Document number: DSP0226

Date: 2006-04-05

Version: 1.0.0a

Web Services for Management (WS-Management)

Document type: Specification

Document status: Preliminary

Document language: E

Contents

1 Introduction *1*
1.1 Requirements **1**
1.2 Notations and Terminology **1**
1.3 Notational Conventions **1**
1.4 Conformance **2**
1.5 XML Namespaces **2**
1.6 Terminology **3**
2 Addressing *4*
2.1 Endpoint References **4**
2.1.1 WS-Management Default Addressing Model 4
2.1.2 Selectors 10
2.1.3 Faults for Default Addressing Format 14
2.1.4 Service-Specific Endpoint References 14
2.2 Other WS-Addressing Headers **14**
2.3 mustUnderstand Usage **15**
2.4 wsa:To **16**
2.5 wsa:ReplyTo **17**
2.6 wsa:FaultTo **18**
2.7 wsa:MessageID and wsa:RelatesTo **19**
2.8 wsa:Action **20**
2.9 wsa:From **23**
3 WS-Management Control Headers *23*
3.1 Operation Timeout **23**
3.2 Maximum Envelope Size **24**
3.3 Locale **25**
3.4 Options **26**
3.5 Returned EPR **29**
4 Resource Access *30*
4.1 Introduction **30**
4.2 WS-Transfer **30**
4.3 Addressing Uniformity **32**
4.4 WS-Transfer:Get **33**
4.5 WS-Transfer:Delete **33**
4.6 WS-Transfer:Create **34**

71 **4.7 WS-Transfer:Put 36**
72 **4.8 Fragment-Level WS-Transfer 38**
73 **4.9 Fragment-Level WS-Transfer:Get 40**
74 **4.10 Fragment-Level WS-Transfer:Put 42**
75 **4.11 Fragment-Level WS-Transfer:Create 44**
76 **4.12 Fragment-Level WS-Transfer:Delete 46**
77 ***5 WS-Enumeration 47***
78 **5.1 Introduction 47**
79 **5.2 WS-Enumeration:Enumerate 48**
80 5.2.1 General 48
81 5.2.2 Enumeration "Count" Option 49
82 5.2.3 Optimization For Enumerations With Small Result Sets 50
83 **5.3 Filter Interpretation 53**
84 **5.4 WS-Enumeration:Pull 55**
85 **5.3 WS-Enumeration:Release 57**
86 **5.4 Ad-Hoc Queries and Fragment-Level Enumerations 58**
87 **5.5 Enumeration of EPRs 58**
88 ***6 Custom Actions (Methods) 60***
89 **6.1 General 60**
90 ***7 Eventing 61***
91 **7.1 General 61**
92 **7.2 Subscribe 61**
93 7.2.1 General 61
94 7.2.2 Filtering 63
95 7.2.3 Connection Retries 65
96 7.2.4 wse:SubscribeResponse 66
97 7.2.5 Heartbeats 66
98 7.2.6 Bookmarks 68
99 7.2.7 Delivery Modes 70
100 7.2.8 Event Action URI 71
101 7.2.9 Delivery Sequencing and Acknowledgement 71
102 7.2.10 Push Mode 72
103 7.2.11 PushWithAck Mode 73
104 7.2.12 Batched Delivery Mode 73
105 7.2.13 Pull Delivery Mode 77
106 **7.3 GetStatus 78**
107 **7.4 Unsubscribe 78**
108 **7.5 Renew 79**
109 **7.6 SubscriptionEnd 79**
110 **7.7 Acknowledgement of Delivery 80**
111 **7.8 Refusal of Delivery 81**

