**Document Number: DSP0226**

**Date: 2008-02-12**

**Version: 1.0.0**

# Web Services for Management (WS-Management) Specification

**Document Type: Specification**

**Document Status: Final Standard**

**Document Language: E**

# CONTENTS

Foreword ..................... vi
1 Scope ..................... 1
2 Normative References ..................... 1
2.1 Approved References ..................... 1
2.2 Other References ..................... 2
3 Terms and Definitions ..................... 2
4 Symbols and Abbreviated Terms ..................... 4
5 Addressing ..................... 6
5.1 Endpoint References ..................... 6
5.2 mustUnderstand Usage ..................... 15
5.3 wsa:To ..................... 15
5.4 Other WS-Addressing Headers ..................... 16
6 WS-Management Control Headers ..................... 22
6.1 wsman:OperationTimeout ..................... 22
6.2 wsman:MaxEnvelopeSize ..................... 23
6.3 wsman:Locale ..................... 24
6.4 wsman:OptionSet ..................... 25
6.5 wsman:RequestEPR ..................... 28
7 Resource Access ..................... 29
7.1 WS-Transfer ..................... 29
7.2 Addressing Uniformity ..................... 31
7.3 WS-Transfer:Get ..................... 31
7.4 WS-Transfer:Put ..................... 32
7.5 WS-Transfer:Delete ..................... 34
7.6 WS-Transfer:Create ..................... 34
7.7 Fragment-Level WS-Transfer ..................... 36
7.8 Fragment-Level WS-Transfer:Get ..................... 38
7.9 Fragment-Level WS-Transfer:Put ..................... 39
7.10 Fragment-Level WS-Transfer:Delete ..................... 42
7.11 Fragment-Level WS-Transfer:Create ..................... 43
8 WS-Enumeration ..................... 45
8.1 General ..................... 45
8.2 WS-Enumeration:Enumerate ..................... 45
8.3 Filter Interpretation ..................... 50
8.4 WS-Enumeration:Pull ..................... 52
8.5 WS-Enumeration:Release ..................... 54
8.6 Ad-Hoc Queries and Fragment-Level Enumerations ..................... 54
8.7 Enumeration of EPRs ..................... 55
9 Custom Actions (Methods) ..................... 57
10 Eventing ..................... 57
10.1 General ..................... 57
10.2 Subscribe ..................... 58
10.3 GetStatus ..................... 74
10.4 Unsubscribe ..................... 74
10.5 Renew ..................... 74
10.6 SubscriptionEnd ..................... 75
10.7 Acknowledgement of Delivery ..................... 75
10.8 Refusal of Delivery ..................... 77
10.9 Dropped Events ..................... 77
11 Metadata and Discovery ..................... 79

12 Security ..... 81
12.1 Security Profiles ..... 82
12.2 Security Considerations for Event Subscriptions ..... 82
12.3 Including Credentials with a Subscription ..... 83
12.4 Correlation of Events with Subscription ..... 86
12.5 Transport-Level Authentication Failure ..... 87
12.6 Security Implications of Third-Party Subscriptions ..... 87
13 Transports and Message Encoding ..... 87
13.1 SOAP ..... 87
13.2 Lack of Response ..... 88
13.3 Replay of Messages ..... 88
13.4 Encoding Limits ..... 88
13.5 Binary Attachments ..... 89
13.6 Case-Sensitivity ..... 89
14 Faults ..... 90
14.1 Introduction ..... 90
14.2 Fault Encoding ..... 90
14.3 NotUnderstood Faults ..... 92
14.4 Degenerate Faults ..... 92
14.5 Fault Extensibility ..... 93
14.6 Master Faults ..... 93
ANNEX A (informative) Notational Conventions ..... 112
ANNEX B (normative) Conformance ..... 114
ANNEX C (normative) HTTP(S) Transport and Security Profile ..... 115
(informative) ..... 123
ANNEX D XPath Support ..... 123
ANNEX E (normative) Selector Filter Dialect ..... 129
ANNEX F (informative) WS-Management XSD ..... 131
ANNEX G (informative) Acknowledgements ..... 132

## Tables

Table 1 – wsa:Action URI Descriptions ..... 21
Table 2 – wsman:AccessDenied ..... 93
Table 3 – wsa:ActionNotSupported ..... 94
Table 4 – wsman:AlreadyExists ..... 94
Table 5 – wsen:CannotProcessFilter ..... 95
Table 6 – wsman:CannotProcessFilter ..... 95
Table 7 – wsman:Concurrency ..... 96
Table 8 – wse:DeliveryModeRequestedUnavailable ..... 96
Table 9 – wsman:DeliveryRefused ..... 97
Table 10 – wsa:DestinationUnreachable ..... 97
Table 11 – wsman:EncodingLimit ..... 98
Table 12 – wsa:EndpointUnavailable ..... 99
Table 13 – wsman:EventDeliverToUnusable ..... 99
Table 14 – wse:EventSourceUnableToProcess ..... 100
Table 15 – wsen:FilterDialectRequestedUnavailable ..... 100
Table 16 – wse:FilteringNotSupported ..... 100

Table 17 – wsen:FilteringNotSupported ..... 101
Table 18 – wse:FilteringRequestedUnavailable ..... 101
Table 19 – wsman:FragmentDialectNotSupported ..... 102
Table 20 – wsman:InternalError ..... 102
Table 21 – wsman:InvalidBookmark ..... 103
Table 22 – wsen:InvalidEnumerationContext ..... 103
Table 23 – wse:InvalidExpirationTime ..... 104
Table 24 – wsen:InvalidExpirationTime ..... 104
Table 25 – wse:InvalidMessage ..... 105
Table 26 – wsa:InvalidMessageInformationHeader ..... 105
Table 27 – wsman:InvalidOptions ..... 106
Table 28 – wsman:InvalidParameter ..... 106
Table 29 – wxf:InvalidRepresentation ..... 107
Table 30 – wsman:InvalidSelectors ..... 107
Table 31 – wsa:MessageInformationHeaderRequired ..... 108
Table 32 – wsman:NoAck ..... 108
Table 33 – wsman:QuotaLimit ..... 108
Table 34 – wsman:SchemaValidationError ..... 109
Table 35 – wsen:TimedOut ..... 109
Table 36 – wsman:TimedOut ..... 109
Table 37 – wse:UnableToRenew ..... 110
Table 38 – wse:UnsupportedExpirationType ..... 110
Table 39 – wsen:UnsupportedExpirationType ..... 110
Table 40 – wsman:UnsupportedFeature ..... 111
Table A-1 – Prefixes and XML Namespaces Used in This Specification ..... 113
Table C-1 – Basic Authentication Sequence ..... 117
Table C-2 – Digest Authentication Sequence ..... 118
Table C-3 – Basic Authentication over HTTPS Sequence ..... 118
Table C-4 – Digest Authentication over HTTPS Sequence ..... 119
Table C-5 – HTTPS with Client Certificate Sequence ..... 119
Table C-6 – Basic Authentication over HTTPS with Client Certificate Sequence ..... 120
Table C-7 – SPNEGO Authentication over HTTPS Sequence ..... 121
Table C-8 – SPNEGO Authentication over HTTPS with Cilent Certificate Sequence ..... 121
Table D-1 – XPath Level 1 Terminals ..... 125
Table D-2 – XPath Level 2 Terminals ..... 127

# 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.

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

# Web Services for Management (WS-Management) Specification

## 1 Scope

The *Web Services for Management (WS-Management) Specification* describes a general Web services protocol based on SOAP for managing systems 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 composability with other Web services specifications.
- Minimize additional mechanisms beyond the current Web services architecture.

## 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.

### 2.1 Approved References

IETF, RFC 3066, H. Alvestrand, *Tags for the Identification of Languages*, January 2001.

IETF, RFC 3986, T. Berners-Lee et al, *Uniform Resource Identifiers (URI): Generic Syntax*, August 1998.

IETF, RFC 4559, K. Jaganathan et al, *SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows*, June 2006.

OASIS, A. Nadalin et al, *Web Services Security Username Token Profile 1.0*, March 2004.

OASIS, S. Anderson et al, *Web Services Trust Language (WS-Trust)*, December 2005.

The Unicode Consortium, *The Unicode Standard v3.0*, January 2000.

W3C, M. Gudgin, et al, *SOAP Version 1.2 Part 1: Messaging Framework*, June 2003.

W3C, M. Gudgin, et al, *SOAP Message Transmission Optimization Mechanism (MTOM)*, November 2004.

W3C, D. Box et al, *Web Services Addressing (WS-Addressing)*, August 2004.

W3C, J. Alexander et al, *Web Services Enumeration (WS-Enumeration)*, March 2006.

W3C, D. Box et al, *Web Services Eventing (WS-Eventing)*, March 2006.

W3C, S. Bajaj, et al, *Web Services Policy Framework (WS-Policy)*, April 2006.

W3C, J. Alexander et al, *Web Services Transfer (WS-Transfer)*, September 2006.

W3C, J. Clark et al, *XML Path Language Version 1.0 (XPath 1.0)*, November 1999.

W3C, J. Cowan et al, *XML Information Set Second Edition (XML Infoset)*, February 2004.

W3C, H. Thompson et al, *XML Schema Part 1: Structures (XML Schema 1)*, May 2001.

W3C, P. Biron et al, *XML Schema Part 2: Datatypes (XML Schema 2)*, May 2001.

## 2.2 Other References

IETF, RFC 2478, E. Baize et al, *The Simple and Protected GSS-API Negotiation Mechanism*, December 1998.

IETF, RFC 2616, R. Fielding et al, *Hypertext Transfer Protocol (HTTP 1.1)*, June 1999.

IETF, RFC 2818, E. Rescorla, *HTTP over TLS (HTTPS)*, May 2000.

IETF, RFC 4122, P. Leach et al, *A Universally Unique Identifier (UUID) URN Namespace*, July 2005.

K. Ballinger et al, *Web Services Metadata Exchange (WS-MetadataExchange)*, September 2004.

OASIS, G. Della-Libera et al, *WS-Secure Conversation 1.3*, May, 2004.

OASIS, A. Nadalin et al, *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*, March 2004.

W3C, M. Gudgin, et al, *SOAP Version 1.2 Part 2: Adjuncts*, June 2003.

W3C, E. Christensen et al, *Web Services Description Language Version 1.1 (WSDL/1.1)*, March 2001.

W3C, S. Boag et al, XQuery 1.0: An XML Query Language (XQuery 1.0), January 2007.

# 3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

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

### 3.13
**event sink**

a Web service that receives notifications (defined in WS-Eventing)

### 3.14

**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.

### 3.15

**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.

### 3.16

**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.

### 3.17

**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.

### 3.18

**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.

# 4 Symbols and Abbreviated Terms

The following symbols and abbreviations are used in this document.

### 4.1

**BNF**

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

# 5 Addressing

WS-Management relies on WS-Addressing to define references to other Web service endpoints and to define some of the headers used in SOAP messages.

