. However, if the service can support multiple modes and has a certificate or password store, it might not know which authentication model to choose or which credentials to use without being told in the subscription.

WS-Management provides a default mechanism to communicate the desired authentication mode and credentials. However, more sophisticated mechanisms are beyond the scope of this version of WS-Management. For example, the event sink service could export metadata that describes the available options, allowing the publisher to negotiate an appropriate option. Extension profiles can define other mechanisms enabled through a SOAP header with mustUnderstand="true".

WS-Management defines an additional field in the Delivery block that can communicate authentication information, as shown in the following outline:

```
<s:Body>
  <wsme:Subscribe>
    <wsme:Delivery>
      <wsme:NotifyTo>  Delivery EPR </wsme:NotifyTo>
      <wsman:Auth Profile="authentication-profile-URI"/>
    </wsme:Delivery>
  </wsme:Subscribe>
</s:Body>
```

The following definitions provide additional, normative constraints on the preceding outline:

wsman:Auth

block that contains authentication information to be used by the service (acting as publisher) when authenticating to the event sink at event delivery time

wsman:Auth/@Profile

a URI that indicates which security profile to use when making the connection to deliver events

If the wsman:Auth block is not present, by default the service infers what to do by using the NotifyTo address and any preconfigured policy or settings it has available. If the wsman:Auth block is present and no security-related tokens are communicated, the service needs to know which credentials to use by its own internal configuration.

If the service is already configured to use a specific certificate when delivering events, the subscriber can request standard mutual authentication, as shown in the following outline:

```
<s:Body>
  <wsme:Subscribe>
    <wsme:Delivery>
    <wsme:NotifyTo> HTTPS address </wsme:NotifyTo>
      <wsman:Auth
      Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/
        mutual"/>
    </wsme:Delivery>
  </wsme:Subscribe>
</s:Body>
```

If the service knows how to retrieve a proper user name and password for event delivery, simple HTTP Basic or Digest authentication can be used, as shown in the following outline:

```
<s:Body>
  <wsme:Subscribe>
    <wsme:Delivery>
      <wsme:NotifyTo> HTTP address </wsme:NotifyTo>
      <wsman:Auth
        Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/
          digest"/>
    </wsme:Delivery>
  </wsme:Subscribe>
</s:Body>
```

Services are not required to support any specific profile. The rest of this clause defines special-case profiles for event delivery in which the service needs additional information to select the proper credentials to use when delivering events.

12.4 Including Credentials with a Subscription

This clause intentionally left blank.

12.5 Correlating Events with a Subscription

In many cases, the subscriber will want to ensure that the event delivery corresponds to a valid subscription issued by an authorized party. In this case, it is recommended that reference parameters be introduced into the NotifyTo definition.

EXAMPLE: At subscription time, a UUID could be supplied as a correlation token:

```
(1)  <s:Body>
(2)    <wsme:Subscribe>
(3)      <wsme:Delivery>
(4)        <wsme:NotifyTo>
(5)          <wsa:Address> address <wsa:Address>
(6)          <wsa:ReferenceParameters>
(7)            <MyNamespace:uuid>
(8)              uuid:b0f685ec-e5c9-41b5-b91c-7f580419093e
(9)            </MyNamespace:uuid>
(10)           </wsa:ReferenceParameters>
(11)       </wsme:NotifyTo>
(12)       ...
(13)     </wsme:Delivery>
(14)     ...
(15)   </wsme:Subscribe>
(16) </s:Body>
```

This definition requires that the service include the MyNamespace:uuid value as a SOAP header with each event delivery (see 5.1). The service can use this value to correlate the event with any subscription that it issued and to validate its origin.

This is not a transport-level or SOAP-level authentication mechanism as such, but it does help to maintain and synchronize valid lists of subscriptions and to determine whether the event delivery is authorized, even though the connection itself could have been authenticated.

This mechanism still can require the presence of the wsman:Auth block to specify which security mechanism to use to actually authenticate the connection at event-time.

Each new subscription can receive at least one unique reference parameter that is never reused, such as the illustrated UUID, for this mechanism to be of value.

Other reference parameters can be present to help route and correlate the event delivery as required by the subscriber.

12.6 Transport-Level Authentication Failure

Because transports typically go through their own authentication mechanisms prior to any SOAP traffic occurring, the first attempt to connect might result in a transport-level authentication failure. In such cases, SOAP faults will not occur, and the means of communicating the denial to the client is implementation- and transport-specific.

12.7 Security Implications of Third-Party Subscriptions

Without proper authentication and authorization, WS-Management implementations can be vulnerable to distributed denial-of-service attacks through third-party subscriptions to events. This vulnerability is discussed in 10.10.

13 Transports and Message Encoding

This clause describes encoding rules that apply to all transports.

13.1 SOAP

WS-Management qualifies the use of SOAP as indicated in this clause.

R13.1-1: A service shall at least receive and send SOAP 1.2 SOAP Envelopes.

R13.1-2: A service may reject a SOAP Envelope with more than 32,767 octets.

R13.1-3: A service should not send a SOAP Envelope with more than 32,767 octets unless the client has specified a wsman:MaxEnvelopeSize header that overrides this limit.

Large SOAP Envelopes are expected to be serialized using attachments.

R13.1-4: Any Request Message may be encoded using either Unicode 3.0 (UTF-16) or UTF-8 encoding. A service shall accept the UTF-8 encoding type for all operations and should accept UTF-16 as well.

R13.1-5: A service shall emit Responses using the same encoding as the original request. If the service does not support the requested encoding or cannot determine the encoding, it should use UTF-8 encoding to return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/CharacterSet

R13.1-6: For UTF-8 encodings, the service may fail to process any message that begins with the UTF-8 BOM (0xEF 0xBB 0xBF), and shall send UTF-8 responses without the BOM.

The presence of BOM in 8-bit character encodings reduces interoperation. Where extended characters are a requirement, UTF-16 can be used.

R13.1-7: If UTF-16 is the encoding, the service shall support either byte-order mark (BOM) U+FEFF (big-endian) or U+FFFE (little-endian) as defined in the Unicode 3.0 specification as the first character in the message (see the Unicode BOM FAQ).

R13.1-8: If a request includes contradictory encoding information in the BOM and HTTP charset header or if the information does not fully specify the encoding, the service shall fault with an HTTP status of "bad request message" (400).

Repeated headers with the same QName but different values that imply contradictory behavior are considered a defect originating on the client side of the conversation. Returning a fault helps identify faulty clients. However, an implementation might be resource-constrained and unable to detect duplicate headers, so the repeated headers can be ignored. Repeated headers with the same QName that contains informational or non-contradictory instructions are possible, but none are defined by this specification or its dependencies.

R13.1-9: If a request contains multiple SOAP headers with the same QName from WS-Management, Addressing, or clause 10 of this specification, the service should not process them and should issue a wsa:InvalidMessageInformationHeaders fault if they are detected. (No SOAP headers are defined in clause 7 "Resource Access" or clause 8 "Enumeration of Datasets".)

R13.1-10: By default, a compliant service should not fault requests with leading and trailing whitespace in XML element values and should trim such whitespace by default as if the whitespace had not occurred. Services should not emit messages containing leading or trailing whitespace within element values unless the whitespace values are properly part of the value. If the service cannot accept whitespace usage within a message because the XML schema establishes other whitespace usage, the service should emit a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Whitespace

Clients can send messages with leading or trailing whitespace in the values, and services are permitted to eliminate unneeded "cosmetic" whitespace on both sides of the element value without faulting. (See XML Schema Part 2: Datatypes.)

R13.1-11: Services should not fault messages that contain XML comments, because this is part of the XML standard. Services may emit messages that contain comments that relate to the origin and processing of the message or add comments for debugging purposes.

13.2 Lack of Response

If an operation succeeds but a response cannot be computed or actually delivered because of run-time difficulties or transport problems, no response is sent and the connection is terminated.

This behavior is preferable to attempting a complex model for sending responses in a delayed fashion. Implementations can generally keep a log of all requests and their results, and allow the client to reconnect later to enumerate the operation log (using Enumerate) if it failed to get a response. The format and behavior of such a log is beyond the scope of this specification. In any case, the client needs to be coded to take into account a lack of response; all abnormal message conditions can safely revert to this scenario.

R13.2-1: If correct responses or faults cannot be computed or generated due to internal service failure, a response should not be sent.

Regardless, the client has to deal with cases of no response, so the service can simply force the client into that mode rather than send a response or fault that is not defined in this specification.

13.3 Replay of Messages

This section intentionally left blank.

R13.3-1: This rule intentionally left blank.

13.4 Encoding Limits

Most of the following limits are in characters. However, the maximum overall SOAP envelope size is defined in octets. Implementations are free to exceed these limits. A service is considered conformant if it observes these limits. Any limit violation results in a wsman:EncodingLimit fault.

R13.4-1: A service may fail to process any URI with more than 2048 characters and should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/URILimitExceeded

R13.4-2: A service should not generate a URI with more than 2048 characters.

R13.4-3: A service may fail to process an Option Name of more than 2048 characters.

R13.4-4: A service may fail to process an Option value of more than 4096 characters.

R13.4-5: A service may fault any operation that would require a single reply exceeding 32,767 octets.

R13.4-6: A service may always emit faults that are 4096 octets or less in length, regardless of any requests by the client to limit the response size. Clients need to be prepared for this minimum in case of an error.

R13.4-7: When the default addressing model is in use, a service may fail to process a Selector Name of more than 2048 characters.

R13.4-8: A service may have a maximum number of selectors that it can process. If the request contains more selectors than this limit, the service should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/SelectorLimit

R13.4-9: A service may have a maximum number of options that it can process. If the request contains more options than this limit, the service should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionLimit

13.5 Binary Attachments

SOAP Message Transmission Optimization Mechanism (MTOM) is used to support binary attachments to WS-Management. If a service supports attachments, the following rules apply:

R13.5-1: A conformant service may optionally support binary attachments to any operation using the SOAP MTOM proposal.

R13.5-2: If a service supports attachments, the service shall support the Abstract Transmission Optimization Feature.

R13.5-3: If a service supports attachments, the service shall support the Optimized MIME Multipart Serialization Feature.

Other attachment types are not prohibited. Specific transports can impose additional encoding rules.

13.6 Case-Sensitivity

While XML and SOAP are intrinsically case-sensitive with regard to schematic elements, WS-Management can be used with many underlying systems that are not intrinsically case-sensitive. This support primarily applies to values, but can also apply to schemas that are automatically and dynamically generated from other sources.

A service can observe any case usage required by the underlying execution environment.

The only requirement is that messages are able to pass validation tests against any schema definitions. At any time, a validation engine could be interposed between the client and server in the form of a proxy, so schematically valid messages are a practical requirement.

Otherwise, this specification makes no requirements as to case usage. A service is free to interpret values in a case-sensitive or case-insensitive manner.

It is recommended that case usage not be altered in transit by any part of the WS-Management processing chain. The case usage established by the sender of the message is to be retained throughout the lifetime of that message.

14 Faults

Many of the operations outlined in WS-Management can generate faults. This clause describes how these faults should be formatted into SOAP messages.

14.1 Introduction

Faults are returned when the SOAP message is successfully delivered by the transport and processed by the service, but the message cannot be processed properly. If the transport cannot successfully deliver the message to the SOAP processor, a transport error occurs.

R14.1-1: A service should support only SOAP 1.2 (or later) faults.

Generally, faults are not to be issued unless they are expected as part of a request-response pattern. For example, it would not be valid for a client to issue a Get message, receive the GetResponse message, and then *fault* that response.

14.2 Fault Encoding

This clause discusses XML fault encoding.

R14.2-1: A conformant service shall use the following fault encoding format and normative constraints for faults in the WS-Management space or any of its dependent specifications:

```
<s:Envelope>
  xmlns:s="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <s:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
  <wsa:Action>
  <wsa:MessageID>
    uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
    </wsa:MessageID>
    <wsa:RelatesTo>
      uuid:d9726315-bc91-430b-9ed8-ce5ffb858a85
    </wsa:RelatesTo>
  </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 following definitions provide additional, normative constraints on the preceding outline:

s:Envelope/s:Header/wsa:Action
: a valid fault Action URI from the relevant specification that defined the fault

s:Envelope/s:Header/wsa:MessageId
: element that shall be present for the fault, like any non-fault message

s:Envelope/s:Header/wsa:RelatesTo
: element that shall, like any other reply, contain the MessageID of the original request that caused the fault

s:Body/s:Fault/s:Value
: element that shall be either s:Sender or s:Receiver, as specified in 14.6 in the "Code" field

s:Body/s:Fault/s:Subcode/s:Value
: for WS-Management-related messages, shall be one of the subcode QNames defined in 14.6
: If the service exposes custom methods or other messaging, this value may be another QName not in the Master Faults described in 14.6.

s:Body/s:Fault/s:Reason
: optional element that should contain localized text that explains the fault in more detail
: Typically, this text is extracted from the "Reason" field in the Master Fault tables (14.6). However, the text may be adjusted to reflect a specific circumstance. This element may be repeated for multiple languages. The xml:lang attribute shall be present on the s:Text element.

s:Body/s:Fault/s:Detail
: optional element that should reflect the recommended content from the Master Fault tables (14.6)

The preceding fault template is populated by examining entries from the Master Fault tables in 14.6, which includes all relevant faults from WS-Management and its underlying specifications.

s:Reason and s:Detail are always optional, but they are recommended. In addition, s:Reason/s:Text contains an xml:lang attribute to indicate the language used in the descriptive text.

R14.2-2: Fault wsa:Action URI values vary from fault to fault. The service shall issue a fault using the correct URI, based on the specification that defined the fault. Faults defined in this specification shall have the following URI value:

http://schemas.dmtf.org/wbem/wsman/1/wsman/fault

The Master Fault tables in 14.6 contain the relevant wsa:Action URIs. The URI values are directly implied by the QName for the fault.

14.3 NotUnderstood Faults

There is a special case for faults relating to mustUnderstand attributes on SOAP headers. SOAP specifications define the fault differently than the encoding in 14.2 (see 5.4.8 in SOAP 1.2). In practice, the fault varies only in indicating the SOAP header that was not understood, the QName, and the namespace (see line 5 in the following outline).