112 7.9 Dropped Events 81
113 *8 Metadata and Discovery* *83*
114 *9 Security* *85*
115 9.1 Introduction 85
116 9.2 Security Profiles 86
117 9.3 Security Considerations for Event Subscriptions 86
118 9.4 Including Credentials with a Subscription 87
119 9.5 Correlation of Events with Subscription 90
120 9.6 Transport-Level Authentication Failure 91
121 9.7 Security Implications of Third-Party Subscriptions 91
122 *10 Transports and Message Encoding* *91*
123 10.1 Introduction 91
124 10.2 SOAP 91
125 10.3 Lack of Response 93
126 10.4 Replay of Messages 93
127 10.5 Encoding Limits 93
128 10.6 Binary Attachments 94
129 10.7 Case-Sensitivity 94
130 *11 Faults* *94*
131 11.1 Introduction 94
132 11.2 Fault Encoding 95
133 11.3 NotUnderstood Faults 96
134 11.4 Degenerate Faults 97
135 11.5 Fault Extensibility 97
136 11.6 Master Fault Table 98
137 11.6.1 wsman:AccessDenied 98
138 11.6.2 wsa:ActionNotSupported 98
139 11.6.3 wsman:AlreadyExists 99
140 11.6.4 wsen:CannotProcessFilter 99
141 11.6.5 wsman:Concurrency 100
142 11.6.6 wse:DeliveryModeRequestedUnavailable 100
143 11.6.7 wsman:DeliveryRefused 101
144 11.6.8 wsa:DestinationUnreachable 101
145 11.6.9 wsman:EncodingLimit 102
146 11.6.10 wsa:EndpointUnavailable 103
147 11.6.11 wse:EventSourceUnableToProcess 104
148 11.6.12 wsen:FilterDialectRequestedUnavailable 104
149 11.6.13 wse:FilteringNotSupported 105
150 11.6.14 wsen:FilteringNotSupported 105
151 11.6.15 wse:FilteringRequestedUnavailable 106

11.6.16 wsman:InternalError 106
11.6.17 wsman:FragmentDialectNotSupported 107
11.6.18 wsman:InvalidBookmark 107
11.6.19 wsen:InvalidEnumerationContext 108
11.6.20 wse:InvalidExpirationTime 108
11.6.21 wsen:InvalidExpirationTime 109
11.6.22 wse:InvalidMessage 109
11.6.23 wsa:InvalidMessageInformationHeader 109
11.6.24 wsman:InvalidOptions 110
11.6.25 wsman:InvalidParameter 110
11.6.26 wxf:InvalidRepresentation 111
11.6.27 wsman:InvalidSelectors 112
11.6.28 wsa:MessageInformationHeaderRequired 112
11.6.29 wsman:NoAck 113
11.6.30 wsman:QuotaLimit 113
11.6.31 wsman:SchemaValidationError 113
11.6.32 wsen:TimedOut 114
11.6.33 wsman:TimedOut 114
11.6.34 wse:UnableToRenew 115
11.6.35 wse:UnsupportedExpirationType 115
11.6.36 wsen:UnsupportedExpirationType 115
11.6.37 wsman:UnsupportedFeature 116

12 Appendix A: HTTP(S) Transport and Security Profile 117

12.1 Introduction 117

12.2 HTTP(S) Encoding 117

12.3 IPSec and HTTP 119

12.4 HTTP(S) Security Profiles 119

12.5 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic 119

12.6 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest 120

12.7 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/basic 120

12.8 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/digest 121

12.9 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual 121

12.10 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic 122

12.11 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/digest 123

12.12 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego-kerberos 123

12.13 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/spnego-kerberos 123

12.14 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/spnego-kerberos 124

13 Appendix B: XPath Support 124

13.1 Level 1 125

13.2 Level 2 127

14 Appendix C: WS-Management XSD 129

15 Appendix D: Authors and Acknowledgements 129

195 **15.1 Additional Contributors** **129**
196 ***16 Appendix E: References*** ***130***
197

Abstract

This specification describes a general SOAP-based protocol for managing systems such as PCs, servers, devices, Web services and other applications, and other manageable entities.