## 5.1 Endpoint References

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

### 5.1.1 Use of WS-Addressing Endpoint References

WS-Management uses WS-Addressing EPRs as the addressing model 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 systems with well-established addressing models or with opaque EPRs retrieved from a discovery service, services may use those service-specific addressing models if they are based on WS-Addressing.

> **R5.1.1-1**: All messages that are addressed to a resource class or instance that are referenced by an EPR must follow the WS-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 wsen:Pull or wsen: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 WS-Addressing EPR is still required in the message to help route it to the correct handler.

Rule **R5.1.1-1** clarifies that messages such as wsen:Pull or wse:Renew still require a full EPR. For wsen:Pull, for example, this EPR would be the same as the original wsen:Enumerate, even though wsen: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 wse:Renew request uses the EPR obtained by the wse:SubscriptionManager received in the wse:SubscribeResponse.

When a service includes an EPR in a response message, it must be willing to accept that EPR in a subsequent request message 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.

> **R5.1.1-2**: An EPR returned by a service shall be acceptable to that service to refer to the same managed resource.

Additionally, EPRs must be durable: when a service returns an EPR to a client, that EPR must continue to be valid while the managed resource still exists.

> **R5.1.1-3**: All EPRs returned by a service, whether expressed using the WS-Management default addressing model (see 5.1.2) or any other addressing model, shall be valid as long as the managed resource exists.

### 5.1.2 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.

The remainder of this document often 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 5.1.2 and 5.1.2.2 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): 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:

```
<wsa:EndpointReference>
  <wsa:Address>
    Network address
  </wsa:Address>
  <wsa:ReferenceParameters>
    <wsman:ResourceURI> resource URI </wsman:ResourceURI>
    <wsman:SelectorSet>
      <wsman:Selector Name="selector-name"> *
        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 5.1.2.2

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

The preceding format is used when defining addresses in metadata, or when specifying return addresses in message bodies, as in wse:NotifyTo, wsa:ReplyTo, and wsa:FaultTo.

When the default addressing model is used in a SOAP message, WS-Addressing specifies that translations take place and the headers are flattened out. Although this requirement is described in WS-Addressing, it is worth repeating because of its critical nature.

EXAMPLE: The following is an example EPR definition:

```
 <wsa:EndpointReference xmlns:wsa="...">
   <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 properties and reference parameters are unwrapped and juxtaposed.

```
<s:Envelope ...>
  <s:Header>
    <wsa:To> Address </wsa:To>
    <wsa:Action> Action URI </wsa:Action>
    <wsman:ResourceURI 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 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 WS-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 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.

#### 5.1.2.1 ResourceURI

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

**R5.1.2.1-1**: 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.

**R5.1.2.1-2**: 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>
```

**R5.1.2.1-3**: 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 wse:SubscriptionManager element of the wse:SubscribeResponse message (WS-Eventing), 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.

**R5.1.2.1-4:** 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.

**R5.1.2.1-5**: 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.

**R5.1.2.1-6**: When the default addressing model is used and the wsman:ResourceURI element is missing or of the 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.

**R5.1.2.1-7**: The wsman:ResourceURI element shall be used to indicate only the identity of a resource, and may not be used to indicate the action being applied to that resource, which is properly expressed using the wsa:Action URI.

Note that 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 wsen:Enumerate operations.

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:

```
(1)  <s:Header>
(2)    <wsa:To>
(3)      http://1.2.3.4/wsman/
(4)    </wsa:To>
(5)    <wsman:ResourceURI>http://example.org/2005/02/networkcards/reset
         </wsman:ResourceURI>
(6)    <wsa:Action>
(7)      http://example.org/2005/02/networkcards/reset
(8)    </wsa:Action>
(9)    ...
(10) </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:

```
(1)  <s:Header>
(2)    <wsa:To>
(3)      http://1.2.3.4/wsman
(4)    </wsa:To>
(5)    <wsman:ResourceURI>http://example.org/2005/02/networkcards
         </wsman:ResourceURI>
(6)    <wsa:Action>
(7)      http://example.org/2005/02/networkcards/reset
(8)    </wsa:Action>
(9)    ...
(10) </s:Header>
```

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

EXAMPLE 3:

```
(1)  <s:Header>
(2)    <wsa:To>http://1.2.3.4/wsman</wsa:To>
(3)    <wsman:ResourceURI>
(4)      http://example.org/products/management/networkcards
(5)    </wsman:ResourceURI>
(6)    <wsa:Action>
(7)      http://example.org/2005/02/netcards/reset
(8)    </wsa:Action>
(9)    ...
(10) </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.

#### 5.1.2.2 Selectors

In the WS-Management default addressing model, selectors are optional elements used to identify instances within a resource class. For operations such as wxf:Get or wxf: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.

**R5.1.2.2-1**: 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.

**R5.1.2.2-2**: All content within the SelectorSet element is to be treated as a single reference parameter with a scope relative to the ResourceURI.

**R5.1.2.2-3**: 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

**R5.1.2.2-4**: 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:

```
(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>  service transport address </wsa:To>
(8)     <wsman:ResourceURI> ResourceURI </wsman:ResourceURI>
(9)     <wsman:SelectorSet>
(10)      <wsman:Selector Name="name"> value </wsman:Selector> +
(11)    </wsman:SelectorSet> ?
(12)    ...
(13)  </s:Header>
(14)  <s:Body> ... </s:Body>
(15) </s:Envelope>
```

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

wsa:To

network address

wsman:ResourceURI

used to indicate the resource class

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">
```

```
        <wsa:EndpointReference>
          <wsa:Address> address </wsa:Address>
          <wsa:ReferenceParameters>
            <wsman:ResourceURI> resource URI </wsman:ResourceURI>
            <wsman:SelectorSet>
              <wsman:Selector Name="name"> value </wsman:Selector>
            </wsman:SelectorSet>
          </wsa:ReferenceParameters>
        </wsa:EndpointReference>
      </wsman:Selector>
    </wsman:SelectorSet>
    ...
  </s:Header>
  <s:Body> ... </s:Body>
</s:Envelope>
```

**R5.1.2.2-5**: 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 of a supported schema.

**R5.1.2.2-6**: 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

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

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

**R5.1.2.2-7**: A service may fail to process a selector name of more than 2048 characters.

**R5.1.2.2-8**: 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.

#### 5.1.2.3 Faults for Default Addressing Model

When faults based on the default format are generated, they often 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.

### 5.1.3 Service-Specific Endpoint References

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

**R5.1.3-1**: A conformant service may not understand the header values used by the WS-Management default addressing model. If the client marks the wsman:ResourceURI with mustUnderstand="true", the service shall return an s:NotUnderstood fault.

**R5.1.3-2**: 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 WS-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 WS-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.

## 5.2 mustUnderstand Usage

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, as mustUnderstand is implied:

- wsa:To
- wsa:MessageID
- wsa:RelatesTo
- wsa:Action

**R5.2-1**: 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.

**R5.2-2**: 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.

## 5.3 wsa:To

In request messages, the wsa:To address contains the network 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 EPR typically contains additional fields (reference parameters) to allow the service to identify a resource within its scope. For example, when the default addressing model is in use, these additional fields are the ResourceURI and SelectorSet fields.

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.

**R5.3-1**: 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>
```

**R5.3-2**: 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>
```

If the service exposes only one set of resources, the wsa:To header is the only addressing element required.

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.

NOTE: All secondary messages that are continuations of prior messages, such as wsen:Pull or wsen:Release (both of which continue wsen:Enumerate), still contain an EPR. The fact that these messages also contain context information from a prior message is not material to the SOAP messaging and addressing model.

**R5.3-3**: The service should issue faults when failing to evaluate the address of the resource in 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 Other WS-Addressing Headers

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

### 5.4.1 Processing WS-Addressing Headers

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

**R5.4.1-1**: A conformant service shall recognize and process the following WS-Addressing header blocks. Any others are optional as specified in WS-Addressing and may be present, but a conformant service may reject any additional headers and fail to process the message, issuing a s:NotUnderstood fault.

- **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.

## 5.4.2 wsa:ReplyTo

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

> **R5.4.2-1**: 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 URI http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous (see WS-Addressing), 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, wse:SubscriptionEnd, and so on, do not require a response and may omit a wsa:ReplyTo element.

> **R5.4.2-2**: 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 **R5.4.2-1** 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

> **R5.4.2-3**: 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.

> **R5.4.2-4**: The service shall fully duplicate the entire wsa:Address of the wsa:ReplyTo element in the wsa:To header of the reply, even if some of the information is not understood by the service.

This rule applies in cases where the client includes suffixes on the HTTP or HTTPS address that the service does not understand. The service returns these suffixes nonetheless.

> **R5.4.2-5**: Any reference parameters supplied in the wsa:ReplyTo address shall be included in the actual response message as top-level headers as specified in WS-Addressing unless the response is a fault. If the response is a fault, the service should include the reference parameters but may omit these values if the resulting message size would exceed encoding limits.

WS-Addressing allows clients to include client-defined reference parameters in wsa:ReplyTo headers. The *WS-Addressing* specification 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:

```
(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:ReplyTo>
(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:ReplyTo>
(16)     ...
(17)   </s:Header>
(18)   <s:Body> ... </s:Body>
(19) </s:Envelope>
```

**R5.4.2-6**: 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.

### 5.4.3 wsa:FaultTo

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

**R5.4.3-1**: 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.

**R5.4.3-2**: If wsa:FaultTo is omitted, the service shall return the fault to the wsa:ReplyTo address if a fault occurs.

**R5.4.3-3**: 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 http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous (see the *WS-Addressing* specification). 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.

**R5.4.3-4**: Any reference parameters supplied in the wsa:FaultTo address should be included as top-level headers in the actual fault, as specified in the *WS-Addressing* specification. In some cases, including this information would cause the fault to exceed encoding size limits, and thus may be omitted in those cases.

WS-Addressing allows clients to include client-defined reference parameters in wsa:FaultTo headers. The *WS-Addressing* specification requires that these reference parameters be extracted from requests and placed in the faults by removing the ReferenceParameters wrapper and placing all of the values as top-level SOAP headers in the fault. This allows clients to better correlate faults with the original requests. This step can be omitted in cases where the resulting fault would be large enough to exceed encoding limit restrictions (see 6.2, rules in 13.1, and rules in 13.4).

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>
```

**R5.4.3-5**: If the wsa:FaultTo address is not usable, the service should not reply to the request. Similarly, if no wsa:FaultTo address is supplied, and the service does not have sufficient information to fault the response properly, 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.

**R5.4.3-6**: 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.

## 5.4.4 wsa:MessageID and wsa:RelatesTo

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

**R5.4.4-1**: 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 IETF 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.

**R5.4.4-2**: 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 (**R5.4.4-1**), 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.