```
(1)  <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(2)    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(3)
(4)    <s:Header>
(5)      <s:NotUnderstood qname="QName of header" xmlns:ns="XML namespace of
           header"/>
```

```
(6)      <wsa:Action>
(7)        http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
(8)      </wsa:Action>
(9)      <wsa:MessageID>
(10)       urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(11)     </wsa:MessageID>
(12)     <wsa:RelatesTo>
(13)       urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a85
(14)     </wsa:RelatesTo>
(15)   </s:Header>
(16)
(17)   <s:Body>
(18)     <s:Fault>
(19)       <s:Code>
(20)         <s:Value>s:MustUnderstand</s:Value>
(21)       </s:Code>
(22)       <s:Reason>
(23)         <s:Text xml:lang="en-US">Header not understood</s:Text>
(24)       </s:Reason>
(25)     </s:Fault>
(26)   </s:Body>
(27)
(28) </s:Envelope>
```

The preceding fault template can be used in all cases of failure to process mustUnderstand attributes. Lines 5–8 show the important content, indicating which header was not understood and including a generic wsa:Action that specifies that the current message is a fault.

The wsa:RelatesTo element is included so that the client can correlate the fault with the original request. Over transports other than HTTP in which requests might be interlaced, this might be the only way to respond to the correct sender.

If the original wsa:MessageID itself is faulty and the connection is request-response oriented, the service can attempt to send back a fault without the wsa:RelatesTo field, or can simply fail to respond, as discussed in 14.4.

14.4 Degenerate Faults

In rare cases, the SOAP message might not contain enough information to properly generate a fault. For example, if the wsa:MessageID is garbled, the service will have difficulty returning a fault that references the original message. Some transports might not be able to reference the sender to return the fault.

If the transport guarantees a simple request-response pattern, the service can send back a fault with no wsa:RelatesTo field. However, in some cases, there is no guarantee that the sender can be reached (for example, if the wsa:FaultTo contains an invalid address, so there is no way to deliver the fault).

In all cases, the service can revert to the rules of 13.3, in which no response is sent. The service can attempt to log the requests in some way to help identify the defective client.

14.5 Fault Extensibility

A service can include additional fault information beyond what is defined in this specification. The appropriate extension element is the s:Detail element, and the service-specific XML can appear at any location within this element, provided that it is properly mapped to an XML namespace that defines the schema for that content. WS-Management makes use of this extension technique for the wsman:FaultDetail URI values, as shown in the following outline:

```
(1)  <s:Detail>
(2)    <wsman:FaultDetail>... </wsman:FaultDetail>
(3)    <ExtensionData xmlns="vendor-specific-namespace">...</ExtensionData>
(4)    ...
(5)  </s:Detail>
```

The extension data elements can appear before or after any WS-Management-specific extensions mandated by this specification. More than one extension element is permitted.

14.6 Master Faults

This clause includes all faults from this specification and all underlying specifications. This list is the normative fault list for WS-Management.

> **R14.6-1:** A service shall return faults from the following list when the operation that caused them was a message in this specification for which faults are specified. A conformant service may return other faults for messages that are not part of WS-Management.

It is critical to client interoperation that the same fault be used in identical error cases. If each service returns a distinct fault for "Not Found", for example, constructing interoperable clients would be impossible. In Table 5 through Table 43, the source specification of a fault is based on its QName.

Table 5 – wsman:AccessDenied

Fault Subcode	wsman:AccessDenied
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The sender was not authorized to access the resource.
Detail	None
Comments	This fault is returned generically for all access denials that relate to authentication or authorization failures. This fault does not indicate locking or concurrency conflicts or other types of denials unrelated to security by itself.
Applicability	Any message
Remedy	The client acquires the correct credentials and retries the operation.

5971 **Table 6 – wsa:ActionNotSupported**

Fault Subcode	wsa:ActionNotSupported
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	The action is not supported by the service.
Detail	<s:Detail> <wsa:Action> *Incorrect Action URI* </wsa:Action> </s:Detail> <!-- The unsupported Action URI is returned, if possible -->
Comments	This fault means that the requested action is not supported by the implementation. As an example, read-only implementations (supporting only Get and Enumerate) return this fault for any operations other than these two. If the implementation never supports the action, the fault can be generated as shown in the "Detail" row of this table. However, if the implementation supports the action in a general sense, but it is not an appropriate match for the resource, an additional detail code can be added to the fault, as follows: <s:Detail> <wsa:Action> *The offending Action URI* </wsa:Action> <wsman:FaultDetail> http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ActionMismatch </wsman:FaultDetail> </s:Detail> This situation can occur when the implementation supports Put, for example, but the client attempts to update a read-only resource.
Applicability	All messages
Remedy	The client consults metadata provided by the service to determine which operations are supported.

5972 **Table 7 – wsman:AlreadyExists**

Fault Subcode	wsman:AlreadyExists
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The sender attempted to create a resource that already exists.
Detail	None
Comments	This fault is returned in cases where the user attempted to create a resource that already exists.
Applicability	Create
Remedy	The client uses Put or creates a resource with a different identity.

Table 8 – wsmen:CannotProcessFilter

Fault Subcode	wsmen:CannotProcessFilter
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The requested filter could not be processed.
Detail	<s:Detail> <wsman:SupportedSelectorName> Valid selector name for use in filter expression </wsman:SupportedSelectorName> * </s:Detail>
Comments	This fault is returned for syntax errors or other semantic problems with the filter. For use with the SelectorFilter dialect (see ANNEX E), the service can include one or more SupportedSelectorName elements to provide a list of supported selector names in the event that the client has requested filtering on one or more unsupported selector names. If the filter is valid, but the service cannot execute the filter due to misconfiguration, lack of resources, or other service-related problems, more specific faults can be returned, such as wsman:QuotaLimit or wsman:InternalError.
Applicability	Enumerate
Remedy	The client fixes the filter problem and tries again.

Table 9 – wsman:CannotProcessFilter

Fault Subcode	wsman:CannotProcessFilter
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The requested filter could not be processed.
Detail	<s:Detail> <wsman:SupportedSelectorName> Valid selector name for use in filter expression </wsman:SupportedSelectorName> * </s:Detail>
Comments	This fault is returned for syntax errors or other semantic problems with the filter such as exceeding the subset supported by the service. For use with the SelectorFilter dialect (see ANNEX E), the service can include one or more SupportedSelectorName elements to provide a list of supported selector names in the event that the client has requested filtering on one or more unsupported selector names. If the filter is valid, but the service cannot execute the filter due to misconfiguration, lack of resources, or other service-related problems, more specific faults can be returned, such as wsman:QuotaLimit, wsman:InternalError, or wsme:EventSourceUnableToProcess.
Applicability	Subscribe, fragment-level resource access operations
Remedy	The client fixes the filter problem and tries again.

5975 **Table 10 – wsman:Concurrency**

Fault Subcode	wsman:Concurrency
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The action could not be completed due to concurrency or locking problems.
Detail	None
Comments	This fault means that the requested action could not be carried out either due to internal concurrency or locking problems or because another user is accessing the resource. This fault can occur if a resource is being enumerated using Enumerate and another client attempts operations such as Delete, which would affect the result of the enumeration in progress.
Applicability	All messages
Remedy	The client waits and tries again.

5976 **Table 11 – wsme:DeliveryModeRequestedUnavailable**

Fault Subcode	wsme:DeliveryModeRequestedUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The requested delivery mode is not supported.

Detail	`<s:Detail>`
	`  <wsme:SupportedDeliveryMode>... </wsme:SupportedDeliveryMode>`
	`  <wsme:SupportedDeliveryMode>...</wsme:SupportedDeliveryMode>`
	`   ...`
	`</s:Detail>`
	`<!-- This is a simple, optional list of one or more supported delivery mode URIs. It may be left empty. -->`

Comments	This fault is returned for unsupported delivery modes for the specified resource. If the stack supports the delivery mode in general, but not for the specific resource, this fault is still returned. Other resources might support the delivery mode. The fault does not imply that the delivery mode is not supported by the implementation.
Applicability	Subscribe
Remedy	The client selects one of the supported delivery modes.

5977 **Table 12 – wsman:DeliveryRefused**

Fault Subcode	wsman:DeliveryRefused
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Receiver
Reason	The receiver refuses to accept delivery of events and requests that the subscription be canceled.
Detail	None
Comments	This fault is returned by event receivers to force a cancellation of a subscription. This fault can happen when the client tried to Unsubscribe, but failed, or when the client lost knowledge of active subscriptions and does not want to keep receiving events that it no longer owns. This fault can help clean up spurious or leftover subscriptions when clients are reconfigured or reinstalled and their previous subscriptions are still active.
Applicability	Any event delivery message in any mode
Remedy	The service stops delivering events for the subscription and cancels the subscription, sending any applicable SubscriptionEnd messages.

5978 **Table 13 – wsa:DestinationUnreachable**

Fault Subcode	wsa:DestinationUnreachable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	No route can be determined to reach the destination role defined by the Addressing To header.
Detail	<s:Detail> <wsman:FaultDetail> http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI </wsman:FaultDetail> ? </s:Detail> When the default addressing model is in use, the wsman:FaultDetail field may contain http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI.
Comments	This fault is returned as the general "Not Found" case for a resource, in which the resource EPR cannot be mapped to the real-world resource. This fault is not used merely to indicate that the resource is temporarily offline, which is indicated by wsa:EndpointUnavailable.
Applicability	All request messages
Remedy	The client attempts to diagnose the version of the service, query any metadata, and perform other diagnostic operations to determine why the request cannot be routed.

5979 **Table 14 – wsman:EncodingLimit**

Fault Subcode	wsman:EncodingLimit
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	An internal encoding limit was exceeded in a request or would be violated if the message were processed.
Detail	<s:Detail> <wsman:FaultDetail> Optional; one of the following enumeration values </wsman:FaultDetail> ...any service-specific additional XML content... </s:Detail> Possible enumeration values in the <wsman:FaultDetail> element are as follows: Unsupported character set: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/CharacterSet Unsupported MTOM or other encoding types: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EncodingType Requested maximum was too large: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize Requested maximum envelope size was too small: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MinimumEnvelopeLimit Too many options: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionLimit Used when the default addressing model is in use and indicates that too many selectors were used for the corresponding ResourceURI: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/SelectorLimit Service reached its own internal limit when computing response: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit Operation succeeded and cannot be reversed, but result is too large to send: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess Request contained a character outside of the range that is supported by the service: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnsupportedCharacter URI was too long: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/URILimitExceeded Client-side whitespace usage is not supported: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Whitespace
Comments	This fault is returned when a system limit is exceeded, whether a published limit or a service-specific limit.
Applicability	All request messages
Remedy	The client sends messages that fit the encoding limits of the service.

5980 **Table 15 – wsa:EndpointUnavailable**

Fault Subcode	wsa:EndpointUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Receiver
Reason	The specified endpoint is currently unavailable.

Detail	<s:Detail> <wsa:RetryAfter> *xs:duration* </wsa:RetryAfter> <!-- optional --> ...optional service-specific XML content <wsman:FaultDetail> *A detail URI value* </wsman:FaultDetail> </s:Detail> http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ResourceOffline Used when the resource is known, but temporarily unavailable
Comments	This fault is returned if the message was correct and the EPR was valid, but the specified resource is offline. In practice, it is difficult for a service to distinguish between "Not Found" cases and "Offline" cases. In general, wsa:DestinationUnreachable is preferable.
Applicability	All request messages
Remedy	The client can retry later, after the resource is again online.

5981 **Table 16 – wsman:EventDeliverToUnusable**

Fault Subcode	wsman:EventDeliverToUnusable
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The event source cannot process the subscription because it cannot connect to the event delivery endpoint as requested in the Delivery element.
Detail	<s:Detail> ...any service-specific content to identify the error... </s:Detail>
Comments	This fault is limited to cases of connectivity issues in contacting the “deliver to” address. These issues include: • The NotifyTo address is not usable because it is incorrect (system or device not reachable, badly formed address, and so on). • Permissions cannot be acquired for event delivery (for example, the wsman:Auth element does not refer to a supported security profile, and so on). • The credentials associated with the NotifyTo are not valid (for example, the account does not exist, the certificate thumbprint is not a hex string, and so on). The service can include extra information that describes the connectivity error to help in troubleshooting the connectivity problem.
Applicability	Subscribe
Remedy	The client ensures connectivity from the service computer back to the event sink including firewalls and authentication/authorization configuration.

5982 **Table 17 – wsme:EventSourceUnableToProcess**

Fault Subcode	wsme:EventSourceUnableToProcess
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Receiver
Reason	The event source cannot process the subscription.
Detail	None
Comments	This event source is not capable of fulfilling a Subscribe request for local reasons unrelated to the specific request.
Applicability	Subscribe
Remedy	The client retries the subscription later.

5983 **Table 18 – wsmen:FilterDialectRequestedUnavailable**

Fault Subcode	wsmen:FilterDialectRequestedUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The requested filtering dialect is not supported.

Detail	`<s:Detail>`
	`  <wsmen:SupportedDialect> .... </wsmen:SupportedDialect> +`
	`</s:Detail>`

Comments	This fault is returned when the client requests a filter type or query language not supported by the service. The filter dialect can vary from resource to resource or can apply to the entire service.
Applicability	Enumerate
Remedy	The client switches to a supported dialect or performs a simple enumeration with no filter.

5984 **Table 19 – wsme:FilteringNotSupported**

Fault Subcode	wsme:FilteringNotSupported
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	Filtering over the event source is not supported.
Detail	None
Comments	This fault is returned when the service does not support filtered subscriptions for the specified event source, but supports only simple delivery of all events for the resource. NOTE: The service might support filtering over a different event resource or might not support filtering for *any* resource. The same fault applies.
Applicability	Subscribe
Remedy	The client subscribes using unfiltered delivery.

5985 **Table 20 – wsmen:FilteringNotSupported**

Fault Subcode	wsmen:FilteringNotSupported
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	Filtered enumeration is not supported.
Detail	None
Comments	This fault is returned when the service does not support filtering of enumerations at all, but supports only simple enumeration. If enumeration as a whole is not supported, the correct fault is wsa:ActionNotSupported. NOTE: The service might support filtering over a different enumerable resource or might not support filtering for *any* resource. The same fault applies.
Applicability	Enumerate
Remedy	The client switches to a simple enumeration.

5986 **Table 21 – wsme:FilteringRequestedUnavailable**

Fault Subcode	wsme:FilteringRequestedUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The requested filter dialect is not supported.
Detail	<s:Detail> <wsme:SupportedDialect>.. </wsme:SupportedDialect> + <wsman:FaultDetail> ..the following URI, if applicable </wsman:FaultDetail> </s:Detail> Possible URI value: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired
Comments	This fault is returned when the client requests a filter dialect not supported by the service. In some cases, a subscription *requires* a filter because the result of an unfiltered subscription may be infinite or extremely large. In these cases, the URI http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired needs to be included in the s:Detail element.
Applicability	Subscribe
Remedy	The client switches to a supported filter dialect or uses no filtering.

5987 **Table 22 – wsman:FragmentDialectNotSupported**

Fault Subcode	wsman:FragmentDialectNotSupported
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The requested fragment filtering dialect or language is not supported.
Detail	<s:Detail> <wsman:FragmentDialect> xs:anyURI </wsman:FragmentDialect> <wsman:FragmentDialect> xs:anyURI </wsman:FragmentDialect> .... </s:Detail> The preceding optional URI values indicate supported dialects.
Comments	This fault is returned when the service does not support the requested fragment-level filtering dialect. If the implementation supports the fragment dialect in general, but not for the specific resource, this fault is still returned. Other resources might support the fragment dialect. This fault does not imply that the fragment dialect is not supported by the implementation.
Applicability	Enumerate, Get, Create, Put, Delete
Remedy	The client uses a supported filtering dialect or no filtering.

5988 **Table 23 – wsman:InternalError**

Fault Subcode	wsman:InternalError
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Receiver
Reason	The service cannot comply with the request due to internal processing errors.
Detail	<s:Detail> ...service-specific extension XML elements.... <s:Detail>
Comments	This fault is a generic error for capturing internal processing errors within the service. For example, this is the correct fault if the service cannot load necessary executable images, its configuration is corrupted, hardware is not operating properly, or any unknown or unexpected internal errors occur. It is expected that the service needs to be reconfigured, restarted, or reinstalled, so merely asking the client to retry will not succeed.
Applicability	All messages
Remedy	The client repairs the service out-of-band to WS-Management.

5989 **Table 24 – wsman:InvalidBookmark**

Fault Subcode	wsman:InvalidBookmark
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The bookmark supplied with the subscription is not valid.

Detail	`<s:Detail>`
	`<wsman:FaultDetail>`
	If possible, one of the following URI values
	`</wsman:FaultDetail>`
	`</s:Detail>`
	Possible URI values:
	The service is not able to back up and replay from that point:
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired
	The service is not able to decode the bookmark:
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFormat

Comments	This fault is returned if a bookmark has expired, is corrupt, or is otherwise unknown.
Applicability	Subscribe
Remedy	The client issues a new subscription without any bookmarks or locates the correct bookmark.

5990 **Table 25 – wsmen:InvalidEnumerationContext**

Fault Subcode	wsmen:InvalidEnumerationContext
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Receiver
Reason	The supplied enumeration context is invalid.
Detail	None
Comments	An invalid enumeration context was supplied with the message. Typically, this fault will happen with Pull. The enumeration context may be invalid due to expiration, an invalid format, or reuse of an old context no longer being tracked by the service. The service also can return this fault for any case where the enumerator has been terminated unilaterally on the service side, although one of the more descriptive faults is preferable, because this usually happens on out-of-memory errors (wsman:QuotaLimit), authorization failures (wsman:AccessDenied), or internal errors (wsman:InternalError).
Applicability	Pull, Release (whether a pull-mode subscription, or a normal enumeration)
Remedy	The client abandons the enumeration and lets the service time it out, because Release will fail as well.

5991 **Table 26 – wsme:InvalidExpirationTime**

Fault Subcode	wsme:InvalidExpirationTime
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The expiration time is not valid.
Detail	None
Comments	The expiration time is not valid at all or within the limits of the service. This fault is used for outright errors (expirations in the past, for example) or expirations too far into the future. If the service does not support expiration times at all, a wsman:UnsupportedFeature fault can be returned with the correct detail code.
Applicability	Subscribe
Remedy	The client issues a new subscription with a supported expiration time.

5992 **Table 27 – wsmen:InvalidExpirationTime**

Fault Subcode	wsmen:InvalidExpirationTime
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The expiration time is not valid.
Detail	None
Comments	Because WS-Management recommends against implementing the Expiration feature, this fault might not occur with most implementations. See clause 8 for more information.
Applicability	Enumerate
Remedy	Not applicable

5993 **Table 28 – wsme:InvalidMessage**

Fault Subcode	wsme:InvalidMessage
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The request message has unknown or invalid content and cannot be processed.
Detail	None
Comments	This fault is generally not used in WS-Management, although it can be used for cases not covered by other faults. If the content violates the schema, a wsman:SchemaValidationError fault can be sent. If specific errors occur in the subscription body, one of the more descriptive faults can be used. This fault is not to be used to indicate unsupported features, only unexpected or unknown content in violation of this specification.
Applicability	Pub/sub request messages
Remedy	The client issues valid messages that comply with this specification.

5994 **Table 29 – wsa:InvalidMessageInformationHeader**

Fault Subcode	wsa:InvalidMessageInformationHeader
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	A message information header is not valid, and the message cannot be processed.
Detail	<s:Detail> ...the invalid header... </s:Detail>
Comments	This fault can occur with any type of SOAP header error. The header might be invalid in terms of schema or value, or it might constitute a semantic error. This fault is not to be used to indicate an invalid resource address (a "not found" condition for the resource), but to indicate actual structural violations of the SOAP header rules in this specification. Examples are repeated MessageIDs, missing RelatesTo on a response, badly formed addresses, or any other missing header content.
Applicability	All messages
Remedy	The client reformats message using the correct format, values, and number of message information headers.

Table 30 – wsman:InvalidOptions

Fault Subcode	wsman:InvalidOptions
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	One or more options are not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue
Comments	This fault generically covers all cases where the option names or values are not valid, or they are used in incorrect combinations.
Applicability	All request messages
Remedy	The client discovers supported option names and valid values by consulting metadata or other mechanisms. Such metadata is beyond the scope of this specification.

Table 31 – wsman:InvalidParameter

Fault Subcode	wsman:InvalidParameter
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	An operation parameter is not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName
Comments	This fault is returned when a parameter to a custom action is not valid. This fault is a default for new implementations that need to have a generic fault for this case. The method can also return any specific fault of its own.
Applicability	All messages with custom actions
Remedy	The client consults the WSDL for the operation and determines how to supply the correct parameter.

Table 32 – wsmt:InvalidRepresentation

Fault Subcode	wsmt:InvalidRepresentation
Action URI	http://schemas.xmlsoap.org/ws/2004/09/transfer/fault
Code	s:Sender
Reason	The XML content is not valid.

Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFragment
Comments	This fault may be returned when the input XML is not valid semantically or uses the wrong schema for the resource. However, a wsman:SchemaValidationError fault can be returned if the error is related to XML schema violations as such, as opposed to invalid semantic values. Note the anomalous case in which a schema violation does not occur, but the namespace is simply the wrong one; in this case, http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace is returned.
Applicability	Put, Create
Remedy	The client corrects the request XML.

5998 **Table 33 – wsman:InvalidSelectors**

Fault Subcode	wsman:InvalidSelectors
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The selectors for the resource are not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelectors http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelectors http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DuplicateSelectors
Comments	This fault covers all cases where the specified selectors were incorrect or unknown for the specified resource.
Applicability	All request messages
Remedy	The client retrieves documentation or metadata and corrects the selectors.

Table 34 – wsa:MessageInformationHeaderRequired

Fault Subcode	wsa:MessageInformationHeaderRequired
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	A required header is missing.
Detail	<s:Detail> The XML QName of the missing header </s:Detail>
Comments	A required message information header (To, MessageID, or Action) is not present.
Applicability	All messages
Remedy	The client adds the missing message information header.

Table 35 – wsman:NoAck

Fault Subcode	wsman:NoAck
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The receiver did not acknowledge the event delivery.
Detail	None
Comments	This fault is returned when the client (subscriber) receives an event with a wsman:AckRequested header and does not (or cannot) acknowledge the receipt. The service stops sending events and terminates the subscription.
Applicability	Any event delivery action (including heartbeats, dropped events, and so on) in any delivery mode
Remedy	For subscribers, the subscription is resubmitted without the acknowledgement option. For services delivering events, the service cancels the subscription immediately.

Table 36 – wsman:QuotaLimit

Fault Subcode	wsman:QuotaLimit
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The service is busy servicing other requests.
Detail	None
Comments	This fault is returned when the SOAP message is otherwise correct, but the service has reached a resource or quota limit.
Applicability	All messages
Remedy	The client can retry later.

6002 **Table 37 – wsman:SchemaValidationError**

Fault Subcode	wsman:SchemaValidationError
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The supplied SOAP violates the corresponding XML schema definition.
Detail	None
Comments	This fault is used for any XML parsing failure or schema violations. Full validation of the SOAP against schemas is not expected in real-time, but processors might in fact notice schema violations, such as type mismatches. In all of these cases, this fault applies. In debugging modes where validation is occurring, this fault can be returned for *all* errors noted by the validating parser.
Applicability	All messages
Remedy	The client corrects the message.

6003 **Table 38 – wsmen:TimedOut**

Fault Subcode	wsmen:TimedOut
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Receiver
Reason	The enumerator has timed out and is no longer valid.
Detail	None
Comments	This fault is not to be used in WS-Management due to overlap with wsman:TimedOut, which covers all the other messages.
Applicability	Pull
Remedy	The client can retry the Pull request.

6004 **Table 39 – wsman:TimedOut**

Fault Subcode	wsman:TimedOut
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Receiver
Reason	The operation has timed out.
Detail	None
Comments	The operation could not be completed within the wsman:OperationTimeout value, or an internal override timeout was reached by the service while trying to process the request. This fault is also returned in all enumerations when no content is available for the current Pull request. Clients can simply retry the Pull request again until a different fault is returned.
Applicability	All requests
Remedy	The client can retry the operation. If the operation is a write (delete, create, or custom operation), the client can consult the system operation log before blindly attempting a retry or attempt a Get or other read operation to try to discover the result of the previous operation.

6005 **Table 40 – wsme:UnableToRenew**

Fault Subcode	wsme:UnableToRenew
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The subscription could not be renewed.
Detail	None
Comments	This fault is returned in all cases where the subscription cannot be renewed but is otherwise valid.
Applicability	wsme:Renew
Remedy	The client issues a new subscription.

6006 **Table 41 – wsme:UnsupportedExpirationType**

Fault Subcode	wsme:UnsupportedExpirationType
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The specified expiration type is not supported.
Detail	None
Comments	A specific time for expiration (as opposed to duration) is not supported. This fault is not to be used if the value itself is incorrect; it is only to be used if the *type* is not supported.
Applicability	Subscribe
Remedy	The client corrects the expiration to use a duration time.

6007 **Table 42 – wsmen:UnsupportedExpirationType**

Fault Subcode	wsmen:UnsupportedExpirationType
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The specified expiration type is not supported.
Detail	None
Comments	The specified expiration type is not supported. For example, a specific time-based expiration type might not be supported (as opposed to a duration-based expiration type). This fault is not to be used if the value itself is incorrect; it is only to be used if the *type* is not supported.
Applicability	Enumerate
Remedy	The client corrects the expiration time or omits it and retries.

6008 **Table 43 – wsman:UnsupportedFeature**

Fault Subcode	wsman:UnsupportedFeature
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The specified feature is not supported.

Detail	`<s:Detail>`
	`<wsman:FaultDetail>`
	If possible, one of the following URI values
	`</wsman:FaultDetail>`
	`</s:Detail>`
	Possible URI values:
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AsynchronousRequest
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Bookmarks
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DeliveryRetries
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EnumerationMode
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAccess
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OperationTimeout

Comments	This fault indicates that an unsupported feature was attempted.
Applicability	Any message
Remedy	The client corrects or removes the unsupported feature request and retries.

6009 **Table 44 – wsme:UnsupportedExpirationType**

Fault Subcode	wsme:UnsupportedExpirationType
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	Only expiration durations are supported.
Detail	None
Comments	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.
Applicability	Subscribe, wsme:Renew

Remedy	

6010 **Table 45 – wsmen:UnableToRenew**

Fault Subcode	wsmen:UnableToRenew
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	*Text explaining the failure; e.g., "The event source has too many subscribers".*
Detail	None
Comments	This fault is sent when the event source is not capable of fulfilling a Renew request for local reasons unrelated to the specific request.
Applicability	wsmen:Renew
Remedy	

6011 **Table 46 – wsa:InvalidMessage**

Fault Subcode	wsa:InvalidMessage
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	*The message is not valid and cannot be processed.*
Detail	*The invalid message*
Comments	If a request message does not comply with the corresponding outline in the previous row, the request shall fail and the event source or subscription manager may generate this fault indicating that the request is invalid.
Applicability	Subscribe, Renew, wsme:GetStatus, Unsubscribe
Remedy	

6012 **Table 47 – wsme:CannotProcessFilter**

Fault Subcode	wsme:CannotProcessFilter
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	*Cannot filter as requested*
Detail	None
Comments	A filter was specified can not be processed.
Applicability	Subscribe
Remedy	

6013

ANNEX A
(informative)

Notational Conventions

This annex specifies the notations and namespaces used in this specification.

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 "|" indicates a choice between alternatives.
- The characters "[" and "]" indicate that enclosed items are to be treated as a group with respect to cardinality or choice.
- An ellipsis ("...") indicates a point of extensibility that allows other child or attribute content. Additional children and attributes may be added at the indicated extension points but must not contradict the semantics of the parent 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 A-1) indicate the namespace of the element being defined.

Throughout the document, whitespace within XML element values is used for readability. In practice, a service can accept and strip leading and trailing whitespace within element values as if whitespace had not been used.

A.1 XML Namespaces

Table A-1 lists XML namespaces used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant. Unless otherwise noted, the XML Schema for each specification can be retrieved by resolving the XML namespace URI for each specification listed in Table A-1.

Table A-1 – Prefixes and XML Namespaces Used in This Specification

Prefix	XML Namespace	Specification
wsman	http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd	This specification
wsmid	http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd	This specification – discovery of supported protocol versions
s	http://www.w3.org/2003/05/soap-envelope	*SOAP 1.2*
xs	http://www.w3.org/2001/XMLSchema	*XML Schema 1*, *XML Schema 2*
wsdl	http://schemas.xmlsoap.org/wsdl	WSDL/1.1
wsa	Either wsa04 or wsa10	Either wsa04 or wsa10
wsa04	http://schemas.xmlsoap.org/ws/2004/08/addressing	Clause 5 of this specification
wsa10	http://www.w3.org/2005/08/addressing	*WS-Addressing W3C Recommendation*
wsam	http://www.w3.org/2007/05/addressing/metadata	*WS-Addressing Metadata W3C Recommendation*
wsme	http://schemas.xmlsoap.org/ws/2004/08/eventing	Clause 10 of this specification
wsmen	http://schemas.xmlsoap.org/ws/2004/09/enumeration	Clause 8 of this specification
wsmt	http://schemas.xmlsoap.org/ws/2004/09/transfer	Clause 7 of this specification
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	*WS-Policy*

ANNEX B
(normative)

Conformance

This annex specifies the conformance rules used in this specification.

An implementation is not conformant with this specification if it fails to satisfy one or more of the "shall" or "required" level requirements defined in the conformance rules for each section, as indicated by the following format:

R*nnnn*: Rule text

General conformance rules are defined as follows:

RB-1: To be conformant, the service shall comply with all the rules defined in this specification. Items marked with shall are required, and items marked with should are highly advised to maximize interoperation. Items marked with may indicate the preferred implementation for expected features, but interoperation is not affected if they are ignored.

RB-2: Conformant services of this specification shall use this XML namespace Universal Resource Identifier:

```
(1)   http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
```

RB-3: A SOAP node shall not use the XML namespace identifier for this specification unless it complies with the conformance rules in this specification.

This specification does not mandate that all messages and operations need to be supported. It only requires that any supported message or operation obey the conformance rules for that message or operation. It is important that services not use the XML namespace identifier for WS-Management in SOAP operations in a manner that is inconsistent with the rules defined in this specification.

ANNEX C
(normative)

HTTP(S) Transport and Security Profile

C.1 General

Although WS-Management is a SOAP protocol and not tied to a specific network transport, interoperation requires some common standards to be established. This clause centers on establishing common usage over HTTP 1.1 and HTTPS. In addition to HTTP and HTTPS, this specification allows any SOAP-enabled transport to be used as a carrier for WS-Management messages.

For identification and referencing, each transport is identified by a URI, and each authentication mechanism defined in this specification is also identified by a URI.

As new transports are standardized, they can also acquire a URI for referencing purposes, and any new authentication mechanisms that they expose can also be assigned URIs for publication and identification purposes in XML documents. As new transports are standardized for WS-Management, the associated transport-specific requirements can be defined and published to ensure interoperability.

The SOAP HTTP binding described in section 7 of SOAP Version 1.2 Part 2: Adjuncts is used for WS-Management encoding over HTTP and HTTPS.

C.2 HTTP(S) Binding

This clause clarifies how SOAP messages are bound to HTTP(S).

RC.2-1: A service that supports the SOAP HTTP(S) binding shall at least support it using HTTP 1.1.

RC.2-2: A service shall at least implement the Responding SOAP Node of the SOAP Request-Response Message Exchange Pattern:

http://www.w3.org/2003/05/soap/mep/request-response/

RC.2-3: A service may choose not to implement the Responding SOAP Node of the SOAP Response Message Exchange Pattern:

http://www.w3.org/2003/05/soap/mep/soap-response/

RC.2-4: A service may choose not to support the SOAP Web Method Feature.

RC.2-5: A service shall at least implement the Responding SOAP Node of an HTTP one-way Message Exchange Pattern where the SOAP Envelope is carried in the HTTP Request and the HTTP Response has a Status Code of 202 Accepted and an empty Entity Body (no SOAP Envelope).

The message exchange pattern described in RB.2-5 is used to carry SOAP messages that require no response.

RC.2-6: A service shall at least support Request Message SOAP Envelopes and one-way SOAP Envelopes delivered using HTTP Post.

RC.2-7: In cases where the service cannot respond with a SOAP message, the HTTP error code 500 (Internal Server Error) should be returned and the client side should close the connection.

RC.2-8: For services that support HTTPS, the transport layer handles negotiation of the proper encryption protocol. Services may implement an Identify response that is unauthenticated to facilitate negotiation. **RC.2-9:** When delivering faults, an HTTP status code of 500 should be used in the response for s:Receiver faults, and a code of 400 should be used for s:Sender faults.

RC.2-10: The URL used with the HTTP-Post operation to deliver the SOAP message is not required to have the same content as the wsa:To URI used in the SOAP address. Often, the HTTP URL has the same content as the wsa:To URI in the message, but may additionally contain other message routing fields suffixed to the network address using a service-defined separator token sequence. It is recommended that services require only the wsa:To network address URL to promote uniform client-side processing and behavior, and to include service-level routing in other parts of the address.

RC.2-11: In the absence of other requirements, it is recommended that the path portion of the URL used with the HTTP-POST operation be /wsman for resources that require authentication and /wsman-anon for resources that do not require authentication. If these paths are used, unauthenticated requests should not be supported for /wsman and authentication must not be required for /wsman-anon.

RC.2-12: If the SOAPAction header is present in an HTTP/HTTPS-based request that carries a SOAP message, it must match the wsa:Action URI present in the SOAP message. The SOAPAction header is optional, and a service must not fault a request if this header is missing.

Because WS-Management is based on SOAP 1.2, the optional SOAPAction header is merely used as an optimization. If present, it shall match the wsa:Action URI used in the SOAP message. The service is permitted to fault the request by simply examining the SOAPAction header, if the action is not valid, without examining the SOAP content. However, the service may not fault the request if the SOAPAction header is omitted.

RC.2-13: If a service supports attachments, the service shall support the HTTP Transmission Optimization Feature.

RC.2-14: If a service cannot process a message with an attachment or unsupported encoding type, and the transport is HTTP or HTTPS, it shall return HTTP error 415 as its response (unsupported media).

RC.2-15: If a service cannot process a message with an attachment or unsupported encoding type using transports other than HTTP/HTTPS, it should return a wsman:EncodingLimit fault with the following detail code:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EncodingType

C.3 HTTP(S) Security Profiles

This specification defines a set of security profiles for use with HTTP and HTTPS. Conformant services need not support HTTP or HTTPS, but if supported these predefined profiles provide the client with at least one way to access the service. Other specifications can define additional profiles for use with HTTP or HTTPS.

RC.3-1: A conformant service that supports HTTP shall support one of the predefined HTTP-based profiles.

RC.3-2: A conformant service that supports HTTPS shall support one of the predefined HTTPS-based profiles.

RC.3-3: A conformant service should not expose WS-Management over a completely unauthenticated HTTP channel except for situations such as Identify (see clause 11), debugging, or as determined by the service.

The service is not required to export only a single HTTP or HTTPS address. The service can export multiple addresses, each of which supports a specific security profile or multiple profiles.

If clients support all predefined profiles, they are assured of some form of secure access to a WS-Management implementation that supports HTTP, HTTPS, or both.

C.3.1 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic

This profile is essentially the "standard" profile, but it is limited to Basic authentication.

The typical sequence is shown in Table C-1.

Table C-1 – Basic Authentication Sequence

	Client		Service
1	Client connects with no authorization header.	→	Service sees no header.
2		←	Service sends 401 return code, listing Basic as the authorization mode.
3	Client provides Basic authorization header.	→	Service authenticates the client.

This behavior is normal for HTTP. If the client connects with a Basic authorization header initially and if it is valid, the request immediately succeeds.

Basic authentication is not recommended for unsecured transports. If used with HTTP alone, for example, the transmission of the password constitutes a security risk. However, if the HTTP transport is secured with IPSec, for example, the risk is substantially reduced.

Similarly, Basic authentication is suitable when performing testing, prototyping, or diagnosis.

C.3.2 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest

This profile is essentially the same as the "standard" profile, but it is limited to the use of Digest authentication.

The typical sequence is shown in Table C-2.

Table C-2 – Digest Authentication Sequence

	Client		Service
1	Client connects with no authorization header.	→	Service sees no header.
2		←	Service sends 401 return code, listing Digest as the authorization mode.
3	Client provides Digest authorization header.	→	
4		←	Service begins authorization sequence of secure token exchange.
5	Client continues authorization sequence.	→	Service authenticates client.

This behavior is normal for HTTP. If the client connects with a Digest authorization header initially and if it is valid, the token exchange sequence begins.

C.3.3 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/basic

This profile establishes the use of Basic authentication over HTTPS. This profile is used when only a server-side certificate encrypts the connection, but the service still needs to authenticate the client.

The typical sequence is shown in Table C-3.

Table C-3 – Basic Authentication over HTTPS Sequence

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service sees no header, but establishes an encrypted connection.
2		←	Service sends 401 return code, listing Basic as the authorization mode.
3	Client provides Basic authorization header.	→	Service authenticates the client.

If the client connects with a Basic authorization header initially and if it is valid, the request immediately succeeds.

C.3.4 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/digest

This profile establishes the use of Digest authentication over HTTPS. This profile is used when only a server-side certificate encrypts the connection, but the service still needs to authenticate the client.

The typical sequence is shown in Table C-4.

Table C-4 – Digest Authentication over HTTPS Sequence

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service sees no header, but establishes an encrypted connection.

2		←	Service sends 401 return code, listing Digest as the auth mode.
3	Client provides Digest authorization header.	→	
4		←	Service begins authorization sequence of secure token exchange.
5	Client continues authorization sequence.	→	Service authenticates client.

This behavior is normal for HTTPS. If the client connects with a Digest authorization header initially and if it is valid, the token exchange sequence begins.

C.3.5 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual

In this security mode, the client supplies an X.509 certificate that is used to authenticate the client. No HTTP or HTTPS authorization header is required in the HTTP-Post request.

However, as a hint to the service, the following HTTP/HTTPS authorization header may be present.

Authorization: http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual

Because the service can be configured to always look for the certificate, this authorization header is not required.

This simple sequence is shown in Table C-5.

Table C-5 – HTTPS with Client Certificate Sequence

	Client		Service
1	Client connects with no authorization header but supplies an X.509 certificate.	→	Service ignores the authorization header and retrieves the client-side certificate.
2		←	Service accepts or denies access with 403.7 or 403.16 return codes.

C.3.6 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic

In this profile, the http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual profile is used first to authenticate both sides using X.509 certificates. Individual operations are subsequently authenticated using HTTP Basic authorization headers.

This profile authenticates both the client and service initially and provides one level of security, typically at the machine or device level. The second level of authentication typically performs authorization for specific operations, although it can act as a simple, secondary authentication mechanism with no authorization semantics.

The typical sequence is shown in Table C-6.

Table C-6 – Basic Authentication over HTTPS with Client Certificate Sequence

	Client		Service
1	Client connects with certificate and special authorization header.	→	Service queries for client certificate and authenticates. If certificate is missing or invalid, the sequence stops here with 403.7 or 403.16 return codes.

2		←	After authenticating the certificate, the service sends 401 return code, listing available Basic authorization mode as a requirement.
3	Client selects Basic as the authorization mode to use and includes it in the Authorization header, as defined for HTTP 1.1.	→	Service authenticates the client again before performing the operation.

In the initial request, the HTTPS authorization header must be as follows:

Authorization: http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic

This indicates to the service that this special mode is in use, and that it can query for the client certificate to ensure that subsequent requests are properly challenged for Basic authorization if the HTTP Authorization header is missing from a request.

The Authorization header is treated as normal HTTP basic:

Authorization: Basic ...user/password encoding

This use of Basic authentication is secure (unlike its normal use in HTTP) because the transmission of the user name and password is performed over an encrypted connection.

C.3.7 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/digest

This profile is the same as http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic, except that the HTTP Digest authentication model is used after the initial X.509 certificate-based mutual authentication is completed.

In the initial request, the HTTPS authorization header must be as follows:

Authorization:
http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/digest

C.3.8 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego-kerberos

In this profile, the client connects to the server using HTTPS with only server-side certificates to encrypt the connection.

Authentication is carried out based on RFC 4559, which describes the use of GSSAPI SPNEGO over HTTP (Table C-7). This mechanism allows HTTP to carry out the negotiation protocol of RFC 4178 to authenticate the user based on Kerberos Version 5.

Table C-7 – SPNEGO Authentication over HTTPS Sequence

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service sees no header, but establishes an encrypted connection.
2		←	Service sends 401 return code, listing **Negotiate** as an available HTTP authentication mechanism.
3	Client uses the referenced Internet draft to start a SPNEGO sequence to negotiate for Kerberos V5.	→	...

4	...	←	Service engages in SPNEGO sequence to authenticate client using Kerberos V5.
5	Client is authenticated.	→	Service authenticates client.

C.3.9 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/spnego-kerberos

This mode is the same as http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego-kerberos except that the server and client mutually authenticate one another at the transport layer prior to beginning the Kerberos authentication sequence (Table C-8). See RFC 4178 for details.

Table C-8 – SPNEGO Authentication over HTTPS with Client Certificate Sequence

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service queries for client certificate and authenticates. If certificate is missing or invalid, the sequence stops here with 403.7 or 403.16 return codes.
2		←	After the mutual certificate authentication sequence, service sends 401 return code, listing **Negotiate** as an available HTTP authentication mechanism.
3	Client uses the referenced Internet draft to start a SPNEGO sequence to negotiate for Kerberos V5.	→	...
4	...	←	Service engages in SPNEGO sequence to authenticate client using Kerberos V5.
5	Client is authenticated.	→	Service authenticates client.

Typically, this is used to mutually authenticate devices or machines, and then subsequently perform user- or role-based authentication.

C.3.10 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/spnego-kerberos

This profile is the same as http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego-kerberos except that it is performed over an HTTP connection. See RFC 4178 for details.

Although this profile supports secure authentication, because it is not encrypted, it represents security risks such as information disclosure because the SOAP traffic is in plain text. It is not to be used in environments that require a high level of security.

C.4 IPSec and HTTP

HTTP with Basic authentication is weak on an unsecured network. If IPSec is in use, however, this weakness is no longer an issue. IPSec provides high-quality cryptographic security, data origin authentication, and anti-replay services.

Because IPSec is intended for machine-level authentication and network traffic protection, it is insufficient for real-world management in many cases, which can require additional authentication of specific users to authorize access to resource classes and instances. IPSec needs to be used in conjunction with one of the profiles in this clause for user-level authentication. However, it obviates the need for HTTPS-based traffic and allows safe use of HTTP-based profiles.

From the network perspective, the use of HTTP Basic authentication when the traffic is carried over a network secured by IPSec is intrinsically safe and equivalent to using HTTPS with server-side certificates. Other specifications can define IPSec security profiles that combine IPSec with appropriate authentication mechanisms.

ANNEX D
(informative)

XPath Support

D.1 General

Implementations typically need to support XPath for several purposes, such as fragment-level access (7.7), datasets (8), and filtering (10.2.2). Because the full XPath 1.0 specification is large, subsets are typically required in resource-constrained implementations.

The purpose of this clause is to identify the minimum set of syntactic elements that implementations can provide to promote maximum interoperability. In most cases, implementations provide large subsets of full XPath, but they need additional definitions to ensure that the subsets meet minimum requirements. The Level 1 and Level 2 BNF definitions in this annex establish such minimums for use in the WS-Management space.

This specification defines two subset profiles for XPath: Level 1 with basic node selector support and no filtering (for supporting Fragment-level access as described in 7.7), and Level 2 with basic filtering support (for enumerating and receving notifications). Level 2 is a formal superset of Level 1.

The following BNFs both are formal LL(1) grammars. A parser can be constructed automatically from the BNF using an appropriate tool, or a recursive-descent parser can be implemented manually by inspection of the grammar.

Within the grammars, non-terminal tokens are surrounded by angled brackets, and terminal tokens are in uppercase and not surrounded by angled brackets.

XML namespace support is explicitly absent from these definitions. Processors that meet the syntax requirements can provide a mode in which the elements are processed without regard to XML namespaces, but can also provide more powerful, namespace-aware processing.

The default execution context of the XPath is specified explicitly in 8.4 and 10.2.2.

For the following dialects, XML namespaces and QNames are not expected to be supported by default and can be silently ignored by the implementation.

These dialects are for informational purposes only and are not intended as Filter Dialects in actual SOAP messages. Because they are XPath compliant (albeit subsets), the Filter Dialect in the SOAP messages is still that of full XPath:

http://www.w3.org/TR/1999/REC-xpath-19991116

D.2 Level 1

Level 1 contains just the necessary XPath to identify nodes within an XML document or fragment and is targeted for use with Fragment-level access (7.7) of this specification.

EXAMPLE:

```
(1)  <path> ::= <root_selector> TOKEN_END_OF_INPUT;
(2)  <root_selector> ::= TOKEN_SLASH <element_sequence>;
(3)  <root_selector> ::= <attribute>;
(4)  <root_selector> ::= <relpath> <element_sequence>;
(5)  <root_selector> ::=  TOKEN_DOT

(6)  <relpath> ::= <>;
(7)  <relpath> ::= TOKEN_DOT TOKEN_SLASH;
(8)  <relpath> ::= TOKEN_DOT_DOT TOKEN_SLASH;

(9)  <element_sequence> ::= <element> <optional_filter_expression> <more>;

(10) <more> ::= TOKEN_SLASH <follower>;
(11) <more> ::= <>;

(12) <follower> ::= <attribute>;
(13) <follower> ::= <text_function>;
(14) <follower> ::= <element_sequence>;

(15) <optional_filter_expression> ::=
(16)   TOKEN_OPEN_BRACKET <filter_expression> TOKEN_CLOSE_BRACKET;

(17) <optional_filter_expression> ::= <>;

(18) <attribute> ::= TOKEN_AT_SYMBOL <name>;

(19) <element> ::= <name>;

(20) <text_function> ::=
(21)   TOKEN_TEXT TOKEN_OPEN_PAREN TOKEN_CLOSE_PAREN;

(22) <name> ::= TOKEN_XML_NAME;

(23) <filter_expression> ::= <array_location>;

(24) <array_location> ::= TOKEN_UNSIGNED_POSITIVE_INTEGER;
```

This dialect allows selecting any XML node based on its name or array position, or any attribute by its name. Optionally, the text() NodeTest can trail the entire expression to select only the raw value of the name, excluding the XML element name wrapper.

Terminals in the grammar are defined as shown in Table D-1.

Table D-1 – XPath Level 1 Terminals

`TOKEN_SLASH`	The character '/'
`TOKEN_DOT`	The character '.'
`TOKEN_DOT_DOT`	The characters '..'
`TOKEN_END_OF_INPUT`	End of input
`TOKEN_OPEN_BRACKET`	The character '['
`TOKEN_CLOSE_BRACKET`	The character ']'
`TOKEN_AT_SYMBOL`	The character '@'
`TOKEN_XML_NAME`	Equivalent to XML Schema type xs:token
`TOKEN_UNSIGNED_POSITIVE_INTEGER`	Values in the subrange 1..4294967295
`TOKEN_TEXT`	The characters 'text'
`TOKEN_OPEN_PAREN`	The character '('
`TOKEN_CLOSE_PAREN`	The character ')'

Using the following XML fragment, some examples are shown assuming that the element "a" is the context node (that is, represents the resource or event document).

EXAMPLE 1:

```
(1)  <Envelope>
(2)    <Body>
(3)      <a>
(4)        <b x="y"> 100 </b>
(5)        <c>
(6)          <d> 200 </d>
(7)        </c>
(8)        <c>
(9)          <d> 300 </d>
(10)         <d> 400 </d>
(11)       </c>
(12)     </a>
(13)   </Body>
(14) </Envelope>
```

EXAMPLE 2:

```
(1) / // Selects <a> and all its content
(2) /a  // Selects <a> and all its content
(3) . // Selects <a> and all its content
(4) ../a // Selects <a> and all its content
(5) b // Selects <b x="y"> 100 </b>
(6) c // Selects both <c> nodes, one after the other
(7) c[1] // Selects <c><d>200</d></c>
(8) c[2]/d[2] // Selects <d> 400 </d>
(9) c[2]/d[2]/text() // Selects 400
(10) b/text() // Selects  100
(11) b/@x // Selects x="y"
```

The only filtering expression capability is an array selection. XPath can return a node set. In 7.7 of this specification, the intent is to select a specific node, not a set of nodes, so if the situation occurs as illustrated on line (20) above, most implementations simply return a fault stating that it is unclear which `<c>` was meant and require the client to actually select one of the two available `<c>` elements using the array syntax. Also, text() cannot be suffixed to attribute selection.

A service that supports Fragment-level access as described in 7.7 of this specification is encouraged to support a subset of XPath at least as powerful as that described in Level 1.

Clearly, the service can expose full XPath 1.0 or any other subset that meets or exceeds the requirements defined here.

A service that supports the Level 1 XPath dialect must ensure that it observes matching of a single node. If more than one element of the same name is at the same level in the XML, the array notation must be used to distinguish them.

D.3 Level 2

Level 2 contains everything defined in Level 1, plus general-purpose filtering functionality with the standard set of relational operators and parenthesized sub-expressions (with AND, OR, NOT, and so on). This dialect is suitable for filtering using enumerations and subscription filters. This dialect is a strict superset of Level 1, with the <filter_expression> production being considerably extended to contain a useful subset of the XPath filtering syntax.

EXAMPLE 1:

```
(1)  <path> ::= <root_selector> TOKEN_END_OF_INPUT;
(2)  <root_selector> ::= TOKEN_SLASH <element_sequence>;
(3)  <root_selector> ::= <relpath> <element_sequence>;
(4)  <root_selector> ::= <attribute>;
(5)  <root_selector> ::= TOKEN_DOT;

(6)  <relpath> ::= <> ;
(7)  <relpath> ::= TOKEN_DOT TOKEN_SLASH;
(8)  <relpath> ::= TOKEN_DOT_DOT TOKEN_SLASH;

(9)  <element_sequence> ::= <element> <optional_filter_expression> <more>;

(10) <more> ::= TOKEN_SLASH <follower>;
(11) <more> ::= <>;

(12) <follower> ::= <attribute>;
(13) <follower> ::= <text_function>;
(14) <follower> ::= <element_sequence>;

(15) <optional_filter_expression> ::= TOKEN_OPEN_BRACKET <filter_expression>
       TOKEN_CLOSE_BRACKET;
(16) <optional_filter_expression> ::= <>;

(17) <attribute> ::= TOKEN_AT_SYMBOL <name>;

(18) <element> ::= <name>;

(19) <text_function> ::= TOKEN_TEXT TOKEN_OPEN_PAREN TOKEN_CLOSE_PAREN;

(20) <name> ::= TOKEN_XML_NAME;

(21) <filter_expression> ::= <array_location>;

(22) <array_location> ::= TOKEN_UNSIGNED_POSITIVE_INTEGER;

(23) // Next level, simple OR expression
(24) <or_expression> ::= <and_expression> <or_expression_rest>;
(25) <or_expression_rest> ::= TOKEN_OR <and_expression> <or_expression_rest>;
(26) <or_expression_rest> ::= <>;

(27) // Next highest level, AND expression
(28) <and_expression> ::= <rel_expression> <and_expression_rest>;
(29) <and_expression_rest> ::= TOKEN_AND <rel_expression>
   <and_expression_rest>;
(30) <and_expression_rest> ::= <>;
```

```
(31) // Next level of precedence >, <, >=, <=, =, !=
(32) <rel_expression> ::= <sub_expression> <rel_expression_rest>;
(33) <rel_expression_rest> ::= <name> <rel_op> <const>;
(34) <rel_expression_rest> ::= <>;

(35) // Identifier, literal, or identifier + param_list (function call)
(36) <sub_expression> ::= TOKEN_OPEN_PAREN <filter_expression>
   TOKEN_CLOSE_PAREN;
(37) <sub_expression> ::= TOKEN_NOT TOKEN_OPEN_PAREN <filter_expression>
      TOKEN_CLOSE_PAREN;

(38) // Relational operators
(39) <rel_op> ::= TOKEN_GT;     // >
(40) <rel_op> ::= TOKEN_LT;     // <
(41) <rel_op> ::= TOKEN_GE;     // >=
(42) <rel_op> ::= TOKEN_LE;     // <=
(43) <rel_op> ::= TOKEN_EQ;    // =
(44) <rel_op> ::= TOKEN_NE;    // !=

(45) <const> ::=  QUOTE TOKEN_STRING QUOTE;
```

Terminals in the grammar are defined as shown in Table D-2.

Table D-2 – XPath Level 2 Terminals

TOKEN_SLASH	The character '/'
TOKEN_DOT	The character '.'
TOKEN_DOT_DOT	The characters '..'
TOKEN_END_OF_INPUT	End of input
TOKEN_OPEN_BRACKET	The character '['
TOKEN_CLOSE_BRACKET	The character ']'
TOKEN_AT_SYMBOL	The character '@'
TOKEN_XML_NAME	Equivalent to XML Schema type xs:token
TOKEN_UNSIGNED_POSITIVE_INTEGER	Values in the subrange 1..4294967295
TOKEN_TEXT	The characters 'text'
TOKEN_OPEN_PAREN	The character '('
TOKEN_CLOSE_PAREN	The character ')'
TOKEN_AND	The characters 'and'
TOKEN_OR	The characters 'or'
TOKEN_NOT	The characters 'not'
TOKEN_STRING	Equivalent to XML Schema type xs:string
QUOTE	The character '"'

EXAMPLE 2: This dialect allows the same type of selection syntax as Level 1, but adds filtering, as in the following generic examples, given the Level 1 example document above:

```
(1)  b[@x="y"] // Select <b> if it has attribute x="y"
(2)  b[.="100"]  // Select <b> if it is 100
(3)  c[d="200"]  // Select <c> if <d> is 200
(4)  c/d[.="200"] // Select <d> if it is 200

(5)  b[.="100" and @x="z"]   // Select <b> if it is 100 and has @x="z"
(6)  c[d="200" or d="300"]   // Select all <c> with d=200 or d=300

(7)  c[2][not(.="400" or  @x="100")]
(8)  // Select second <c> provided that:
```

```
(9)  // its value is not 400 and it does not have an attribute x set to 100
(10) c/d[.="100" or  (@x="400" and .="500")]
(11) // Select <d> provided that:
(12) // its value is 100 or it has an attribute x set to 400 and its value is
   500
```

In essence, this dialect allows selecting any node based on a filter expression with the complete set of relational operators, logical operators, and parenthesized sub-expressions.

A service that supports XPath-based filtering dialects as described in this specification is encouraged to support a subset of XPath at least as powerful as that described in Level 2.

Clearly, the service can expose full XPath 1.0 or any other subset that meets or exceeds the requirements defined here.

In the actual operation, such as Enumerate or Subscribe, the XPath dialect is identified under the normal URI for full XPath:

http://www.w3.org/TR/1999/REC-xpath-19991116

ANNEX E
(normative)

Selector Filter Dialect

The Selector filter dialect is a simple filtering dialect that allows a filtered enumeration or subscription with no representation change.

Selectors are part of the default addressing model as defined in 5.1. This dialect is intended for implementations that support the default addressing model because it gives the ability to support filtering using a similar syntax while avoiding additional processing overhead of supporting more complex dialects.

This specification defines the following dialect filter URI for the Selector dialect:

http://schemas.dmtf.org/wbem/wsman/1/wsman/SelectorFilter

If a service uses the WS-Management default addressing model, it can support this filter dialect for enumeration and subscription operations.

The Selector filter dialect can be used to specify name value pairs in the selector syntax to filter the results from an Enumerate request or to identify the events of interest in a Subscribe request. The selectors act as a selection mechanism against the resource class space implied by the ResourceURI; however, there is no implication that the selector values are keys or even part of the returned resource.

The syntax for the filter in an Enumerate request is as follows:

```
<s:Header>
  <wsa:To> Service transport address </wsa:To>
  <wsman:ResourceURI> Resource URI </wsman:ResourceURI>
  ...
</s:Header>
<s:Body>
  <wsmen:Enumerate>
    <wsman:Filter
      Dialect="http://schemas.dmtf.org/wbem/wsman/1/wsman/SelectorFilter">
  <wsman:SelectorSet>
    <wsman:Selector Name=”selector-name”>
      selector-value
    </wsman:Selector> +
  </wsman:SelectorSet>
  </wsman:Filter>
  ...
</wsmen:Enumerate>
</s:Body>
```

Because the filter syntax does not include resource type information, the Resource URI specified in the addressing block is used for identifying the resource type. Each of the individual selectors within a SelectorSet are logically joined by AND for determining the result of the filter.

RE-1: If the Selector Filter dialect is supported, a service shall accept as selector names the local (NCName) part of the QNames of any of the top-level elements that represent the resource instance or event and may accept additional selector names. If the service supports filtering only on a subset of these QNames and the filter refers to an unsupported QName, the service shall respond with a wsme:CannotProcessFilter fault (or wsman:CannotProcessFilter for Subscribe), and should provide in the fault detail the list of selector names that are supported for filtering by the service.

RE-2: For each selector name specified in the filter, the result of the operation shall contain only instances for which that named element has the given value. Elements that are not referenced from the filter can have any value.

It is possible that some resource or event representations include elements of the same name, but from different XML Namespaces. In this case, the service can choose to match on any of the elements where the type matches the provided selector. Clients can be written to anticipate this, such that there might be additional post-processing necessary to identify the set of desired instances.

RE-3: If a resource or event representation includes two or more elements with QNames for which the local part is identical but whose namespace names are different, and all of the following conditions are present, the service shall not fault the request, and shall process the filter such that it matches exactly one of the elements for which filtering is supported, using an algorithm of the service's choosing:

- A selector filter contains a wsman:Selector element whose Name attribute matches the local part of each of these elements.
- At least one of the matching elements has a type and value space consistent with the provided selector type and value.
- The service supports filtering on at least one of the corresponding elements per RE-1.

RE-4: If a resource or event representation includes elements of an array type, and a filter contains a wsman:Selector element whose Name attribute matches the local part of the QName of these elements and the service supports filtering on the corresponding element per RE-1, the service shall process the filter such that the results include all representations for which at least one element of the array has a value equal to the value provided by the selector.

Processing of the SelectorSet element when used as a filter follows the same processing rules as when used in EPRs (as described in 5.4.2), with respect to duplicate selector names, type mismatches, unexpected selectors, size restrictions, and so on.

RE-5: If the filter expression contains a SelectorSet that is invalid with respect to the rules in 5.4.2, the service should fault with wsme:CannotProcessFilter (or wsman:CannotProcessFilter for Subscribe) containing the appropriate detail code.

ANNEX F
(informative)

Identify XML Schema

A normative copy of the XML schema of the Identify response message can be retrieved at the following address:

http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd

The following non-normative copy of the XML schema is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Notice
DSP8012
Document: WS-Management Identify XML Schema
Version: 1.0.1
Status: Final
Date: 02/27/2009
Author: DMTF WS-Management Work Group Email:wsman-chair@dmtf.org
Description: XML Schema for WS-Management Identify Operation.

Copyright © 2009 Distributed Management Task Force, Inc. (DMTF). All
rights reserved. DMTF is a not-for-profit association of industry members
dedicated to promoting enterprise and systems management and interoperability.
Members and non-members may reproduce DMTF specifications and documents,
provided that correct attribution is given. As DMTF specifications may be
revised from time to time, the particular version and release date should
always be noted. Implementation of certain elements of this standard or
proposed standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights"). DMTF makes no
representations to users of the standard as to the existence of such rights,
and is not responsible to recognize, disclose, or identify any or all such
third party patent right, owners or claimants, nor for any incomplete or
inaccurate identification or disclosure of such rights, owners or claimants.
DMTF shall have no liability to any party, in any manner or circumstance, under
any legal theory whatsoever, for failure to recognize, disclose, or identify
any such third party patent rights, or for such party’s reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures. DMTF shall have no liability to any party implementing such
standard, whether such implementation is foreseeable or not, nor to any patent
owner or claimant, and shall have no liability or responsibility for costs or
losses incurred if a standard is withdrawn or modified after publication, and
shall be indemnified and held harmless by any party implementing the standard
from any and all claims of infringement by a patent owner for such
implementations. For information about patents held by third-parties which have
notified the DMTF that, in their opinion, such patent may relate to or impact
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.

-->
<xs:schema
  targetNamespace="http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanident
ity.xsd"

xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
elementFormDefault="qualified" version="1.0.1">
<xs:complexType name="IdentifyType">
  <xs:sequence>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
      processContents="lax" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>
<xs:element name="Identify" type="wsmid:IdentifyType" />

<xs:simpleType name="restrictedProtocolVersionType">

  <xs:restriction base="xs:anyURI">
    <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity/NoAnonymo
usDisclosure" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ProtocolVersionType">
  <xs:union memberTypes="wsmid:restrictedProtocolVersionType xs:anyURI"
/>

</xs:simpleType>
<xs:element name="ProtocolVersion" type="wsmid:ProtocolVersionType" />
<xs:element name="ProductVendor" type="xs:string" />
<xs:element name="ProductVersion" type="xs:string" />
<xs:element name="InitiativeName" type="xs:string" />
<xs:element name="InitiativeVersion" type="wsmid:VERSION_VALUE"/>
<xs:element name="SecurityProfileName" type="xs:anyURI" />
<xs:complexType name="SecurityProfilesType">
  <xs:sequence>

    <xs:element ref="wsmid:SecurityProfileName" minOccurs="0"
      maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:element name="SecurityProfiles" type="wsmid:SecurityProfilesType" />
<xs:element name="AddressingVersionURI" type="xs:anyURI" />
<xs:element name="IntiativeSupport">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="wsmid:InitiativeName" minOccurs="0" maxOccurs="1"
/>

      <xs:element ref="wsmid:InitiativeVersion" minOccurs="0"
maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="IdentifyResponseType">
  <xs:sequence>
    <xs:element ref="wsmid:ProtocolVersion" maxOccurs="unbounded" />
    <xs:element ref="wsmid:ProductVendor" minOccurs="0" />
    <xs:element ref="wsmid:ProductVersion" minOccurs="0" />