Authors

Akhil Arora, Sun Microsystems, Inc.
Josh Cohen, Microsoft Corporation (Chair)
Jim Davis, WBEM Solutions, Inc.
Mike Dutch, Symantec Corporation
Zulah Eckert, BEA Software
Eugene Golovinsky, BMC Software, Inc.
Yasuhiro Hagiwara, NEC Corporation
Jackson He, Intel Corporation
David Hines, Intel Corporation
Reiji Inohara, NEC Corporation
Christane Kämpfe, Fujitsu-Siemens Computers
Vincent Kowalski, BMC Software, Inc.
Vishwa Kumbalimutt, Microsoft Corporation
Richard Landau, Dell Inc.
James Martin, Intel Corporation
Raymond McCollum, Microsoft Corporation (Editor)
Milan Milenkovic, Intel Corporation
Paul Montgomery, Advanced Micro Devices, Inc.
Bryan Murray, Hewlett-Packard (Editor)
Alexander Nosov, Microsoft Corporation
Abhay Padlia, Novell, Inc.
Roger Reich, Symantec Corporation
Brian Reistad, Microsoft (Editor)
Larry Russon, Novell, Inc.
Jeffrey Schlimmer, Microsoft Corporation
Dr. Hemal Shah, Broadcom
Sharon Smith, Intel Corporation
Enoch Suen, Dell Inc.
Vijay Tewari, Intel Corporation
William Vambenepe, Hewlett-Packard
Kirk Wilson, CA, Inc.
Dr. Jerry Xie, Intel Corporation

Status

The first edition of this specification, published in October 2004, was a pre-release version for public comment. The February 2005 edition was the basis of successful interoperation tests with multiple vendors. The June 2005 edition is based on the corrections and feedback from those tests. This version is a DMTF submission in progress.

1 Introduction

The Web services architecture is based on a suite of specifications that define rich functions and that may be composed to meet varied service requirements.

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 comprises the abilities to

- 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, and
- 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 may also choose not to support one or more areas of functionality listed above if that functionality is not appropriate to the target device or system.

1.1 Requirements

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 service architecture.

1.2 Notations and Terminology

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

1.3 Notational Conventions

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

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

- The syntax appears as an XML instance, but values in italics indicate data types instead of values.
- Characters are appended to elements and attributes to indicate cardinality:
  - "?" (0 or 1)
  - "*" (0 or more)
  - "+" (1 or more)
- The character "|" 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/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. If a receiver does not recognize an extension, the receiver SHOULD NOT process the message and MAY fault.
- XML namespace prefixes (see Table 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 should accept and strip leading and trailing whitespace within element values as if whitespace had not been used. (See conformance rule R10.3-9.)

1.4 Conformance

An implementation is not conformant with this specification if it fails to satisfy one or more of the MUST 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:

R1.4-1: To be conformant, the service MUST comply with all the rules defined in this specification. Items marked with MUST 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 should not be affected if they are ignored.

R1.4-2: A SOAP node MUST NOT use the XML namespace identifier for this specification (see section 1.5) unless it complies with the conformance rules in this specification.

This specification does not mandate that all messages and operations must 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 inconsistent with the rules defined in this specification.

1.5 XML Namespaces

R1.5-1: Conformant services of this specification MUST use this XML namespace Universal Resource Identifier (URI):

```
(1) http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
```

Table 1 lists XML namespaces used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

311 **Table 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 [SOAP 1.2]
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]
wsdl	http://schemas.xmlsoap.org/wsdl	WSDL/1.1 [WSDL 1.1]
wsa	http://schemas.xmlsoap.org/ws/2004/08/addressing	WS-Addressing [WS-Addressing]
wse	http://schemas.xmlsoap.org/ws/2004/08/eventing	WS-Eventing [WS-Eventing]
wsen	http://schemas.xmlsoap.org/ws/2004/09/enumeration	WS-Enumeration [WS-Enumeration]
wxf	http://schemas.xmlsoap.org/ws/2004/09/transfer	WS-Transfer [WS-Transfer]
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	WS-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 Utility

312 ## 1.6 Terminology

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

314 **Client**

315 The client application using the Web services defined in this document to access the management
316 service.

317 **Service**

318 An application that provides management services to clients by exposing the Web services defined in this
319 document. Typically, a service is equivalent to the network "listener," is associated with a physical
320 transport address, and is essentially a type of manageability access point.