**R5.4.4-3**: 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.

**R5.4.4-4**: 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>
```

NOTE: The mustUnderstand attribute can be omitted for either wsa:MessageID or wsa:RelatesTo with no change in meaning.

### 5.4.5 wsa:Action

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

**R5.4.5-1**: The wsa:Action URI shall not be used to identify the specific resource class or instance, but only to identity the operation to use against that resource.

**R5.4.5-2**: For all resource endpoints, a service shall return a wsa:ActionNotSupported fault (defined in WS-Addressing) 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.

**R5.4.5-3**: 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 1 according to the rules of this specification. The service may optionally expose additional similar functionality using a distinct wsa:Action URI.

**Table 1 – 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" |
| 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<br>(not required in WS-Management) |
| http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse | Response to "Renew"<br>(not required in WS-Management) |
| http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus | Gets the status of the enumerator<br>(not required in WS-Management) |
| http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse | Response to "GetStatus"<br>(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<br>(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 |

| Action URI | Description |
|---|---|
| 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 |

**R5.4.5-4**: A custom action may be supported if the operation is a custom method whose semantic meaning is not present in the table, or if the item is an event.

**R5.4.5-5**: All event deliveries shall contain a unique action URI that identifies the type of the event delivery. For singleton deliveries 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 9 of this specification.

### 5.4.6 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.

**R5.4.6-1**: 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.

**R5.4.6-2**: 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 note that 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 wxf: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, 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 wxf:Put or wxf:Delete, or a wxf: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 3066 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:

```
(1)<s:Envelope
(2)    xml:lang="en-us"
(3)    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(4)    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(5)    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(6)  <s:Header> ... </s:Header>
(7)  <s:Body> ... </s:Body>
(8)</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 wsen:Enumerate and wsen:Pull messages. The locale is clearly established during the initial wsen:Enumerate request, so changing the locale during the enumeration serves no purpose. The service ignores any wsman:Locale elements in subsequent wsen:Pull messages, but the client can ensure that the value does not change between wsen: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, the service may ignore the entire wsman:OptionSet block. If it is present 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 wsen:Enumerate and wsen:Pull messages. The set of options is established once during the initial wsen: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 wse: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 WS-Transfer "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. Note that 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

Resource access applies to all synchronous operations regarding getting, setting, and enumerating values. The WS-Transfer**Error! Reference source not found.** specification is used as a basis for simple unary resource access: Get, Put, Delete, and Create. Multi-instance retrieval is achieved using WS-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.

## 7.1 WS-Transfer

WS-Transfer brings wxf:Get, wxf:Put, wxf:Create, and wxf:Delete into the WS-Management space.

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

```
<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/Get
    </wsa:Action>
    <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>
```

Note that the wsa:ReplyTo occurs on the same connection as the request (line 10), the action is a wxf:Get (line 14), and the ResourceURI (line 17) 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 note that the s:Body has no content in a wxf:Get request.

EXAMPLE (continued): The following shows a hypothetical response to the preceding hypothetical wxf: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>
```

Note 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 wxf:Delete, except that no content exists in the response s:Body. The wxf:Create and wxf: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

In general, the service can expose addressing usage that is identical for the WS-Transfer operations. Where practical, the EPR of the resource can be the same whether a wxf:Get, wxf:Delete, or wxf: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.

wxf: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 wxf:Create operation would not have the same addressing headers as the corresponding wxf:Get or wxf: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 wxf:Get, wxf:Put, and wxf: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 wxf: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 namespaces declarations would be considered invalid.

When performing WS-Transfer operations, side effects might occur. For example, deletion of a particular resource by using wxf:Delete can result in several other dependent instances disappearing, and a wxf:Create operation can result in the logical creation of more than one resource that can be subsequently returned through a wxf:Get operation. Similarly, a wxf: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 WS-Transfer:Get

The wxf:Get operation retrieves resource representations. The message can be targeted to return a complex XML Infoset (an "object") 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 wxf: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 wxf: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, wxf:Get is designed to return XML that correspond 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 WS-Transfer (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 WS-Transfer:Put

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 wxf:Put operation.

**R7.4-1**: A conformant service may support wxf: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 wxf:Get, a service should support updating the resource by using wxf:Put. The service may additionally export a custom method for updates.

**R7.4-3**: If a single resource instance contains a mix of read-only and read-write values, the wxf:Put message may contain both the read-only and read-write values if the XML content is legal with regard to its XML schema namespace. In such cases, the service should ignore the read-only values during the update operation. If none of the values are writeable, the service should return a wsa:ActionNotSupported fault.

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

Note that a complication arises because wxf: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 wxf:Put message, even if it is not being altered from its original value.

If the schema definition has default values for elements that are optional (minOccurs=0), the wxf:Put message can omit these values and rely on the defaults being observed during the update.

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

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

```
(1)  <s:Body>
(2)    <MyObject xmlns="examples.org/2005/02/MySchema">
(3)      <A> 100 </A>
(4)      <B> 200 </B>
(5)      <C> 100 </C>
(6)    </MyObject>
(7)  </s:Body>
```

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

```
(8)  <xs:element name="MyObjecct">
(9)    <xs:complexType>
(10)      <xs:sequence>
(11)        <xs:element name="A" type="xs:int" minOccurs="1" maxOccurs="1"/>
(12)        <xs:element name="B" type="xs:int" minOccurs="1" maxOccurs="1"/>
(13)        <xs:element name="C" type="xs:int" minOccurs="1" maxOccurs="1"/>
(14)        ...
(15)      </xs:sequence>
(16)    </xs:complexType>
(17) </xs :element>
```

In this case, the corresponding wxf: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 wxf:Get to preserve the current values of <A> and <C>, change <B> to the desired value, and then write the object using wxf: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.

To update isolated values without having to supply values that will not change, use the fragment-level transfer mechanism described in 7.7.

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

**R7.4-5**: If the supplied Body does not have the correct content to update the resource, the service should return a wxf: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

**R7.4-6**: If an object cannot be updated because of locking conditions, simultaneous access, or similar conflicts, the service should return a wsman:Concurrency fault.

**R7.4-7**: A wxf: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 wxf: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 wxf:Put.

**R7.4-8**: 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 wxf:Put message. Because implementations can rarely assure this, they can always return the new representation.

**R7.4-9**: 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

**R7.4-10**: The wxf: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 WS-Transfer:Delete

The WS-Transfer:Delete operation deletes resource instances.

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

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

**R7.5-2**: A conformant service should support wxf:Delete using the same EPR as a corresponding wxf: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 wxf:Get, a conformant service should support deletion using wxf: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, wxf: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 which itself is not to be deleted, either the client can perform a wxf:Put operation with those properties removed, or the service can support fragment-level WS-Transfer (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 WS-Transfer:Create

The WS-Transfer:Create operation creates resources and models a logical "constructor." In general, the addressing is not the same as that used for wxf:Get or wxf: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 wxf:Create.

**R7.6-2**: If a single resource can be created using a SOAP message and that resource can be subsequently retrieved using wxf:Get, then a service should support creation of the resource using wxf: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 wxf: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 wxf:Create to perform an update on an existing representation. If the targeted object already exists, the service should return a wsman:AlreadyExists fault.

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

WS-Transfer specifies that wxf:CreateResponse contains the initial representation of the object. However, due to restrictions in *WSDL/1.1* (and the upcoming *WSDL 2.0* specification), a SOAP Body cannot actually be defined that contains juxtaposed elements at the top level.

This specification places a restriction such that the only returned value is the wxf:ResourceCreated element, which contains the EPR of the newly created resource.

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 WS-Transfer (7.7).

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

**R7.6-6**: The response shall not contain the initial representation of the object, in spite of language within the *WS-Transfer* specification.

This last restriction is due to the fact that some SOAP processors cannot process multiple child elements within a SOAP s:Body. In general, clients can simply issue a wxf:Get message to retrieve the representation, because they will have just acquired an EPR to the new resource.

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:wxf="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>
    <wxf:ResourceCreated>
      <wsa:Address>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous/
      </wsa:Address>
      <wsa:ReferenceParameters>
        <wsman:ResourceURI>
          http://example.org/2005/02/virtualDrive
        </wsman:ResourceURI>
        <wsman:SelectorSet>
          <wsman:Selector Name="ID"> F: </wsman:Selector>
        </wsman:SelectorSet>
      </wsa:ReferenceParameters>
    </wxf:ResourceCreated>
  </s:Body>
</s:Envelope>
```

This example assumes that the default addressing model is in use. The response contains a wxf: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 wxf:Get operation.

Note that the service might use a network address that is the same as the <wsa:To> address in the wxf:Create request, or it might simply use the anonymous address as shown (line 16).

**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 wxf:Get, wxf:Put, and wxf:Create to share the same schema. Note that wxf: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 section 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 WS-Transfer

Because WS-Transfer 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 WS-Transfer operations. This access is done through special use of WS-Transfer.

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 transfer of fragments or individual elements of a representation object, several modifications to the basic WS-Transfer operations are made.

**R7.7-1**: A conformant service may support fragment-level WS-Transfer. If the service supports fragment-level access, the service shall not behave as if normal WS-Transfer 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 WS-Transfer 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:

```
<wsman:FragmentTransfer s:mustUnderstand="true"
  Dialect="URIToNewFragmentDialect">
  dialect expression
</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.

Note that XPath is explicitly defined as a dialect due to its importance, but it is not mandated 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 WS-Transfer 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 WS-Transfer Get operation was applied against the addressed resource without using fragment transfer
- **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 WS-Enumeration, the XPath is interpreted as defined in the *WS-Enumeration* specification, 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 WS-Transfer a proper superset of normal WS-Transfer.

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 transfers 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 transfers 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 wxf:InvalidRepresentation fault with the following detail code:

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

Fragment transfer 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 WS-Transfer 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 WS-Transfer 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" WS-Transfer operations.

The following clauses show how the underlying WS-Transfer operations change when transferring XML fragments.

## 7.8 Fragment-Level WS-Transfer:Get

Fragment-level WS-Transfer:Get is similar to full wxf: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 will execute 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 wxf: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>
(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 and might or might not contain XML namespace information or attributes.

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 WS-Transfer:Put

Fragment-level WS-Transfer:Put works like regular wxf: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: wxf:Put is *always* an update operation of an existing element, whether a simple element or an array. To create or insert new elements, wxf: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 wxf: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 7.1, if the client wanted to update the <BootPartition> value from 0 to 1, the following wxf: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 wxf: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**: As for normal wxf:Put, the service may ignore any read-only values supplied as part of the fragment for updating.

**R7.9-2**: If the service encounters an attempt to update a read-only value, 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 wxf: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 wxf:Create operation.

WS-Transfer states that if the new representation differs from the input, the new representation is to be returned in the response. With fragment-level wxf: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:

```
<s:Header> ...
  <wsman:FragmentTransfer s:mustUnderstand="true">
    AssetLabel
  </wsman:FragmentTransfer>
  ...
</Header>
<s:Body>
  <wsman:XmlFragment xmlns:xsi="www.w3.org/2001/XMLSchema-instance">
    <AssetLabel xsi:nil="true"/>
  </wsman:XmlFragment>
</s:Body>
```

## 7.10 Fragment-Level WS-Transfer:Delete

Fragment-level WS-Transfer: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 WS-Transfer: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:

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

To set a value to NULL without removing it as an element, use fragment-level wxf: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 wxf:Delete operations, even with fragment-level access, and all normal faults for wxf: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 WS-Transfer:Create

Fragment-level WS-Transfer: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 wxf:Put can be used. For array access, the XPath array access notation (the [ ] operators) can conveniently be used.

NOTE: wxf: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 wxf:Create contains the value to be inserted and is the same as for fragment-level wxf: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:

```
<wsman:FragmentTransfer s:mustUnderstand="true">
  BlockedIPAddress[3]
</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. Note that using wxf:Put at this location *overwrites* the existing array element, whereas wxf:Create inserts a *new* element, making the array larger.