```

```
      <xs:element ref="wsmid:IntiativeSupport" minOccurs="0" maxOccurs="unbounded"/>
      <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="wsmid:SecurityProfiles" minOccurs="0"
        maxOccurs="1" />
      <xs:element ref="wsmid:AddressingVersionURI" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <xs:element name="IdentifyResponse" type="wsmid:IdentifyResponseType" />

  <xs:simpleType name="VERSION_VALUE">

    <xs:annotation>
      <xs:documentation>Version values must be in form of M.N.U (Major, Minor, Update)</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="\d*.\d*.\d*" />
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

ANNEX G
(informative)

Resource Access Operations XML Schema and WSDL

A normative copy of the XML schemas (XML Schema 1, XML Schema 2) for the resource access operations can be retrieved at the following address:

http://schemas.dmtf.org/wbem/wsman/1/DSP8031_1.0.xsd

The following non-normative copy of the XML schema is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DMTF - Distributed Management Task Force, Inc. - http://www.dmtf.org

Document number: DSP8031
Date: 2010-02-19
Version: 1.0.0
Document status: DMTF Standard

Title: WS-Management Resource Access Operations XML Schema

Document type: Specification (W3C XML Schema)
Document language: E

Abstract: XML Schema for WS-Management Resource Access Operations.

Contact group: DMTF WS-Management Work Group, wsman-chair@dmtf.org

Copyright (C) 2008-2010 Distributed Management Task Force, Inc. (DMTF).
All rights reserved.  DMTF is a not-for-profit association of industry
members dedicated to promoting enterprise and systems management and
interoperability.  Members and non-members may reproduce DMTF
specifications and documents, provided that correct attribution is
given.  As DMTF specifications may be revised from time to time,
the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed
standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights").  DMTF makes
no representations to users of the standard as to the existence
of such rights, and is not responsible to recognize, disclose,
or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants.  DMTF shall have no
liability to any party, in any manner or circumstance, under any legal
theory whatsoever, for failure to recognize, disclose, or identify any
such third party patent rights, or for such party's reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures.  DMTF shall have no liability to any party implementing
such standard, whether such implementation is foreseeable or not, nor
to any patent owner or claimant, and shall have no liability or
responsibility for costs or losses incurred if a standard is withdrawn
or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims
of infringement by a patent owner for such implementations.  For
information about patents held by third-parties which have notified the
DMTF that, in their opinion, such patent may relate to or impact
```

```
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.

Change log:
1.0.0 - 2009-11-01 - Work in Progress release
1.0.0 - 2010-02-19 - DMTF Standard release

  -->
<xs:schema
  targetNamespace="http://schemas.xmlsoap.org/ws/2004/09/transfer"
  xmlns:tns="http://schemas.xmlsoap.org/ws/2004/09/transfer"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa04="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsa10="http://www.w3.org/2005/08/addressing"
  elementFormDefault="qualified"
  blockDefault="#all" >

  <xs:import
    namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    schemaLocation="http://schemas.dmtf.org/wbem/wsman/1/DSP8034_1.0.xsd"
/>
  <xs:import
    namespace="http://www.w3.org/2005/08/addressing"
    schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd" />

<!--
  The type of the AnyEPRType is effectively
  the union of wsa04:EndpointReferenceType and
  wsa10:EndpointReferenceType. Unfortunately, xs:union only
  works for simple types. As a result, we have to define
  the element in an unvalidated way to accommodate either
  addressing type.
  -->

   <xs:complexType name="AnyEPRType">
     <xs:sequence>
       <xs:any minOccurs='1' maxOccurs='unbounded' processContents='skip'
         namespace='##other' />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="ResourceCreated" type="tns:AnyEPRType"/>

  <!-- The following GED is defined for convenience. This GED
       may be used in cases where a resource-specific GED is
       not available. -->
  <xs:element name="TransferElement">
    <xs:complexType>
      <xs:sequence>
        <xs:any minOccurs='1' maxOccurs='unbounded'
          processContents='skip' namespace='##other'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

A normative copy of the WSDL description for the resource access operations can be retrieved from the following address:

http://schemas.dmtf.org/wbem/wsman/1/DSP8035_1.0.wsdl

The following non-normative copy of the WSDL description is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DMTF - Distributed Management Task Force, Inc. - http://www.dmtf.org

Document number: DSP8035
Date: 2010-02-19
Version: 1.0.0
Document status: DMTF Standard

Title: WS-Management Resource Access Operations WSDL

Document type: Specification (W3C WSDL Document)
Document language: E

Abstract: WSDL for WS-Management Resource Access Operations.

Contact group: DMTF WS-Management Work Group, wsman-chair@dmtf.org

Copyright (C) 2008-2010 Distributed Management Task Force, Inc. (DMTF).
All rights reserved.  DMTF is a not-for-profit association of industry
members dedicated to promoting enterprise and systems management and
interoperability.  Members and non-members may reproduce DMTF
specifications and documents, provided that correct attribution is
given.  As DMTF specifications may be revised from time to time,
the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed
standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights").  DMTF makes
no representations to users of the standard as to the existence
of such rights, and is not responsible to recognize, disclose,
or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants.  DMTF shall have no
liability to any party, in any manner or circumstance, under any legal
theory whatsoever, for failure to recognize, disclose, or identify any
such third party patent rights, or for such party's reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures.  DMTF shall have no liability to any party implementing
such standard, whether such implementation is foreseeable or not, nor
to any patent owner or claimant, and shall have no liability or
responsibility for costs or losses incurred if a standard is withdrawn
or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims
of infringement by a patent owner for such implementations.  For
information about patents held by third-parties which have notified the
DMTF that, in their opinion, such patent may relate to or impact
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.

Change log:
1.0.0 - 2009-11-01 - Work in Progress release
1.0.0 - 2010-02-19 - DMTF Standard release

  -->
<wsdl:definitions
    targetNamespace="http://schemas.xmlsoap.org/ws/2004/09/transfer"
  xmlns:tns="http://schemas.xmlsoap.org/ws/2004/09/transfer"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

<wsdl:types>
  <xs:schema>
    <xs:import
      namespace="http://schemas.xmlsoap.org/ws/2004/09/transfer"
      schemaLocation="http://schemas.dmtf.org/wbem/wsman/1/DSP8031_1.0.xsd"
      />
  </xs:schema>
</wsdl:types>

<!--
In some of the messages defined below a "resource-specific-GED"
is expected to be inserted before the WSDL is processed by any tooling.
Thus the WSDL as presented is not usable until after this substitution
is done.
-->

<wsdl:message name="EmptyMessage"/>
<wsdl:message name="CreateRequestMessage">
  <wsdl:part name="Body" element="resource-specific-GED"/>
</wsdl:message>
<wsdl:message name="CreateResponseMessage">
  <wsdl:part name="Body" element="tns:ResourceCreated"/>
</wsdl:message>
<wsdl:message name="GetResponseMessage">
  <wsdl:part name="Body" element="resource-specific-GED"/>
</wsdl:message>
<wsdl:message name="PutRequestMessage">
  <wsdl:part name="Body" element="resource-specific-GED"/>
</wsdl:message>
<wsdl:message name="PutResponseMessage">
  <!-- Note this 'part' may be omitted -->
  <wsdl:part name="Body" element="resource-specific-GED"/>
</wsdl:message>

<wsdl:portType name="Resource">
  <wsdl:documentation>
    This port type defines a resource that may be read,
    written, and deleted.
  </wsdl:documentation>
  <wsdl:operation name="Get">
    <wsdl:input
      message="tns:EmptyMessage"
      wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Get"
      wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Get" />
    <wsdl:output
      message="tns:GetResponseMessage"
      wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse"
      wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse" />
  </wsdl:operation>
  <wsdl:operation name="Put">
    <wsdl:input
      message="tns:PutRequestMessage"
      wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Put"
```

```
 (116)         wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Put"
/>
 (117)       <wsdl:output
 (118)         message="tns:PutResponseMessage"
 (119)
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse"
 (120)
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse"/>
 (121)     </wsdl:operation>
 (122)     <wsdl:operation name="Delete">
 (123)       <wsdl:input
 (124)         message="tns:EmptyMessage"
 (125)
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete"
 (126)
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete" />
 (127)       <wsdl:output
 (128)         message="tns:EmptyMessage"
 (129)
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse"
 (130)
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse"
 (131)       />
 (132)     </wsdl:operation>
 (133)   </wsdl:portType>
 (134)
 (135)   <wsdl:portType name="ResourceFactory">
 (136)     <wsdl:documentation>
 (137)       This port type defines a Web service that can create new
 (138)       resources.
 (139)     </wsdl:documentation>
 (140)     <wsdl:operation name="Create">
 (141)       <wsdl:input
 (142)         message="tns:CreateRequestMessage"
 (143)
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Create"
 (144)
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/Create" />
 (145)       <wsdl:output
 (146)         message="tns:CreateResponseMessage"
 (147)
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse"
 (148)
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse"
 (149)       />
 (150)     </wsdl:operation>
 (151)   </wsdl:portType>
 (152) </wsdl:definitions>
```

ANNEX H
(informative)

Enumeration Operations XML Schema and WSDL

A normative copy of the XML schemas for the enumeration operations can be retrieved at the following address:

http://schemas.dmtf.org/wbem/wsman/1/DSP8033_1.0.xsd

The following non-normative copy of the XML schema is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DMTF - Distributed Management Task Force, Inc. - http://www.dmtf.org

Document number: DSP8033
Date: 2010-02-19
Version: 1.0.0
Document status: DMTF Standard

Title: WS-Management Enumeration Operations XML Schema

Document type: Specification (W3C XML Schema)
Document language: E

Abstract: XML Schema for WS-Management Enumeration Operations.

Contact group: DMTF WS-Management Work Group, wsman-chair@dmtf.org

Copyright (C) 2008–2010 Distributed Management Task Force, Inc. (DMTF).
All rights reserved.  DMTF is a not-for-profit association of industry
members dedicated to promoting enterprise and systems management and
interoperability.  Members and non-members may reproduce DMTF
specifications and documents, provided that correct attribution is
given.  As DMTF specifications may be revised from time to time,
the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed
standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights").  DMTF makes
no representations to users of the standard as to the existence of
such rights, and is not responsible to recognize, disclose,
or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants.  DMTF shall have no
liability to any party, in any manner or circumstance, under any legal
theory whatsoever, for failure to recognize, disclose, or identify any
such third party patent rights, or for such party's reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures.  DMTF shall have no liability to any party implementing
such standard, whether such implementation is foreseeable or not, nor
to any patent owner or claimant, and shall have no liability or
responsibility for costs or losses incurred if a standard is withdrawn
or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims
of infringement by a patent owner for such implementations.  For
information about patents held by third-parties which have notified the
DMTF that, in their opinion, such patent may relate to or impact
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.
```

```

Change log:
1.0.0 - 2009-11-01 - Work in Progress release
1.0.0 – 2010-02-19 – DMTF Standard release

  -->
<xs:schema
    targetNamespace="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:tns="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    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.dmtf.org/wbem/wsman/1/DSP8034_1.0.xsd"
/>
  <xs:import
     namespace="http://www.w3.org/2005/08/addressing"
     schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd" />

  <!-- Types and global elements -->
  <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" />
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <xs:simpleType name="PositiveDurationType">
    <xs:restriction base="xs:duration">
      <xs:minExclusive value="P0Y0M0DT0H0M0S" />
    </xs:restriction>
  </xs:simpleType>

  <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="EnumerationContextType">
    <xs:complexContent mixed="true">
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

```

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

<!--
  The type of the AnyEPRType is effectively
  the union of wsa04:EndpointReferenceType and
  wsa10:EndpointReferenceType. Unfortunately, xs:union only
  works for simple types. As a result, we have to define
  the element in an unvalidated way to accommodate either
  addressing type.
  -->

<xs:complexType name="AnyEPRType">
  <xs:sequence>
    <xs:any minOccurs='1' maxOccurs='unbounded' processContents='skip'
            namespace='##other' />
  </xs:sequence>
</xs:complexType>

<!-- Enumerate request -->
<xs:element name="Enumerate">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EndTo" type="tns:AnyEPRType"
                  minOccurs="0" />
      <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>

<!-- Used for a fault response -->
<xs:element name="SupportedDialect" type="xs:anyURI" />

<!-- Enumerate response -->
<xs:element name="EnumerateResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Expires" type="tns:ExpirationType"
                  minOccurs="0" />
      <xs:element name="EnumerationContext"
                  type="tns:EnumerationContextType" />
      <xs:any namespace="##other" processContents="lax"
              minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
```

```
      <xs:anyAttribute namespace="##other" processContents="lax" />
    </xs:complexType>
  </xs:element>

  <!-- Pull request -->
  <xs:element name="Pull">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="EnumerationContext"
                    type="tns:EnumerationContextType" />
        <xs:element name="MaxTime" type="tns:PositiveDurationType"
                    minOccurs="0" />
        <xs:element name="MaxElements" type="xs:positiveInteger"
                    minOccurs="0" />
        <xs:element name="MaxCharacters" type="xs:positiveInteger"
                    minOccurs="0" />
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax" />
    </xs:complexType>
  </xs:element>

  <!-- Pull response -->
  <xs:element name="PullResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="EnumerationContext"
                    type="tns:EnumerationContextType"
                    minOccurs="0" />
        <xs:element name="Items" type="tns:ItemListType"
                    minOccurs="0" />
        <xs:element name="EndOfSequence" minOccurs="0" />
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax" />
    </xs:complexType>
  </xs:element>

  <!-- Renew request -->
  <xs:element name="Renew">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="EnumerationContext"
                    type="tns:EnumerationContextType" />
        <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:element name="EnumerationContext"
                    type="tns:EnumerationContextType"
                    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:element name="EnumerationContext"
                    type="tns:EnumerationContextType" />
        <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>

  <!-- Release request -->
  <xs:element name="Release">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="EnumerationContext"
                    type="tns:EnumerationContextType" />
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax" />
    </xs:complexType>
  </xs:element>

  <!-- Release response has an empty body -->

  <!-- EnumerationEnd message -->
  <xs:element name="EnumerationEnd">
    <xs:complexType>
    <xs:sequence>
      <xs:element name="EnumerationContext"
                  type="tns:EnumerationContextType" />
      <xs:element name="Code" type="tns:OpenEnumerationEndCodeType" />
      <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="EnumerationEndCodeType">
    <xs:restriction base="xs:anyURI">
    <xs:enumeration
value="http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceShuttingDown" />
    <xs:enumeration
value="http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceCancelling" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="OpenEnumerationEndCodeType">
    <xs:union memberTypes="tns:EnumerationEndCodeType xs:anyURI" />
  </xs:simpleType>
</xs:schema>
```

A normative copy of the WSDL description for enumeration operations can be retrieved from the following address:

http://schemas.dmtf.org/wbem/wsman/1/DSP8037_1.0.wsdl

The following non-normative copy of the WSDL description is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DMTF - Distributed Management Task Force, Inc. - http://www.dmtf.org

Document number: DSP8037
Date: 2010-02-19
Version: 1.0.0
Document status: DMTF Standard

Title: WS-Management Enumeration Operations WSDL

Document type: Specification (W3C WSDL Document)
Document language: E

Abstract: WSDL for WS-Management Enumeration Operations.