The body of wxf:Create contains the value to be inserted and is the same as for fragment-level wxf: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 wxf:Create to perform an update on an existing representation. If the targeted object already exists, the service should return a wsman:AlreadyExists fault.

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

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

EXAMPLE 7: In the following example, note that the wxf:ResourceCreated EPR continues to refer to the entire object, not just 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:wxf="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)    <wxf:ResourceCreated>
(42)      <wsa:Address> ... </wsa:Address>
(43)      <wsa:ReferenceParameters>
(44)        <wsman:SelectorSet>
(45)          <wsman:Selector ...> ... </wsman:Selector>
(46)        </wsman:SelectorSet>
(47)      </wsa:ReferenceParameters>
(48)    </wxf: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 the *WS-Transfer* specification.

# 8 WS-Enumeration

The *WS-Enumeration* specification is used as a basis for iteration through the members of a collection. WS-Management qualifies and extends WS-Enumeration as described in this clause.

## 8.1 General

If a resource with multiple instances provides a mechanism for enumerating or querying the set of instances, WS-Enumeration performs the iteration.

**R8.1-1**: A service may support WS-Enumeration 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 WS-Enumeration 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 WS-Enumeration to expose this. The service may also support other techniques for enumerating the instances.

The *WS-Enumeration* specification indicates that enumeration is a three-part operation:

1) An initial wsen:Enumerate message is issued to establish the enumeration context.
2) wsen:Pull operations are used to iterate over the result set.
3) When the enumeration iterator is no longer required and not yet exhausted, a wsen: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 can implement any of the following messages from WS-Enumeration, but implementing them is not recommended: Renew, GetStatus, or EnumerationEnd, and any associated responses. Because these messages are optional, it is recommended that the service fault both Renew and GetStatus requests with a wsa:ActionNotSupported fault.

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

If the service does not support stateful enumerators, the wsen: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 wsen:Pull and wsen:Release operations are a continuation of the original wsen: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 wsen:Enumerate and subsequent wsen:Pull operations, or between wsen: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 WS-Enumeration:Enumerate

Enumerations are initiated by the wsen:Enumerate message.

### 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 wsen:Enumerate message with an EndTo address; however, **R8.1-4** recommends not supporting the EnumerationEnd message, so 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 a wsen: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 wsen: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 WS-Enumeration, 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 in WS-Enumeration).

**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 wsen: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 wxf:Create or wxf: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 wsen: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 wsen:Enumerate and wsen:Pull messages and their responses. The header used in wsen:Enumerate and wsen: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 wsen:Enumerate or wsen: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 wsen:EnumerateResponse or wsen: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 will respond 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 a wsen:Enumerate or wsen: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 wsen:EnumerateResponse could reasonably include the entire result, without the need for a subsequent wsen: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 child element of the wsen:Enumerate element, and can optionally include the wsman:MaxElements element, as follows:

EXAMPLE:

```
(1)  <s:Body>
(2)    <wsen:Enumerate>
(3)      ...
(4)      <wsman:OptimizeEnumeration/>
(5)      <wsman:MaxElements>xs:positiveInteger</wsman:MaxElements> ?
(6)    </wsen:Enumerate>
(7)  </s:Body>
```

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

wsen:Enumerate/wsman:OptimizeEnumeration

when present as a child of the wsen:Enumerate element, indicates that the client is requesting an optimized enumeration

wsen:Enumerate/wsman:MaxElements

(optional) indicates the maximum number of items the consumer is willing to accept in the wsen:EnumerateResponse

It plays the same role as wsen:Pull/wsen: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 a wsen:Enumerate message and it does not support enumeration optimization, it should ignore the element and complete the enumeration request as described in WS-Enumeration.

If the service ignores the element, the client continues with a subsequent wsen:Pull as if the option was not in force. The client requires no special mechanisms over what was needed for normal WS-Enumeration if the optimization request is ignored.

**R8.2.3-2**: A conformant service that receives a wsen:Enumerate message without the wsman:OptimizeEnumeration element shall not return any enumeration items in the wsen:EnumerateResponse message and shall return a wsen:EnumerationContext initialized to return the first items when the first wsen: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 a wsen: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 a wsen:EnumerateResponse message:

```
(1)  <s:Body>
(2)    <wsen:EnumerateResponse>
(3)      <wsen:EnumerationContext> ... </wsen:EnumerationContext>
(4)      <wsman:Items>
(5)        ...same as for wsen:Items in wsen:PullResponse
(6)      </wsman:Items> ?
(7)      <wsman:EndOfSequence/> ?
(8)      ...
(9)    </wsen: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 wsen:Items in a wsen: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

wsen:EnumerationContext

required context for requesting additional items, if any, in subsequent Pull messages

If the wsman:EndOfSequence is also present, the wsen:EnumerationContext cannot be used in a subsequent wsen:Pull request. The service should observe the same fault usage that would occur if the wsen:EnumerationContext were used in a wsen:Pull request after the wsen:EndOfSequence element occurred in a wsen:PullResponse. Although the wsen:EnumerationContext element must be present, no value is required; therefore, in cases where the wsman:EndOfSequence element is present, the value for wsen:EnumerationContext can be empty.

EXAMPLE:

```
(1) <s:Body>
(2)     <wsen:EnumerateResponse>
(3)       <wsen:EnumerationContext/>
(4)       <wsman:Items>
(5)         Items
(6)       </wsman:Items>
(7)       <wsman:EndOfSequence/>
(8)       ...
(9)     </wsen:EnumerateResponse>
(10)</s:Body>
```

**R8.2.3-4**: A conformant service that supports optimized enumeration and is responding with a wsen: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 wsen:EnumerateResponse message, the client can continue to use the enumeration message exchanges as they are defined in WS-Enumeration.

**R8.2.3-5**: A conformant service that supports optimized enumeration and has not returned all items of the enumeration sequence in the wsen:EnumerateResponse message shall return a wsen:EnumerationContext element that is initialized such that a subsequent wsen:Pull message will return the set of items after those returned in the wsen:EnumerateResponse. If all items of the enumeration sequence have been returned in the wsen:EnumerateResponse message, the service should return an empty wsen: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

In WS-Enumeration, 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 WS-Enumeration 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 wsen:Enumerate element.

If the filter dialect used for the wsen:Enumerate message is XPath 1.0, the context node is the same as that specified by WS-Enumeration.

**R8.3-1**: If a service supports filtered enumeration using wsen: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 wsen:Filter. This rule allows clients coded to WS-Enumeration to interact with a WS-Management service.

**R8.3-3**: If a wsen:Enumerate request contains both wsen:Filter and wsman:Filter, the service shall return a wsen:CannotProcessFilter fault.

Filters are generally intended to select entire XML infosets or "object" 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 WS-Transfer (7.7), the service can implement category 3 as well. Only rarely will services implement category 4.

XPath 1.0 can be used simply for filtering, or 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>
  ...
  <wsen:Items>
    <DiskInfo xmlns="...">
      <LogicalDisk>C:</LogicalDisk>
      <CurrentMegabytes>12</CurrentMegabytes>
      <BackupDrive> true </BackupDrive>
    </DiskInfo>
    ...
  </wsen:Items>
</s:Body>
```

The anchor point for the XPath evaluation is at the first element of each item within the wsen:Items wrapper, and it does not reference the s:Body or wsen:Items elements. The XPath expression is evaluated as if each item in the wsen:Items block was 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 WS-Eventing subscriptions.

## 8.4 WS-Enumeration:Pull

The wsen:Pull message continues an enumeration—that is, it retrieves batches of results from the initial wsen:Enumerate message.

Because wsen: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 wsen:Pull message instead of wsen: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 and supports wsen:Pull with the wsen:MaxCharacters element, the service should implement wsen: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 and supports wsen:Pull with the wsen: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 wsen: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, wsen:MaxCharacters is a hint, and wsman:MaxEnvelopeSize is a strict rule.

**R8.4-3**: If any fault occurs during a wsen:Pull, a compliant service should allow the client to retry wsen: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 a wsen:InvalidEnumerationContext fault.

If a fault occurs with a wsen:Pull request, the service generally does not need to cancel the entire enumeration, but 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 wsen:Pull, it is not required to do so and can in fact change the EnumerationContext unpredictably.

**R8.4-4**: A conformant service may ignore wsen: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 wsen:MaxTime and wsen:MaxCharacters to allow for uniform message construction.

Any fault issued for wsen:Pull applies to the wsen: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 wsen: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**: The service shall accept a wsen:Pull message with an EPR identical to that specified for the original wsen:Enumerate message. A wsa:MessageInformationHeaderRequired fault should be returned if the EPR is missing or different.

If no content is available, the enumerator is still considered active and the wsen:Pull message can be retried.

**R8.4-6**: If a service cannot populate the wsen: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 wsen: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 wsen:EnumerationContext, which may have changed, even though no items were returned, as follows:

```
<s:Body>
  <wsen:PullResponse>
    <wsen:EnumerationContext> ...possibly updated... </wsen:EnumerationContext>
    <wsen:Items/>
  </wsen:PullResponse>
</s:Body>
```

An empty wsen: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 wsen: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 wsen:Pull message times out, the service can simply send back zero items in the expectation that the client can continue with another wsen:Pull message.

**R8.4-7**: The service may terminate the entire enumeration early at any time, in which case a wsen:InvalidEnumerationContext fault is returned. No further operations are possible, including wsen: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 wsen:EndOfSequence marker occurs in the wsen:PullResponse message, the wsen:EnumerationContext element shall be omitted, as the enumeration has completed. The client cannot subsequently issue a wsen:Release message.

Normally, the end of an enumeration in all cases is reported by the wsen:EndOfSequence element being present in the wsen:PullResponse content, not through faults. If the client attempts to enumerate past the end of an enumeration, a wsen:InvalidEnumerationContext fault is returned. The client need not issue a wsen:Release message if the wsen:EndOfSequence actually occurs because the enumeration is then completed and the enumeration context is invalid.

**R8.4-9**: If no wsen:MaxElements element is specified, the batch size is 1, as specified in WS-Enumeration.

**R8.4-10**: If the value of wsen: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 wsen:MaxElements value in metadata, but the format and location of such metadata is beyond the scope of this specification.

**R8.4-11**: The wsen:EnumerationContext element shall be present in all wsen:Pull requests, even if the service uses a constant value for the lifetime of the enumeration sequence. This rule is mandated by WS-Enumeration and repeated here for clarity.

## 8.5 WS-Enumeration:Release

wsen: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 wsen:Release message if the enumeration is terminated early. If a wsen:EndOfSequence marker occurs in a wsen:PullResponse message, the enumerator is already completed and a wsen:Release message cannot be issued because no up-to-date wsen:EnumerationContext exists.

**R8.5-2**: The client may fail to deliver the wsen: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 a wsen:InvalidEnumerationContext fault.

**R8.5-3**: The service shall accept a wsen:Release message with an EPR identical to that specified for the original wsen:Enumerate message, assuming the enumeration is still active and the wsen:EndOfSequence element has not been sent by the service. If the EPR is missing or different, the service should return a wsa:MessageInformationHeaderRequired fault.

**R8.5-4**: The service may accept a wsen:Release message asynchronously to any wsen: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 wsen:Pull message.

In most cases, it is desirable to be able to asynchronously cancel an outstanding wsen:Pull message. This capability requires the service to be able to receive the wsen:Release message asynchronously while still processing a pending wsen:Pull message. Further, it requires that the wsen:EnumerationContext element contain information that is constant between wsen:Pull operations. Note that if the value of wsen:EnumerationContext is a simple increasing integer, wsen:Release will always be using a previous value, so the service might consider it to be invalid. If the wsen:EnumerationContext element contains a value that is constant across wsen: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 wsen:PullResponse element (or wsen:EnumerateResponse element if OptimizedEnumeration is used) should be wrapped with wsman:XmlFragment wrappers as follows:

```
(1) <s:Body>
(2)   <wsen:PullResponse>
(3)     <wsen:EnumerationContext> ..possibly updated.. </wsen:EnumerationContext>
(4)   <wsen:Items>
(5)     <wsman:XmlFragment>
(6)       XML content
(7)     </wsman:XmlFragment>
(8)     <wsman:XmlFragment>
(9)       XML content
(10)      </wsman:XmlFragment>
(11)      ...
(12)    </wsen:Items>
(13)  </wsen: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 wsen:FilterDialectRequestedUnavailable fault, the same as for any other unsupported dialect.

The XPath expression used for filtering is still that described in the *WS-Enumeration* specification. 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 wxf:Get or wxf:Delete requests, for example. Similarly, it is often desirable to enumerate both the objects and the associated EPRs.

The default behavior for wsen:Enumerate is as defined in the *WS-Enumeration* specification. 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 ...>
(2)    <s:Header>
(3)      ...
(4)      <wsa:Action>
(5)        http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
(6)      </wsa:Action>
```

```
    ...
  </s:Header>
  <s:Body>
    <wsen:Enumerate>
      <wsman:Filter Dialect="...">  filter </wsman:Filter>
      <wsman:EnumerationMode> EnumerateEPR </wsman:EnumerationMode>
      ...
    </wsen:Enumerate>
  </s:Body>
</s:Envelope>
```

The hypothetical response would appear as in the following example:

```
<s:Body>
  <wsen:PullResponse>
    <wsen:Items>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      ...
    </wsen:Items>
  </wsen: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 wxf: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 1: The wsman:EnumerationMode example appears as follows:

```
<s:Header>
  ...
  <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
  </wsa:Action>
</s:Header>
<s:Body>
  <wsen:Enumerate>
    <wsman:Filter Dialect="...">  filter </wsman:Filter>
      <wsman:EnumerationMode> EnumerateObjectAndEPR </wsman:EnumerationMode>
      ...
  </wsen:Enumerate>
</s:Body>
```

EXAMPLE 2: The response appears as follows:

```
<s:Body>
  <wsen:PullResponse>
    <wsen:Items>
      <wsman:Item>
        <PayloadObject xmlns="..."> ... </PayloadObject>  <!-- Object -->
        <wsa:EndpointReference> ... </wsa:EndpointReference>     <!-- EPR -->
      </wsman:Item>
      <wsman:Item>
        <PayloadObject xmlns="..."> ... </PayloadObject>  <!-- Object -->
          <wsa:EndpointReference> ... </wsa:EndpointReference>     <!-- EPR -->
```

```
(11)       </wsman:Item>
(12)       ...
(13)     </wsen:Items>
(14)   </wsen: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

# 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, WS-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 WS-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 WS-Addressing Action URI.

Because options are a parameterization technique for message types that are not user-extensible, such as WS-Transfer, 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 Eventing

WS-Management provides eventing functionality through the *WS-Eventing* specification. The use of WS-Eventing for management is qualified and extended as described in this clause.

## 10.1 General

If the service emits events, it can publish those events using WS-Eventing messaging and paradigms. WS-Management places additional restrictions and constraints on the general *WS-Eventing* specification.

**R10.1-1**: If a resource can emit events and allows clients to subscribe to and receive event messages, it shall do so by implementing WS-Eventing as specified in this specification.

**R10.1-2**: If WS-Eventing is supported, the Subscribe, Renew, and Unsubscribe messages shall be supported. SubscriptionEnd is optional, and GetStatus is not recommended.

## 10.2 Subscribe

The Subscribe message allows a client to express interest in receiving events. WS-Management qualifies this message in this clause.

### 10.2.1 General

WS-Management uses wse:Subscribe substantially as documented in WS-Eventing, 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 WS-Addressing EPR.

**R10.2.1-2**: A service need not support distinct addresses and distinct security settings for wse:NotifyTo and wse:EndTo, and may require that these be the same network address, although they may have separate reference parameters in all cases. 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 wse:NotifyTo address in the wse:Delivery block can be used for this purpose. This wse:NotifyTo operation 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 wse: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 wse:NotifyTo address shall be included with each event delivery as top-level headers as specified in WS-Addressing. If wse: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 wse:Delivery block of the wse:Subscribe message. The language is encoded in an xml:lang attribute using RFC 3066 language codes.

The service attempts to localize any descriptive content to the specified language when delivering such events, which is outlined as follows:

```
(1)  <wse:Subscribe>
(2)    <wse:Delivery>
(3)      <wse:NotifyTo>  ... </wse:NotifyTo>
(4)      <wsman:Locale xml:lang="language-code"/>
(5)    </wse:Delivery>
(6)  </wse:Subscribe>
```

In this context, note that 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 wse:Delivery block of the wse: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)<wse:Subscribe>
(2)  <wse:Delivery>
(3)    ...
(4)    <wse:NotifyTo>  ... </wse:NotifyTo>
(5)    <wsman:ContentEncoding> UTF-16 </wsman:ContentEncoding>
(6)  </wse:Delivery>
(7)</wse:Subscribe>
```

### 10.2.2 Filtering

In WS-Eventing, 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 WS-Eventing 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 wse:Filter or wsman:Filter used for the wse:Subscribe message is http://www.w3.org/TR/1999/REC-xpath-19991116 (the default dialect in both cases), the context node is that specified by WS-Eventing (the SOAP Envelope element).

WS-Management defines the wsman:Filter element as a child of the wse: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 wse: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 wse:Filter. This rule allows clients coded to WS-Eventing to interact with a WS-Management service.

**R10.2.2-3**: If a wse:Subscribe request contains both wse:Filter and wsman:Filter, the service shall return a wse:InvalidMessage fault.

To allow eventing filter expressions to be defined independent of the delivery mode, WS-Management defines a new filter dialect that is the same as defined by WS-Eventing 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 was 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/wse: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 by WS-Eventing.

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 will use XPath only for filtering, but 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>
```

Note that 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 was 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.

Note that XPath 1.0 can be used simply for filtering or 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 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 wse: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 fragment-level WS-Transfer (see 7.7 of this specification).

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 WS-Enumeration, 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 will attempt 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)  <wse:Subscribe>
(2)    <wse:Delivery>
(3)      <wse:NotifyTo>  ... </wse:NotifyTo>
(4)      <wsman:ConnectionRetry Total="count"> xs:duration </wsman:ConnectionRetry>
(5)    </wse:Delivery>
(6)  </wse: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 above interval between the attempts

> **R10.2.3-2**: If the retry counts are exhausted, the subscription should be considered expired and any normal operations that would occur upon expiration should occur.

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 wse:SubscribeResponse

The service returns any service-specific reference parameters in the wse:SubscriptionManager EPR, and these are included by the subscriber (client) later when issuing Unsubscribe and Renew messages.

> **R10.2.4-1**: In wse:SubscribeResponse, the service may specify any EPR for the wse:SubscriptionManager. However, it is recommended that the address contain the same wsa:To address as the original wse: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 wse:Expires field in the response, but as specified in WS-Eventing, 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 wse:Subscribe request has an additional field in the wse:Delivery section:

```
(1)  <wse:Delivery>
(2)    ...
(3)    <wsman:Heartbeats> xs:duration </wsman:Heartbeats>
(4)    ...
(5)  </wse: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 wsen: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 wsen: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 will assume 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, as 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:

```
<s:Envelope ...>
  <s:Header>
    <wsa:To> .... </wsa:To>
    <wsa:Action s:mustUnderstand="true">
      http://schemas.dmtf.org/wbem/wsman/1/wsman/Heartbeat
    </wsa:Action>
    ...
  </s:Header>
  <s:Body/>
</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 for WS-Eventing, 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 wse:Subscribe request as follows:

```
<s:Body>
  <wse:Subscribe>
    <wse:Delivery>
        ...
    </wse:Delivery>
    <wsman:SendBookmarks/>
  </wse: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 wse: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.

```
(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 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 wse: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 wse:Subscribe message as a new block, positioned after the WS-Eventing-defined child elements of wse:Subscribe, as in the following outline:

```
(1)  <s:Body>
(2)   <wse:Subscribe>
(3)     <wse:Delivery>  ... </wse:Delivery>
(4)     <wse:Expires> ... </wse:Expires>
(5)     <wsman:Filter> ... </wsman:Filter>
(6)     <wsman:Bookmark>
(7)       ...last known bookmark from a previous delivery...
(8)     </wsman:Bookmark>
(9)     <wsman:SendBookmarks/>
(10)  </wse: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

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 wse: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

Note that 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 will require a new delivery mode.

**R10.2.7-2**: The wse:NotifyTo address in the wse: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 wse: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 the delivery modes defined by WS-Eventing and WS-Management.

#### 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 wsen: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 from WS-Eventing 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 wse:SubscriptionManager content from the wse: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)  <wse:Delivery>
(2)    <wse:NotifyTo> ... </wse:NotifyTo>
(3)    ...
(4)    <wsman:MaxEnvelopeSize Policy="enumConstant">
(5)      xs:positiveInteger
(6)    </wsman:MaxEnvelopeSize>
(7)    ...
(8)  </wse:Delivery>
```

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

wse:Delivery/wsman:MaxEnvelopeSize
  the maximum number of octets for the entire SOAP envelope in a single event delivery

wse: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 wse: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 wse:DeliveryModeRequestedUnavailable.