Contact group: DMTF WS-Management Work Group, wsman-chair@dmtf.org

Copyright (C) 2008-2010 Distributed Management Task Force, Inc. (DMTF).
All rights reserved.  DMTF is a not-for-profit association of industry
members dedicated to promoting enterprise and systems management and
interoperability.  Members and non-members may reproduce DMTF
specifications and documents, provided that correct attribution is
given.  As DMTF specifications may be revised from time to time,
the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed
standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights").  DMTF makes
no representations to users of the standard as to the existence of
such rights, and is not responsible to recognize, disclose,
or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants.  DMTF shall have no
liability to any party, in any manner or circumstance, under any legal
theory whatsoever, for failure to recognize, disclose, or identify any
such third party patent rights, or for such party's reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures.  DMTF shall have no liability to any party implementing
such standard, whether such implementation is foreseeable or not, nor
to any patent owner or claimant, and shall have no liability or
responsibility for costs or losses incurred if a standard is withdrawn
or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims
of infringement by a patent owner for such implementations.  For
```

```
information about patents held by third-parties which have notified the
DMTF that, in their opinion, such patent may relate to or impact
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.

Change log:
1.0.0 - 2009-11-01 - Work in Progress release
1.0.0 – 2010-02-19 – DMTF Standard release

  -->
<wsdl:definitions
    targetNamespace="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsmen="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <wsdl:types>
    <xs:schema>
      <xs:import
        namespace="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
     schemaLocation="http://schemas.dmtf.org/wbem/wsman/1/DSP8033_1.0.xsd"
        />
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="EnumerateMessage">
    <wsdl:part name="Body" element="wsmen:Enumerate" />
  </wsdl:message>
  <wsdl:message name="EnumerateResponseMessage">
    <wsdl:part name="Body" element="wsmen:EnumerateResponse" />
  </wsdl:message>
  <wsdl:message name="PullMessage">
    <wsdl:part name="Body" element="wsmen:Pull" />
  </wsdl:message>
  <wsdl:message name="PullResponseMessage">
    <wsdl:part name="Body" element="wsmen:PullResponse" />
  </wsdl:message>
  <wsdl:message name="RenewMessage" >
    <wsdl:part name="Body" element="wsmen:Renew" />
  </wsdl:message>
  <wsdl:message name="RenewResponseMessage" >
    <wsdl:part name="Body" element="wsmen:RenewResponse" />
  </wsdl:message>
  <wsdl:message name="GetStatusMessage" >
    <wsdl:part name="Body" element="wsmen:GetStatus" />
  </wsdl:message>
  <wsdl:message name="GetStatusResponseMessage" >
    <wsdl:part name="Body" element="wsmen:GetStatusResponse" />
  </wsdl:message>
  <wsdl:message name="ReleaseMessage">
    <wsdl:part name="Body" element="wsmen:Release" />
  </wsdl:message>
  <wsdl:message name="ReleaseResponseMessage" />
  <wsdl:message name="EnumerationEndMessage" >
    <wsdl:part name="Body" element="wsmen:EnumerationEnd" />
  </wsdl:message>

  <wsdl:portType name="DataSource">
    <wsdl:operation name="EnumerateOp">
      <wsdl:input
        message="wsmen:EnumerateMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate"
        />
      <wsdl:output
        message="wsmen:EnumerateResponseMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse"
       />
    </wsdl:operation>
    <wsdl:operation name="PullOp">
      <wsdl:input
        message="wsmen:PullMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull"
        />
      <wsdl:output
        message="wsmen:PullResponseMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse"
        />
    </wsdl:operation>
    <wsdl:operation name="RenewOp" >
      <wsdl:input
        message="wsmen:RenewMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew"
        />
      <wsdl:output
        message="wsmen:RenewResponseMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse"
        />
    </wsdl:operation>
    <wsdl:operation name="GetStatusOp" >
      <wsdl:input
        message="wsmen:GetStatusMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus"
        />
      <wsdl:output
        message="wsmen:GetStatusResponseMessage"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse"
        />
    </wsdl:operation>
    <wsdl:operation name="ReleaseOp">
      <wsdl:input
        message="wsmen:ReleaseMessage"
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release"
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release"
        />
      <wsdl:output
        message="wsmen:ReleaseResponseMessage"
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse"
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse"
        />
    </wsdl:operation>
  </wsdl:portType>

  <!-- The following portType shall be supported by the endpoint to which
       The EnumerationEnd message is sent -->
  <wsdl:portType name="EnumEndEndpoint">
    <wsdl:operation name="EnumerationEndOp" >
      <wsdl:input
        message="wsmen:EnumerationEndMessage"
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd"
wsam:Action="http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd"
        />
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

ANNEX I
(informative)

Notification OperationsXML Schema and WSDL

A normative copy of the XML schemas for the notification operations can be retrieved at the following address:

http://schemas.dmtf.org/wbem/wsman/1/DSP8032_1.0.xsd

The following non-normative copy of the XML schema is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DMTF - Distributed Management Task Force, Inc. - http://www.dmtf.org

Document number: DSP8032
Date: 2010-02-19
Version: 1.0.0
Document status: DMTF Standard

Title: WS-Management Notification Operations XML Schema

Document type: Specification (W3C XML Schema)
Document language: E

Abstract: XML Schema for WS-Management Notification Operations.

Contact group: DMTF WS-Management Work Group, wsman-chair@dmtf.org

Copyright (C) 2008-2009 Distributed Management Task Force, Inc. (DMTF).
All rights reserved.  DMTF is a not-for-profit association of industry
members dedicated to promoting enterprise and systems management and
interoperability.  Members and non-members may reproduce DMTF
specifications and documents, provided that correct attribution is
given.  As DMTF specifications may be revised from time to time,
the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed
standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights").  DMTF makes
no representations to users of the standard as to the existence of
such rights, and is not responsible to recognize, disclose,
or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants.  DMTF shall have no
liability to any party, in any manner or circumstance, under any legal
theory whatsoever, for failure to recognize, disclose, or identify any
such third party patent rights, or for such party's reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures.  DMTF shall have no liability to any party implementing
such standard, whether such implementation is foreseeable or not, nor
to any patent owner or claimant, and shall have no liability or
responsibility for costs or losses incurred if a standard is withdrawn
or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims
of infringement by a patent owner for such implementations.  For
information about patents held by third-parties which have notified the
DMTF that, in their opinion, such patent may relate to or impact
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.
```

```

Change log:
1.0.0 - 2009-11-01 - Work in Progress release
1.0.0 - 2010-02-19 - DMTF Standard release

  -->
<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.dmtf.org/wbem/wsman/1/DSP8034_1.0.xsd" />
  <xs:import
    namespace="http://www.w3.org/2005/08/addressing"
    schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd" />

  <!-- 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>
```

```

<!--
  The type of the AnyEPRType is effectively
  the union of wsa04:EndpointReferenceType and
  wsa10:EndpointReferenceType. Unfortunately, xs:union only
  works for simple types. As a result, we have to define
  the element in an unvalidated way to accommodate either
  addressing type.
  -->

<xs:complexType name="AnyEPRType">
  <xs:sequence>
    <xs:any minOccurs='1' maxOccurs='unbounded' processContents='skip'
            namespace='##other' />
  </xs:sequence>
</xs:complexType>

<xs:element name="NotifyTo" type="tns:AnyEPRType" />

<!-- Subscribe request -->
<xs:element name="Subscribe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EndTo" type="tns:AnyEPRType"
                  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="tns:AnyEPRType" />
      <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>

<!-- SubscriptionEnd message -->
<xs:element name="SubscriptionEnd">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SubscriptionManager"
                  type="tns:AnyEPRType" />
```

```
        <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>
```

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

http://schemas.dmtf.org/wbem/wsman/1/DSP8036_1.0.wsdl

The following non-normative copy of the WSDL description is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DMTF - Distributed Management Task Force, Inc. - http://www.dmtf.org

Document number: DSP8036
Date: 2010-02-19
Version: 1.0.0
Document status: DMTF Standard

Title: WS-Management Notification Operations WSDL

Document type: Specification (W3C WSDL Document)
Document language: E

Abstract: WSDL for WS-Management Notification Operations.

Contact group: DMTF WS-Management Work Group, wsman-chair@dmtf.org

Copyright (C) 2008-2010 Distributed Management Task Force, Inc. (DMTF).
All rights reserved.  DMTF is a not-for-profit association of industry
members dedicated to promoting enterprise and systems management and
interoperability.  Members and non-members may reproduce DMTF
```

```
specifications and documents, provided that correct attribution is
given.  As DMTF specifications may be revised from time to time,
the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed
standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights").  DMTF makes
no representations to users of the standard as to the existence of
such rights, and is not responsible to recognize, disclose,
or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants.  DMTF shall have no
liability to any party, in any manner or circumstance, under any legal
theory whatsoever, for failure to recognize, disclose, or identify any
such third party patent rights, or for such party's reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures.  DMTF shall have no liability to any party implementing
such standard, whether such implementation is foreseeable or not, nor
to any patent owner or claimant, and shall have no liability or
responsibility for costs or losses incurred if a standard is withdrawn
or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims
of infringement by a patent owner for such implementations.  For
information about patents held by third-parties which have notified the
DMTF that, in their opinion, such patent may relate to or impact
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.

Change log:
1.0.0 - 2009-11-01 - Work in Progress release
1.0.0.- 2010-02-19 - DMTF Standard release

  -->
<wsdl:definitions
  targetNamespace="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:wsme="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>
       <xs:import
         namespace="http://schemas.xmlsoap.org/ws/2004/08/eventing"

schemaLocation="http://schemas.dmtf.org/wbem/wsman/1/DSP8032_1.0.xsd" />
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="SubscribeMsg" >
    <wsdl:part name="body" element="wsme:Subscribe" />
  </wsdl:message>
  <wsdl:message name="SubscribeResponseMsg" >
    <wsdl:part name="body" element="wsme:SubscribeResponse" />
  </wsdl:message>

  <wsdl:message name="RenewMsg" >
    <wsdl:part name="body" element="wsme:Renew" />
  </wsdl:message>
  <wsdl:message name="RenewResponseMsg" >
    <wsdl:part name="body" element="wsme:RenewResponse" />
  </wsdl:message>