For this delivery mode, the wse:Delivery element has the following format:

```
(1)  <wse:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
(2)    <wse:NotifyTo>
(3)      wsa:EndpointReferenceType
(4)    </wse: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) </wse:Delivery>
```

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

wse:Delivery/@Mode

required attribute that shall be defined as

http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

wse:Delivery/wse:NotifyTo

required element that shall contain the EPR to which event messages are to be sent for this subscription

wse: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.

wse: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 events
- **Skip:** silently skip oversized events
- **Notify:** notify the subscriber that events were dropped as specified in 10.9

wse: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 none of the MaxElements, MaxEnvelopeSize, and MaxTime elements are 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:

```
(1)  <s:Envelope ...>
(2)    <s:Header>
(3)      ...
(4)      <wsa:Action>
(5)        http://schemas.dmtf.org/wbem/wsman/1/wsman/Events
(6)      </wsa:Action>
(7)      ...
(8)    </s:Header>
(9)    <s:Body>
(10)     <wsman:Events>
(11)       <wsman:Event Action="event action URI">
(12)         ...event body...
(13)       </wsman:Event> +
(14)     </wsman:Events>
(15)   </s:Body>
(16) </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 wse: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)  <wse:Delivery
(3)    Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
(4)    <wse:NotifyTo>
(5)      <wsa:Address>http://2.3.4.5/client</wsa:Address>
(6)    </wse:NotifyTo>
(7)    <wsman:MaxElements>10</wsman:MaxElements>
(8)    <wsman:MaxTime>PT20S</wsman:MaxTime>
(9)    <wsman:MaxEnvelopeSize>8192</wsman:MaxEnvelopeSize>
(10) </wse: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:wse="http://schemas.xmlsoap.org/ws/2004/09/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 wsen:Pull message to retrieve events from the logical queue. Non-delivery subscription processing continues to use messages from WS-Eventing. (For example, wse:Unsubscribe, rather than wsen:Release, is used to cancel a subscription.)

For this delivery mode, the wse:Delivery element has the following format:

```
(1)  <wse:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull">
(2)    ...
(3)  </wse:Delivery>
```

wse: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 wse:DeliveryModeRequestedUnavailable.

wsman:MaxElements, wsman:MaxEnvelopeSize, and wsman:MaxTime do not apply in the wse:Subscribe message when using this delivery mode because the wsen: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 a wsen:EnumerationContext element suitable for use in a subsequent wsen:Pull operation.

EXAMPLE:

```
(1)  <s:Body ...>
(2)    <wse:SubscribeResponse ...>
(3)      <wse:SubscriptionManager>
(4)        wsa:EndpointReferenceType
(5)      </wse:SubscriptionManager>
(6)      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires>
(7)      <wsen:EnumerationContext>...</wsen:EnumerationContext>
(8)      ...
(9)    </wse:SubscribeResponse>
(10) </s:Body>
```

The subscriber extracts the wsen:EnumerationContext and uses it thereafter in wsen:Pull requests.

**R10.2.9.5-4**: If pull mode is active, wsen:Pull messages shall use the EPR of the subscription manager obtained from the wse: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 wsen:Pull request returns no events (because none have occurred since the last "pull"), the service should return a wsman:TimedOut fault. The wsen:EnumerationContext is still considered active, and the subscriber may continue to issue wsen:Pull requests with the most recent wsen:EnumerationContext for which event deliveries actually occurred.

**R10.2.9.5-6**: If pull mode is active and a wsen:Pull request returns events, the service may return an updated wsen:EnumerationContext as specified for wsen:Pull, and the subscriber is expected to use the update, if any, in the subsequent wsen:Pull, as specified for WS-Enumeration. 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 wsen: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 a wsen: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 wsen: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 wsen:Pull.

## 10.3 GetStatus

The GetStatus message is optional for WS-Management.

**R10.3-1**: A conformant service may implement the GetStatus message or its response; however, it is not recommended that services implement this for future compatibility.

If implemented, WS-Management adds no new information to the request or response beyond that defined in WS-Eventing. Heartbeat support can be implemented rather than GetStatus.

## 10.4 Unsubscribe

The wse:Unsubscribe message cancels a subscription.

**R10.4-1**: If a service supports wse:Subscribe, it shall implement the wse: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 wse: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 a wse:Unsubscribe message. Clients can fault any such deliveries or accept them, at their option.

The EPR to use for this message is received from the wse:SubscribeResponse element in the wse:SubscriptionManager element.

## 10.5 Renew

According to WS-Eventing, processing the wse:Renew message is required, but it is not required to succeed.

**R10.5-1**: Although a conformant service shall accept the wse:Renew message as a valid action, the service may always fault the request with a wse:UnableToRenew fault, forcing the client to subscribe from scratch.

wse: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 wse:SubscribeResponse element in the wse:SubscriptionManager element.

## 10.6 SubscriptionEnd

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. If it is implemented, the service may fail to accept a subscription with any address differing from the NotifyTo address.

**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**: If SubscriptionEnd is supported, it is recommended that it be sent to the subscriber prior to sending the UnsubscribeResponse.

If the service delivers events over the same connection as the wse:Subscribe operation, the client typically knows that a subscription has been terminated because the connection itself will close or terminate.

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 WS-Addressing.

wsman:AckRequested

no content; requires that the subscriber acknowledge all deliveries as described below

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.

Note that simple push mode has no way for the client to fault a delivery or acknowledge 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 a wse: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 wse:SubscriptionEnd message to the wse: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 wse: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 wse: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>
</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

wse: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:

```
<wsman:Events>
  <wsman:Event Action="https://foo.com/someEvent">
    …event body
  </wsman:Event>
  <wsman:Event
    Action="http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents">
  <wsman:DroppedEvents Action="https://foo.com/someEvent">
    1
  </wsman:DroppedEvents>
  </wsman:Event>
  <wsman:Event Action="https://foo.com/someEvent">
    ...event body...
  </wsman:Event>
<wsman:Events>
```

Note that the DroppedEvent is an event in itself.

**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.

# 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.

Note the absence of any WS-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: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 WS-Addressing, WS-Transfer, and so on. For example, if a future version of WS-Management supports multiple versions of WS-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.

In addition, vendor-specific content can follow these standardized elements.

**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 WS-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 WS-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

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.

It is expected that more sophisticated transport and SOAP-level profiles will be defined and used, published separately from this specification.

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.

## 12.1 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.

The authentication profiles do not appear in the SOAP traffic, with the exception of the wse: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 will have 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.

## 12.2 Security Considerations for Event Subscriptions

When specifying the wse: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 wse:Delivery block that can communicate authentication information, as shown in the following outline:

```
(1)  <s:Body>
(2)    <wse:Subscribe>
(3)      <wse:Delivery>
(4)        <wse:NotifyTo>  Delivery EPR </wse:NotifyTo>
(5)        <wsman:Auth Profile="authentication-profile-URI"/>
(6)      </wse:Delivery>
(7)    </wse:Subscribe>
(8)  </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 wse: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:

```
(1)  <s:Body>
(2)    <wse:Subscribe>
(3)      <wse:Delivery>
(4)      <wse:NotifyTo> HTTPS address </wse:NotifyTo>
(5)        <wsman:Auth
(6)        Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/
             mutual"/>
(7)      </wse:Delivery>
(8)    </wse:Subscribe>
(9)  </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:

```
(1)  <s:Body>
(2)    <wse:Subscribe>
(3)      <wse:Delivery>
(4)        <wse:NotifyTo> HTTP address </wse:NotifyTo>
(5)        <wsman:Auth
(6)          Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/
               digest"/>
(7)      </wse:Delivery>
(8)    </wse:Subscribe>
(9)  </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.

## 12.3 Including Credentials with a Subscription

In addition to specifying the authentication profile using the wsman:Auth block, the subscriber might want to send additional tokens to the service to indicate which credentials to use when making the connection to deliver events. As stated in 12.2, if no tokens are specified, by default the service needs to be preconfigured to know which credentials to use. However, the service can require that the client supply partial or full credentials with the subscription to use later when making the connection to deliver the events.

The communication of credentials is independent of the authentication mode communicated in the wsman:Auth block. The same user name, password, or certificate identity could be used with a variety of transports.

By default, standard communication of credentials is done using a WS-Trust wst:IssuedTokens header block as defined in section 6.4 of the *WS-Trust* specification. Use of WS-Trust for this purpose helps to assure interoperation of secured event delivery.

EXAMPLE: In the following example, the user name token is conveyed in the headers to the wse:Subscribe message in a wst:IssuedTokens block (lines 10–29):

```
(1)  <s:Envelope ...
(2)     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
(3)     xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
(4)
(5)   <s:Header ...>
(6)     <wsa:Action>
(7)       http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
(8)     </wsa:Action>
(9)     ...
(10)    <wst:IssuedTokens mustUnderstand="true">
(11)      <wst:RequestSecurityTokenResponse>
(12)        <wst:TokenType>
(13)          http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-
(14)            token-profile-1.0#UsernameToken
(15)        </wst:TokenType>
(16)
(17)        <wst:RequestedSecurityToken>
(18)          <wsse:UsernameToken>
(19)            <wsse:Username>JoeUser</wsse:Username>
(20)          </wsse:UsernameToken>
(21)        </wst:RequestedSecurityToken>
(22)
(23)        <wsp:AppliesTo>
(24)          <wsa:EndpointReference><!-- NotifyTo EPR -->
(25)            <wsa:Address><!-- address of event sink --></wsa:Address>
(26)          </wsa:EndpointReference>
(27)        </wsp:AppliesTo>
(28)      </wst:RequestSecurityTokenResponse>
(29)    </wst:IssuedTokens>
(30)
(31)  </s:Header>
(32)  <s:Body ...>
(33)    <wse:Subscribe ...>
(34)      <wse:Delivery>
(35)        <wse:NotifyTo> ... </wse:NotifyTo>
(36)        ...
(37)      </wse:Delivery>
(38)      ...
(39)    </wse:Subscribe>
(40)  </s:Body>
(41) </s:Envelope>
```

This wst:IssuedTokens block is divided into three sections:

- the type of token or credential being passed: the wst:TokenType wrapper (lines 12–15)

  This can refer to user names, X.509 certificates, or other token types.

- the actual security token in a wst:RequestedSecurityToken wrapper (lines 17–21)
- what the tokens apply to: the wsp:AppliesTo block from WS-Policy (lines 23–27)

  In this case, the tokens apply to the wse:NotifyTo address in the subscription. The wse:NotifyTo EPR and the wsp:AppliesTo shall be identical.

Note that the wst:IssuedTokens block needs to have a SOAP mustUnderstand attribute.

The communication of tokens to the service for later use in event delivery connections is independent of the security profile in use. Typically, the subscriber passes one of the following tokens to the service using WS-Trust:

- a user name reference (the service knows the password or other related credentials and only uses the user name as a hint to know which credential to use)
- an X.509 certificate identifier (thumbprint or "hash")

  The service has more than one certificate and needs to know which one to use.
- a user name and password combination, which is directly used to make the connection in the other direction at event-time

  This token type has security implications and is not to be delivered to the service over an unencrypted network transport.
- some combination of the preceding token types (such as a user name and a cookie)

These tokens are all intended for use at the transport level when making the connection and do not appear in the SOAP messages. Other token types can be communicated as well, but they are beyond the scope of this specification.

**R12.3-1**: Whenever a user name is communicated to the service, the following WS-Trust usage should be observed. The wst:TokenType shall be the following URI:

```
<wst:TokenType>
  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-
      profile-1.0#UsernameToken
<wst:/TokenType>
```

Additionally, the wst:RequestedSecurityToken shall be a wsse:UsernameToken that contains the user name:

```
<wst:RequestedSecurityToken>
  <wsse:UsernameToken>
    <wsse:Username>user-name/wsse:Username>
  </wsse:UsernameToken>
</wst:RequestedSecurityToken>
```

The wsse:UsernameToken is defined in *WS-Security Username Token Profile 1.0* (WS-Security Token). WS-Management does not require the use or presence of the other fields in wsse:UsernameToken element, although the implementation can return appropriate errors if other submitted fields are not supported, such as wsse:Nonce.

The password can be optionally supplied in clear text as specified in *WS-Security Token*, but it is best delivered over an encrypted transport:

```
<wst:RequestedSecurityToken>
  <wsse:UsernameToken>
    <wsse:Username>user-name/wsse:Username>
    <wsse:Password>password</wsse:Password>
  </wsse:UsernameToken>
</wst:RequestedSecurityToken>
```

**R12.3-2**: Whenever an X.509 certificate identity is communicated to the service, the following WS-Trust usage should be observed. The wst:TokenType shall be the following URI:

```
<wst:TokenType>
  http://schemas.dmtf.org/wbem/wsman/1/wsman/token/certificateThumbprint
<wst:/TokenType>
```

The wst:RequestedSecurityToken shall be a wsman:CertificateThumbprint that identifies the exact certificate to be used as the client certificate in mutual authentication:

```
<wst:RequestedSecurityToken>
  <wsman:CertificateThumbprint>
    8e5255328d03543a6aa6ea9cf7977ec9b4d7fdb3
  </wsman:CertificateThumbprint></wst:RequestedSecurityToken>
```

This token type contains the SHA-1 hash of the certificate as a hexadecimal string (referred to as the "thumbprint").

NOTE: Although the *WS-Trust* and the standard *WS-Security Token* profiles referenced in this clause provide other options and mechanisms, their use is optional and beyond the scope of this version of WS-Management.