```

```
<wsdl:message name="GetStatusMsg" >
  <wsdl:part name="body" element="wsme:GetStatus" />
</wsdl:message>
<wsdl:message name="GetStatusResponseMsg" >
  <wsdl:part name="body" element="wsme:GetStatusResponse" />
</wsdl:message>

<wsdl:message name="UnsubscribeMsg" >
  <wsdl:part name="body" element="wsme:Unsubscribe" />
</wsdl:message>
<wsdl:message name="UnsubscribeResponseMsg" />

<wsdl:message name="SubscriptionEnd" >
  <wsdl:part name="body" element="wsme:SubscriptionEnd" />
</wsdl:message>

<wsdl:portType name="EventSource" >
  <wsdl:operation name="SubscribeOp" >
    <wsdl:input
      message="wsme:SubscribeMsg"
wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe"
wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe"/>
    <wsdl:output
      message="wsme:SubscribeResponseMsg"
wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse"
wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse"/
>
  </wsdl:operation>
</wsdl:portType>

<!-- The following portType shall be supported by the endpoint to which
     the SubscriptionEnd message is sent.  -->
<wsdl:portType name="EndToEndpoint">
  <wsdl:operation name="SubscriptionEnd" >
    <wsdl:input
      message="wsme:SubscriptionEnd"
wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd"
wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd"/>
  </wsdl:operation>
</wsdl:portType>

<!-- The following portType shall be supported by the endpoint to which
     Notifications are sent.  This portType also serves as a
     mechanism by which Subscribers can know the Notifications that
     will sent by an Event Source. -->
<wsdl:portType name="EventSink">
  <!-- place the Notification messages (operations) here. For example:
  <wsdl:operation name="WeatherReport">
  <wsdl:input message="wr:ThunderStormMessage"
    wsa:Action="urn:weatherReport:ThunderStorm"
    wsam:Action="urn:weatherReport:ThunderStorm" />
</wsdl:operation>
-->
</wsdl:portType>

<wsdl:portType name="SubscriptionManager" >
  <wsdl:operation name="RenewOp" >
```

```
      <wsdl:input
        message="wsme:RenewMsg"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew"/>
      <wsdl:output
        message="wsme:RenewResponseMsg"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse"/>
    </wsdl:operation>
    <wsdl:operation name="GetStatusOp" >
      <wsdl:input
        message="wsme:GetStatusMsg"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus"/>
      <wsdl:output
        message="wsme:GetStatusResponseMsg"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse"/>
    </wsdl:operation>
    <wsdl:operation name="UnsubscribeOp" >
      <wsdl:input
        message="wsme:UnsubscribeMsg"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe"/>
      <wsdl:output
        message="wsme:UnsubscribeResponseMsg"
        wsa:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse"
        wsam:Action="http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

ANNEX J
(informative)

Addressing XML Schema

A normative copy of the XML schemas for the addressing features can be retrieved at the following address:

http://schemas.dmtf.org/wbem/wsman/1/DSP8034_1.0.xsd

The following non-normative copy of the XML schema is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
DMTF - Distributed Management Task Force, Inc. - http://www.dmtf.org

Document number: DSP8034
Date: 2010-02-19
Version: 1.0.0
Document status: DMTF Standard

Title: WS-Management Addressing XML Schema

Document type: Specification (W3C XML Schema)
Document language: E

Abstract: XML Schema for WS-Management Addressing.

Contact group: DMTF WS-Management Work Group, wsman-chair@dmtf.org

Copyright (C) 2008–2010 Distributed Management Task Force, Inc. (DMTF).
All rights reserved.  DMTF is a not-for-profit association of industry
members dedicated to promoting enterprise and systems management and
interoperability.  Members and non-members may reproduce DMTF
specifications and documents, provided that correct attribution is
given.  As DMTF specifications may be revised from time to time,
the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed
standard may be subject to third party patent rights, including
provisional patent rights (herein "patent rights").  DMTF makes
no representations to users of the standard as to the existence of
such rights, and is not responsible to recognize, disclose,
or identify any or all such third party patent right, owners or
claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants.  DMTF shall have no
liability to any party, in any manner or circumstance, under any legal
theory whatsoever, for failure to recognize, disclose, or identify any
such third party patent rights, or for such party's reliance on the
standard or incorporation thereof in its product, protocols or testing
procedures.  DMTF shall have no liability to any party implementing
such standard, whether such implementation is foreseeable or not, nor
to any patent owner or claimant, and shall have no liability or
responsibility for costs or losses incurred if a standard is withdrawn
or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims
of infringement by a patent owner for such implementations.  For
information about patents held by third-parties which have notified the
DMTF that, in their opinion, such patent may relate to or impact
implementations of DMTF standards, visit
http://www.dmtf.org/about/policies/disclosures.php.
```

```

Change log:
1.0.0 - 2009-11-01 - Work in Progress release
1.0.0 – 2010-02-19 – DMTF Standard release
  -->
<xs:schema
  targetNamespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  elementFormDefault="qualified" blockDefault="#all">

  <!-- ////////////////// Addressing ////////////////// -->
  <!-- Endpoint reference -->
  <xs:element name="EndpointReference" type="wsa:EndpointReferenceType"/>
  <xs:complexType name="EndpointReferenceType">
    <xs:sequence>
      <xs:element name="Address" type="wsa:AttributedURI"/>
      <xs:element name="ReferenceProperties"
        type="wsa:ReferencePropertiesType" minOccurs="0"/>
      <xs:element name="ReferenceParameters"
        type="wsa:ReferenceParametersType" minOccurs="0"/>
      <xs:element name="PortType" type="wsa:AttributedQName"
minOccurs="0"/>
      <xs:element name="ServiceName" type="wsa:ServiceNameType"
minOccurs="0"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
           If "Policy" elements from namespace
           "http://schemas.xmlsoap.org/ws/2002/12/policy#policy" are used,
           they must appear first (before any extensibility elements).
          </xs:documentation>
        </xs:annotation>
      </xs:any>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="ReferencePropertiesType">
    <xs:sequence>
      <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ReferenceParametersType">
    <xs:sequence>
      <xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ServiceNameType">
    <xs:simpleContent>
      <xs:extension base="xs:QName">
        <xs:attribute name="PortName" type="xs:NCName"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <!-- Message information header blocks -->
  <xs:element name="MessageID" type="wsa:AttributedURI"/>
  <xs:element name="RelatesTo" type="wsa:Relationship"/>
  <xs:element name="To" type="wsa:AttributedURI"/>
  <xs:element name="Action" type="wsa:AttributedURI"/>
  <xs:element name="From" type="wsa:EndpointReferenceType"/>
  <xs:element name="ReplyTo" type="wsa:EndpointReferenceType"/>
  <xs:element name="FaultTo" type="wsa:EndpointReferenceType"/>
  <xs:complexType name="Relationship">
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
        <xs:attribute name="RelationshipType" type="xs:QName" use="optional"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:simpleType name="RelationshipTypeValues">
    <xs:restriction base="xs:QName">
      <xs:enumeration value="wsa:Reply"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="ReplyAfter" type="wsa:ReplyAfterType"/>
  <xs:complexType name="ReplyAfterType">
    <xs:simpleContent>
      <xs:extension base="xs:nonNegativeInteger">
        <xs:anyAttribute namespace="##other"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:element name="RetryAfter" type="wsa:RetryAfterType"/>
  <xs:complexType name="RetryAfterType">
    <xs:simpleContent>
      <xs:extension base="xs:nonNegativeInteger">
        <xs:anyAttribute namespace="##other"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:simpleType name="FaultSubcodeValues">
    <xs:restriction base="xs:QName">
      <xs:enumeration value="wsa:InvalidMessageInformationHeader"/>
      <xs:enumeration value="wsa:MessageInformationHeaderRequired"/>
      <xs:enumeration value="wsa:DestinationUnreachable"/>
      <xs:enumeration value="wsa:ActionNotSupported"/>
      <xs:enumeration value="wsa:EndpointUnavailable"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:attribute name="Action" type="xs:anyURI"/>
  <!-- Common declarations and definitions -->
  <xs:complexType name="AttributedQName">
    <xs:simpleContent>
      <xs:extension base="xs:QName">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="AttributedURI">
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

ANNEX K
(informative)

WS-Management XML Schema

A normative copy of the XML schemas for WS-Management can be retrieved at the following address:

http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd

The following non-normative copy of the XML schema is provided for convenience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Notice
DSP8015
Document: WS-Management protocol XML Schema
Version: 1.0.0
Status: Final
Date: 01/20/2008
Author: Bryan Murray, et al.
Description: XML Schema for WS-Management protocol

Copyright © 2008 Distributed Management Task Force, Inc. (DMTF). All rights
reserved. DMTF is a not-for-profit association of industry members dedicated to
promoting enterprise and systems management and interoperability. Members and
non-members may reproduce DMTF specifications and documents, provided that
correct attribution is given. As DMTF specifications may be revised from time
to time, the particular version and release date should always be noted.
Implementation of certain elements of this standard or proposed standard may be
subject to third party patent rights, including provisional patent rights
(herein "patent rights"). DMTF makes no representations to users of the
standard as to the existence of such rights, and is not responsible to
recognize, disclose, or identify any or all such third party patent right,
owners or claimants, nor for any incomplete or inaccurate identification or
disclosure of such rights, owners or claimants. DMTF shall have no liability to
any party, in any manner or circumstance, under any legal theory whatsoever,
for failure to recognize, disclose, or identify any such third party patent
rights, or for such party's reliance on the standard or incorporation thereof
in its product, protocols or testing procedures. DMTF shall have no liability
to any party implementing such standard, whether such implementation is
foreseeable or not, nor to any patent owner or claimant, and shall have no
liability or responsibility for costs or losses incurred if a standard is
withdrawn or modified after publication, and shall be indemnified and held
harmless by any party implementing the standard from any and all claims of
infringement by a patent owner for such implementations. For information about
patents held by third-parties which have notified the DMTF that, in their
opinion, such patent may relate to or impact implementations of DMTF standards,
visit http://www.dmtf.org/about/policies/disclosures.php.

Change Requests:
  None
-->
<xs:schema targetNamespace="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" version="1.0.0e">

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

  <xs:complexType name="attributableURI">
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="ResourceURI" type="wsman:attributableURI"/>

  <xs:complexType name="SelectorType">
    <xs:annotation>
      <xs:documentation>
        Instances of this type can be only simple types or EPRs, not arbitrary mixed data.
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent mixed="true">
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:element ref="wsa:EndpointReference" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:NCName" use="required"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="Selector" type="wsman:SelectorType"/>

  <xs:complexType name="SelectorSetType">
    <xs:sequence>
      <xs:element ref="wsman:Selector" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:element name="SelectorSet" type="wsman:SelectorSetType">
    <xs:unique name="oneSelectorPerName">
      <xs:selector xpath="./Selector"/>
      <xs:field xpath="@Name"/>
    </xs:unique>
  </xs:element>

  <xs:complexType name="attributableDuration">
    <xs:simpleContent>
      <xs:extension base="xs:duration">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="OperationTimeout" type="wsman:attributableDuration"/>

  <xs:complexType name="attributablePositiveInteger">
```

```
      <xs:simpleContent>
        <xs:extension base="xs:positiveInteger">
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>

    <xs:simpleType name="PolicyType">
      <xs:restriction base="xs:token">
        <xs:enumeration value="CancelSubscription"/>
        <xs:enumeration value="Skip"/>
        <xs:enumeration value="Notify"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="MaxEnvelopeSizeType">
      <xs:simpleContent>
        <xs:extension base="wsman:attributablePositiveInteger">
          <xs:attribute name="Policy" type="wsman:PolicyType" default="Notify"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <xs:element name="MaxEnvelopeSize" type="wsman:MaxEnvelopeSizeType"/>

    <xs:element name="Locale">
      <xs:complexType>
        <xs:attribute ref="xml:lang" use="required"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:complexType>
    </xs:element>

    <xs:complexType name="OptionType">
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="Name" type="xs:NCName" use="required"/>
          <xs:attribute name="MustComply" type="xs:boolean" default="false"/>
          <xs:attribute name="Type" type="xs:QName"/>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <xs:element name="Option" type="wsman:OptionType"/>

    <xs:element name="OptionSet">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="wsman:Option" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:complexType>
    </xs:element>

    <xs:complexType name="attributableEmpty">
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>

    <xs:element name="RequestEPR" type="wsman:attributableEmpty"/>
    <xs:element name="EPRInvalid" type="wsman:attributableEmpty"/>
    <xs:element name="EPRUnknown" type="wsman:attributableEmpty"/>
```

```

  <xs:complexType name="RequestedEPRType">
    <xs:choice>
      <xs:element ref="wsa:EndpointReference"/>
      <xs:element ref="wsman:EPRInvalid"/>
      <xs:element ref="wsman:EPRUnknown"/>
    </xs:choice>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="RequestedEPR" type="wsman:RequestedEPRType"/>

  <xs:complexType name="mixedDataType">
    <xs:complexContent mixed="true">
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="skip"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="fragmentMixedDataType">
    <xs:complexContent mixed="true">
      <xs:extension base="wsman:mixedDataType">
        <xs:attribute name="Dialect" type="xs:anyURI" default="http://www.w3.org/TR/1999/REC-xpath-19991116"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="FragmentTransfer" type="wsman:fragmentMixedDataType"/>
  <xs:element name="XmlFragment" type="wsman:mixedDataType"/>

  <xs:complexType name="attributableNonNegativeInteger">
    <xs:simpleContent>
      <xs:extension base="xs:nonNegativeInteger">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="TotalItemsCountEstimate" type="wsman:attributableNonNegativeInteger" nillable="true"/>
  <xs:element name="RequestTotalItemsCountEstimate" type="wsman:attributableEmpty"/>

  <xs:element name="OptimizeEnumeration" type="wsman:attributableEmpty"/>
  <xs:element name="MaxElements" type="wsman:attributablePositiveInteger"/>

  <xs:simpleType name="EnumerationModeType">
    <xs:restriction base="xs:token">
      <xs:enumeration value="EnumerateEPR"/>
      <xs:enumeration value="EnumerateObjectAndEPR"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="EnumerationMode" type="wsman:EnumerationModeType"/>

  <xs:complexType name="mixedDataFilterType" mixed="true">
    <xs:complexContent mixed="true">
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:any namespace="##any" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="filterMixedDataType" mixed="true">
    <xs:complexContent mixed="true">
      <xs:extension base="wsman:mixedDataFilterType">
        <xs:attribute name="Dialect" type="xs:anyURI" default="http://www.w3.org/TR/1999/REC-xpath-19991116"/>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="Filter" type="wsman:filterMixedDataType"/>

  <xs:complexType name="ObjectAndEPRType">
    <xs:sequence>
      <xs:any namespace="##any" processContents="lax"/>
      <xs:element ref="wsa:EndpointReference"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Item" type="wsman:ObjectAndEPRType"/>

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

  <xs:element name="Items" type="wsman:anyListType"/>
  <xs:element name="EndOfSequence" type="wsman:attributableEmpty"/>

  <xs:complexType name="attributableLanguage">
    <xs:simpleContent>
      <xs:extension base="xs:language">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:element name="ContentEncoding" type="wsman:attributableLanguage"/>

  <xs:complexType name="ConnectionRetryType">
    <xs:simpleContent>
      <xs:extension base="wsman:attributableDuration">
        <xs:attribute name="Total" type="xs:unsignedLong"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:element name="ConnectionRetry" type="wsman:ConnectionRetryType"/>

  <xs:element name="Heartbeats" type="wsman:attributableDuration"/>
  <xs:element name="SendBookmarks" type="wsman:attributableEmpty"/>

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

  <xs:element name="Bookmark" type="wsman:mixedDataType"/>
  <xs:element name="MaxTime" type="wsman:attributableDuration"/>

  <xs:complexType name="EventType">
    <xs:complexContent>
      <xs:extension base="wsman:attributableAny">
        <xs:attribute name="Action" type="xs:anyURI" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="Event" type="wsman:EventType"/>

  <xs:complexType name="EventsType">
    <xs:sequence>
      <xs:element ref="wsman:Event" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="Events" type="wsman:EventsType"/>

  <xs:element name="AckRequested" type="wsman:attributableEmpty"/>

  <xs:complexType name="attributableInt">
    <xs:simpleContent>
      <xs:extension base="xs:int">
        <xs:anyAttribute namespace="##other" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="DroppedEventsType">
    <xs:simpleContent>
      <xs:extension base="wsman:attributableInt">
        <xs:attribute name="Action" type="xs:anyURI" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:element name="DroppedEvents" type="wsman:DroppedEventsType"/>

  <xs:simpleType name="restrictedProfileType">
    <xs:restriction base="xs:anyURI">
      <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic"/>
      <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest"/>
      <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/basic"/>
```

```
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/digest"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/digest"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego-kerberos"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/spnego-kerberos"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/spnego-kerberos"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="ProfileType">
      <xs:union memberTypes="wsman:restrictedProfileType xs:anyURI"/>
    </xs:simpleType>

    <xs:complexType name="AuthType">
      <xs:complexContent>
        <xs:extension base="wsman:attributableEmpty">
          <xs:attribute name="Profile" type="wsman:ProfileType" use="required"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="Auth" type="wsman:AuthType"/>

    <xs:simpleType name="ThumbprintType">
        <xs:restriction base="xs:string">
            <xs:pattern value="[0-9a-fA-F]{40}"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="CertificateThumbprint" type="wsman:ThumbprintType"/>


    <xs:simpleType name="restrictedFaultDetailType">
      <xs:restriction base="xs:anyURI">
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ActionMismatch"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AsynchronousRequest"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Bookmarks"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/CharacterSet"/>
        <xs:enumeration value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DeliveryRetries"/
```

```
   >
(349)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DuplicateSelector
   s"/>
(350)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EncodingType"/>
(351)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EnumerationMode"/
   >
(352)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime"/>
(353)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired"/>
(354)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired
   "/>
(355)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch"/>
(356)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAcce
   ss"/>
(357)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats"/>
(358)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress"/
   >
(359)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelec
   tors"/>
(360)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Invalid"/>
(361)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName"/>
(362)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFragment"/
   >
(363)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace"
   />
(364)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceUR
   I"/>
(365)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue"/>
(366)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues"/>
(367)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale"/>
(368)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements"/>
(369)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy
   "/>
(370)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize"/
   >
(371)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime"/>
(372)        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MinimumEnvelopeLi
   mit"/>
(373)        <xs:enumeration
```

```
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OperationTimeout"
   />
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionLimit"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ResourceOffline"/
   >
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/SelectorLimit"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLi
   mit"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelecto
   rs"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSucce
   ss"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnsupportedCharac
   ter"/>
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnusableAddress"/
   >
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/URILimitExceeded"
   />
        <xs:enumeration
   value="http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Whitespace"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="FaultDetailType">
      <xs:union memberTypes="wsman:restrictedFaultDetailType xs:anyURI"/>
    </xs:simpleType>

    <xs:element name="FaultDetail" type="wsman:FaultDetailType"/>
    <xs:element name="FragmentDialect" type="wsman:attributableURI"/>
    <xs:element name="SupportedSelectorName" type="xs:NCName"/>

    <!-- Master Fault Table subcode QNames -->
    <xs:element name="AccessDenied"><xs:complexType/></xs:element>
    <xs:element name="AlreadyExists"><xs:complexType/></xs:element>
    <xs:element name="CannotProcessFilter"><xs:complexType/></xs:element>
    <xs:element name="Concurrency"><xs:complexType/></xs:element>
    <xs:element name="DeliveryRefused"><xs:complexType/></xs:element>
    <xs:element name="EncodingLimit"><xs:complexType/></xs:element>
    <xs:element name="EventDeliverToUnusable"><xs:complexType/></xs:element>
    <xs:element
   name="FragmentDialectNotSupported"><xs:complexType/></xs:element>
    <xs:element name="InternalError"><xs:complexType/></xs:element>
    <xs:element name="InvalidBookmark"><xs:complexType/></xs:element>
    <xs:element name="InvalidOptions"><xs:complexType/></xs:element>
    <xs:element name="InvalidParameter"><xs:complexType/></xs:element>
    <xs:element name="InvalidSelectors"><xs:complexType/></xs:element>
```

```
  <xs:element name="NoAck"><xs:complexType/></xs:element>
  <xs:element name="QuotaLimit"><xs:complexType/></xs:element>
  <xs:element name="SchemaValidationError"><xs:complexType/></xs:element>
  <xs:element name="TimedOut"><xs:complexType/></xs:element>
  <xs:element name="UnsupportedFeature"><xs:complexType/></xs:element>

</xs:schema>
```

ANNEX L
(informative)

Change Log

Version	Date	Description
1.0.0	2008-02-12	Released as Final Standard
1.1.0	2010-03-03	Released as DMTF Standard, with the following changes: • Incorporates TEEN specifications inline • Addresses consistency issues with DSP0227 on Put and Fragment Put
1.1.1	2012-07-30	Incorporate additional clarifying text to Forward section for ISO/IEC publication as Publicly Available Specification (PAS)
1.1.1	2012-08-28	DMTF Standard
1.2.0a	2014-06-16	wgv 0.8.0 Release as DMTF Work In Progress
1.2.0b	2014-06-16	wgv 0.9.0 Release as DMTF Standard