## 12.4 Correlation of Events with 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 wse:NotifyTo definition.

EXAMPLE: At subscription time, a UUID could be supplied as a correlation token:

```
<s:Body>
  <wse:Subscribe>
    <wse:Delivery>
      <wse:NotifyTo>
        <wsa:Address> address <wsa:Address>
        <wsa:ReferenceParameters>
          <MyNamespace:uuid>
            uuid:b0f685ec-e5c9-41b5-b91c-7f580419093e
          </MyNamespace:uuid>
          </wsa:ReferenceParameters>
      </wse:NotifyTo>
      ...
    </wse:Delivery>
    ...
  </wse:Subscribe>
</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.

## 12.5 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.

## 12.6 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 section 6.2 ("Access Control") of the *WS-Eventing* specification.

# 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, WS-Addressing, or WS-Eventing, the service should not process them and should issue a wsa:InvalidMessageInformationHeaders fault if they are detected. (No SOAP headers are defined by the *WS-Transfer* and *WS-Enumeration* specifications.)

**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, as 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 wsen: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

A service is not to resend messages that have not been acknowledged at the transport level.

**R13.3-1**: A service shall not resend unacknowledged messages unless they are part of a higher, general-purpose, reliable messaging or transactional protocol layer, in which case the retransmission follows the rules for that protocol.

## 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 outlines 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 wxf:Get message, receive the wxf: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>
```

```
(23)      </s:Code>
(24)      <s:Reason>
(25)        <s:Text xml:lang="en">  [Reason] </s:Text>
(26)      </s:Reason>
(27)      <s:Detail>
(28)        [Detail]
(29)      </s:Detail>
(30)    </s:Fault>
(31)  </s:Body>
(32) </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. Note that 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 (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 2 through Table 40, the source specification of a fault is based on its QName.

The list is alphabetized on the primary subcode name, regardless of the namespace prefix.

**Table 2 – 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. |

4165 **Table 3 – 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><br>  <wsa:Action> *Incorrect Action URI* </wsa:Action><br></s:Detail><br><!-- The unsupported Action URI is returned, if possible --> |
| Comments | This fault means that the requested action is not supported by the implementation.<br>As an example, read-only implementations (supporting only wxf:Get and wsen:Enumerate) will return this fault for any operations other than these two.<br>If the implementation never supports the action, the fault can be generated as shown above. 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:<br>  <s:Detail><br>    <wsa:Action> *The offending Action URI* </wsa:Action><br>    <wsman:FaultDetail><br>      http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ActionMismatch<br>    </wsman:FaultDetail><br>  </s:Detail><br>This situation can occur when the implementation supports wxf: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. |

4166 **Table 4 – 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 | wxf:Create |
| Remedy | The client uses wxf:Put or creates a resource with a different identity. |

4167 **Table 5 – wsen:CannotProcessFilter**

| | |
|---|---|
| Fault Subcode | wsen: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><br><wsman:SupportedSelectorName> Valid selector name for use in filter expression </wsman:SupportedSelectorName> *<br></s:Detail> |
| Comments | This fault is returned for syntax errors or other semantic problems with the filter.<br>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.<br>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 | wsen:Enumerate |
| Remedy | The client fixes the filter problem and tries again. |

4168 **Table 6 – 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><br><wsman:SupportedSelectorName> Valid selector name for use in filter expression </wsman:SupportedSelectorName> *<br></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.<br>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.<br>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 wse:EventSourceUnableToProcess. |
| Applicability | wse:Subscribe, fragment-level WS-Transfer operations |
| Remedy | The client fixes the filter problem and tries again. |

4169 **Table 7 – 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.<br>This fault can occur if a resource is being enumerated using wsen:Enumerate and another client attempts operations such as wxf:Delete, which would affect the result of the enumeration in progress. |
| Applicability | All messages |
| Remedy | The client waits and tries again. |

4170 **Table 8 – wse:DeliveryModeRequestedUnavailable**

| | |
|---|---|
| Fault Subcode | wse: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><br><wse:SupportedDeliveryMode>... </wse:SupportedDeliveryMode><br><wse:SupportedDeliveryMode>...</wse:SupportedDeliveryMode><br>...<br></s:Detail><br><!-- 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.<br>If the stack supports the delivery mode in general, but not for the specific resource, this fault is still returned.<br>Other resources might support the delivery mode. The fault does not imply that the delivery mode is not supported by the implementation. |
| Applicability | wse:Subscribe |
| Remedy | The client selects one of the supported delivery modes. |

4171 **Table 9 – 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.<br><br>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 wse:SubscriptionEnd messages. |

4172 **Table 10 – 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 WS-Addressing To header. |
| Detail | <s:Detail><br><br>  <wsman:FaultDetail> http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI </wsman:FaultDetail> ?<br><br></s:Detail><br><br>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.<br><br>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. |

4173 **Table 11 – 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 was processed. |
| Detail | <s:Detail><br>&lt;wsman:FaultDetail&gt;<br>Optional; one of the following enumeration values<br>&lt;/wsman:FaultDetail&gt;<br>...any service-specific additional XML content...<br>&lt;/s:Detail&gt;<br>Possible enumeration values in the &lt;wsman:FaultDetail&gt; element are as follows:<br>Unsupported character set:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/CharacterSet<br>Unsupported MTOM or other encoding types:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EncodingType<br>Requested maximum was too large:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize<br>Requested maximum envelope size was too small:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MinimumEnvelopeLimit<br>Too many options:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionLimit<br>Used when the default addressing model is in use and indicates that too many selectors were used for the corresponding ResourceURI:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/SelectorLimit<br>Service reached its own internal limit when computing response:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit<br>Operation succeeded and cannot be reversed, but result is too large to send:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess<br>Request contained a character outside of the range that is supported by the service:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnsupportedCharacter<br>URI was too long:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/URILimitExceeded<br>Client-side whitespace usage is not supported:<br>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. |

4174 **Table 12 – 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>`<br>`<wsa:RetryAfter>` *xs:duration* `</wsa:RetryAfter>` `<!-- optional -->`<br>...optional service-specific XML content<br>`<wsman:FaultDetail>` *A detail URI value* `</wsman:FaultDetail>`<br>`</s:Detail>`<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ResourceOffline<br>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.<br>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. |

4175 **Table 13 – 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 wse:Delivery element. |
| Detail | `<s:Detail>`<br>...any service-specific content to identify the error...<br>`</s:Detail>` |
| Comments | This fault is limited to cases of connectivity issues in contacting the “deliver to” address. These issues include:<br>• The wse:NotifyTo address is not usable because it is incorrect (system or device not reachable, badly formed address, and so on).<br>• Permissions cannot be acquired for event delivery (for example, the wsman:Auth element does not refer to a supported security profile, and so on).<br>• The credentials associated with the wse:NotifyTo are not valid (for example, the account does not exist, the certificate thumbprint is not a hex string, and so on).<br>The service can include extra information that describes the connectivity error to help in troubleshooting the connectivity problem. |
| Applicability | wse:Subscribe |
| Remedy | The client ensures connectivity from the service computer back to the event sink including firewalls and authentication/authorization configuration. |

4176 **Table 14 – wse:EventSourceUnableToProcess**

| | |
|---|---|
| Fault Subcode | wse: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 | wse:Subscribe |
| Remedy | The client retries the subscription later. |

4177 **Table 15 – wsen:FilterDialectRequestedUnavailable**

| | |
|---|---|
| Fault Subcode | wsen: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><br><wsen:SupportedDialect> .... </wsen:SupportedDialect> +<br></s:Detail> |
| Comments | This fault is returned when the client requests a filter type or query language not supported by the service.<br>The filter dialect can vary from resource to resource or can apply to the entire service. |
| Applicability | wsen:Enumerate |
| Remedy | The client switches to a supported dialect or performs a simple enumeration with no filter. |

4178 **Table 16 – wse:FilteringNotSupported**

| | |
|---|---|
| Fault Subcode | wse: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.<br>Note that the service might support filtering over a different event resource or might not support filtering for *any* resource. The same fault applies. |
| Applicability | wse:Subscribe |
| Remedy | The client subscribes using unfiltered delivery. |

4179 **Table 17 – wsen:FilteringNotSupported**

| | |
|---|---|
| Fault Subcode | wsen: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.<br><br>Note that the service might support filtering over a different enumerable resource or might not support filtering for *any* resource. The same fault applies. |
| Applicability | wsen:Enumerate |
| Remedy | The client switches to a simple enumeration. |

4180 **Table 18 – wse:FilteringRequestedUnavailable**

| | |
|---|---|
| Fault Subcode | wse: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><br>  <wse:SupportedDialect>.. </wse:SupportedDialect> +<br>  <wsman:FaultDetail> ..the following URI, if applicable </wsman:FaultDetail><br></s:Detail><br>Possible URI value:<br>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.<br>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 | wse:Subscribe |
| Remedy | The client switches to a supported filter dialect or uses no filtering. |

4181 **Table 19 – 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 | wsen:Enumerate, wxf:Get, wxf:Create, wxf:Put, wxf:Delete |
| Remedy | The client uses a supported filtering dialect or no filtering. |

4182 **Table 20 – 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. |

4183 **Table 21 – 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><br><wsman:FaultDetail><br>If possible, one of the following URI values<br></wsman:FaultDetail><br></s:Detail><br>Possible URI values:<br>The service is not able to back up and replay from that point:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired<br>The service is not able to decode the bookmark:<br>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 | wsen:Subscribe |
| Remedy | The client issues a new subscription without any bookmarks or locates the correct bookmark. |

4184 **Table 22 – wsen:InvalidEnumerationContext**

| | |
|---|---|
| Fault Subcode | wsen: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 wsen:Pull.<br>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.<br>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 | wsen:Pull, wsen:Release (whether a pull-mode subscription, or a normal enumeration) |
| Remedy | The client abandons the enumeration and lets the service time it out, as wsen:Release will fail as well. |

4185 **Table 23 – wse:InvalidExpirationTime**

| | |
|---|---|
| Fault Subcode | wse: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.<br>This fault is used for outright errors (expirations in the past, for example) or expirations too far into the future.<br>If the service does not support expiration times at all, a wsman:UnsupportedFeature fault can be returned with the correct detail code. |
| Applicability | wse:Subscribe |
| Remedy | The client issues a new subscription with a supported expiration time. |

4186 **Table 24 – wsen:InvalidExpirationTime**

| | |
|---|---|
| Fault Subcode | wsen: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 wsen:Expiration feature, this fault might not occur with most implementations.<br>Consult the *WS-Enumeration* specification for more information. |
| Applicability | wsen:Enumerate |
| Remedy | Not applicable |

4187 **Table 25 – wse:InvalidMessage**

| | |
|---|---|
| Fault Subcode | wse: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.<br>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.<br>This fault is not to be used to indicate unsupported features, only unexpected or unknown content in violation of this specification. |
| Applicability | WS-Eventing request messages |
| Remedy | The client issues valid messages that comply with this specification. |

4188 **Table 26 – 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><br>...the invalid header...<br></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.<br>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.<br>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. |

4189 **Table 27 – 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><br>  <wsman:FaultDetail><br>      If possible, one of the following URI values<br>  </wsman:FaultDetail><br></s:Detail><br>Possible URI values:<br>   http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported<br>   http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName<br>   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. |

4190 **Table 28 – 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><br>  <wsman:FaultDetail><br>      If possible, one of the following URI values<br>  </wsman:FaultDetail><br></s:Detail><br>Possible URI values:<br>   http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch<br>   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.<br>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. |

4191 **Table 29 – wxf:InvalidRepresentation**

| | |
|---|---|
| Fault Subcode | wxf:InvalidRepresentation |
| Action URI | http://schemas.xmlsoap.org/ws/2004/09/transfer/fault |
| Code | s:Sender |
| Reason | The XML content is not valid. |
| Detail | &lt;s:Detail&gt;<br>&lt;wsman:FaultDetail&gt;<br>If possible, one of the following URI values<br>&lt;/wsman:FaultDetail&gt;<br>&lt;/s:Detail&gt;<br>Possible URI values:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace<br>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.<br>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.<br>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 | wxf:Put, wxf:Create |
| Remedy | The client corrects the request XML. |

4192 **Table 30 – 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 | &lt;s:Detail&gt;<br>&lt;wsman:FaultDetail&gt;<br>If possible, one of the following URI values<br>&lt;/wsman:FaultDetail&gt;<br>&lt;/s:Detail&gt;<br>Possible URI values:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelectors<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelectors<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue<br>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. |

4193 **Table 31 – 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><br>The XML QName of the missing header<br></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. |

4194 **Table 32 – 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.<br>For services delivering events, the service cancels the subscription immediately. |

4195 **Table 33 – 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. |

4196 **Table 34 – 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. Note that 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. |

4197 **Table 35 – wsen:TimedOut**

| | |
|---|---|
| Fault Subcode | wsen: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 | wsen:Pull |
| Remedy | The client can retry the wsen:Pull request. |

4198 **Table 36 – 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 wsen:Pull request. Clients can simply retry the wsen:Pull request again until a different fault is returned. |
| Applicability | All requests |
| Remedy | The client can retry the operation. If the operation was a write (delete, create, or custom operation), the client can consult the system operation log before blindly attempting a retry, or attempt a wxf:Get or other read operation to try to discover the result of the previous operation. |

4199 **Table 37 – wse:UnableToRenew**

| | |
|---|---|
| Fault Subcode | wse: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 | wse:Renew |
| Remedy | The client issues a new subscription. |

4200 **Table 38 – wse:UnsupportedExpirationType**

| | |
|---|---|
| Fault Subcode | wse: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.<br>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 | wse:Subscribe |
| Remedy | The client corrects the expiration to use a duration time. |

4201 **Table 39 – wsen:UnsupportedExpirationType**

| | |
|---|---|
| Fault Subcode | wsen: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).<br>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 | wsen:Enumerate |
| Remedy | The client corrects the expiration time or omits it and retries. |

4202 **Table 40 – 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 | &lt;s:Detail&gt;<br>&lt;wsman:FaultDetail&gt;<br>If possible, one of the following URI values<br>&lt;/wsman:FaultDetail&gt;<br>&lt;/s:Detail&gt;<br>Possible URI values:<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AsynchronousRequest<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Bookmarks<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DeliveryRetries<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EnumerationMode<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAccess<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize<br>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime<br>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. |

4203

# 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.

**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 | http://schemas.xmlsoap.org/ws/2004/08/addressing | *WS-Addressing* |
| wse | http://schemas.xmlsoap.org/ws/2004/08/eventing | *WS-Eventing* |
| wsen | http://schemas.xmlsoap.org/ws/2004/09/enumeration | *WS-Enumeration* |
| wxf | http://schemas.xmlsoap.org/ws/2004/09/transfer | *WS-Transfer* |
| wsp | http://schemas.xmlsoap.org/ws/2004/09/policy | *WS-Policy* |
| wst | http://schemas.xmlsoap.org/ws/2005/02/trust | *WS-Trust* |
| wsse | http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd | *WS-Security* |
| wsu | http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd | *WS-Security* |

# 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.

For interoperability, the standard transports are HTTP 1.1 (RFC 2616) and HTTPS (using TLS 1.0) (RFC 2818).

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 (TLS 1.0), the service shall at least implement TLS_RSA_WITH_RC4_128_SHA. It is recommended that the service also support TLS_RSA_WITH_AES_128_CBC_SHA.

**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 will have 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 used in the TLS 1.0 handshake. |
| 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 a TLS 1.0 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 2478 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 TLS layer prior to beginning the Kerberos authentication sequence (Table C-8). See RFC 2478 for details.

**Table C-8 – SPNEGO Authentication over HTTPS with Cilent 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 2478 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 section 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. For example, the wsman security profile http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic (using HTTPS) is equivalent to simple http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic (using HTTP) if the traffic is actually secured by IPSec.

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 Transfer (7.7), WS-Enumeration (8), and WS-Eventing filters (10.2.2). Because the full *XPath 1.0* specification is large, subsets are typically required in resource-constrained implementations.

The purpose of this section is to identify the minimum set of syntactic elements that implementations can provide to promote maximum interoperability. In most cases, implementations will provide large subsets of full XPath, but 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 Transfer as described in 7.7), and Level 2 with basic filtering support (for WS-Enumeration and WS-Eventing). 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 for WS-Enumeration in 8.4 of this specification, and in WS-Eventing subscription filters in 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 Transfer (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:

```
<Envelope>
  <Body>
    <a>
      <b x="y"> 100 </b>
      <c>
        <d> 200 </d>
      </c>
      <c>
        <d> 300 </d>
        <d> 400 </d>
      </c>
    </a>
  </Body>
</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. Note that 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 note that text() cannot be suffixed to attribute selection.

A service that supports Fragment Transfer 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 in WS-Enumeration and subscription filters using WS-Eventing. 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)  <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: 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 wsen:Enumerate or wse: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 Enumerate and Subscribe 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 a wsen:Enumerate request is as follows:

```
<s:Header>
  <wsa:To> Service transport address </wsa:To>
  <wsman:ResourceURI> Resource URI </wsman:ResourceURI>
  ...
</s:Header>
<s:Body>
  <wsen: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>
    ...
  </wsen: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 wsen: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 5.1.2), 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 5.1.2, the service should fault with wsen:CannotProcessFilter (or wsman:CannotProcessFilter for Subscribe) containing the appropriate detail code.

# ANNEX F
(informative)

# WS-Management XSD

A normative copy of the XML schemas (XML Schema 1, XML Schema 2) for this specification can be retrieved by resolving the XML namespace URIs for this specification (listed in A.1).

## ANNEX G
(informative)

## Acknowledgements

The authors wish to acknowledge the following people.

Chair:

- Josh Cohen – Microsoft

Editors:

- Raymond McCollum – Microsoft
- Bryan Murray – HP
- Brian Reistad – Microsoft

Authors:

- Akhil Arora – Sun Microsystems
- Jim Davis – WBEM Solutions
- Mike Dutch – Symantec
- Zulah Eckert – BEA Systems
- Eugene Golovinsky – BMC Software
- Yasuhiro Hagiwara – NEC
- Jackson He – Intel
- David Hines – Intel
- Reiji Inohara – NEC
- Christane Kämpfe – Fujitsu-Siemens Computers
- Vincent Kowalski – BMC Software
- Vishwa Kumbalimutt – Microsoft
- Richard Landau – Dell
- James Martin – Intel
- Milan Milenkovic – Intel
- Paul Montgomery – AMD
- Alexander Nosov – Microsoft
- Abhay Padlia – Novell
- Roger Reich – Symantec
- Larry Russon – Novell
- Jeffrey Schlimmer – Microsoft
- Dr. Hemal Shah – Broadcom
- Sharon Smith – Intel
- Enoch Suen – Dell
- Vijay Tewari – Intel
- William Vambenepe – HP
- 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