321 **Managed Resource**

322 A Managed Resource is an entity that can be of interest to an administrator. It may be a physical object
323 such as a laptop computer or a printer or an abstract entity such as a service.

324 **Resource Class**

325 A Resource Class is an abstract representation (type) of a managed resource. It defines the
326 representation of management-related operations and properties. An example of a resource class is the
327 description of operations and properties for a set of laptop computers.

328 **Resource Instance**

329 A Resource Instance is an instantiation of a resource class. An example is the set of management-
330 related operations and property values for a specific laptop computer

Selector
A resource-relative name and value pair which acts as an instance-level discriminant when used with the default WS-Management endpoint reference format. This is essentially a filter or "key" which identifies the desired instance of the resource. This may not be present when service-specific endpoint reference formats 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 format.

2 Addressing

2.1 Endpoint References

WS-Management uses WS-Addressing endpoint references (also known as EPRs) as the addressing model for individual resource instances. WS-Management also defines a default endpoint reference format for use in addressing resources. In cases where this default addressing model is not appropriate, such as systems with well-established EPR formats or with opaque EPRs retrieved from a discovery service, services may use those service-specific addressing models, as long as they are based on WS-Addressing.

R2.1-1: All messages that are addressed to a resource class or instance that are referenced by an EPR MUST follow the rules for representing content from the EPR (the address and reference parameters) in the SOAP message. This also applies to continuation messages such as wsen:Pull or wsen:Release, which continue an operation begun in a previous message. Even though there is contextual information in such messages binding 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.

This rule clarifies that messages such as wsen:Pull or wse:Renew still require a full EPR. For wsen:Pull, for example, this would be the same as the original wsen:Enumerate, even though wsen:EnumerateResponse returns a context object which 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.

2.1.1 WS-Management Default Addressing Model

WS-Management defines a default format for endpoint references to resources. A service is not required to use this format for endpoint references, 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 format which contain its component parts, the ResourceURI and SelectorSet SOAP headers.

Description and use of this format in this specification by no means indicates that support for this format of endpoint reference is a requirement for a conformant service.

All of the normative text, examples, and conformance rules in section 2.1.1 and 2.1.2 presume that the service is based on the default addressing format. In cases where this format is not in use, then these rules do not apply.

The default endpoint reference format uses a representation which is a tuple of the following SOAP headers:

1) **wsa:To** (required): the transport address of the service

2) **wsman:ResourceURI** (required if the default addressing format is used): the URI of the resource class representation or instance representation.

3) **wsman:SelectorSet** (optional): Identifies or "selects" the resource instance to be accessed if more than once instance of a resource class exists.

The wsman:ResourceURI value should be marked with an s:mustUnderstand attribute set to "true" in all messages which make use of the default address format. Otherwise, a service which does not understand this format may inadvertently return a resource which was not requested by the client.

The WS-Management default format for endpoint references is defined in SOAP as follows:

```
(1)   <wsa:EndpointReference>
(2)      <wsa:Address>
(3)        Network address
(4)      </wsa:Address>
(5)      <wsa:ReferenceParameters>
(6)        <wsman:ResourceURI> resource URI </wsman:ResourceURI>
(7)        <wsman:SelectorSet>
(8)          <wsman:Selector Name="selector-name"> *
(9)          Selector-value
(10)      </wsman:Selector>
(11)     </wsman:SelectorSet> ?
(12) </wsa:ReferenceParameters>
(13)</wsa:EndpointReference>
(14)  ...                                                    ...
```

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

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 represents the resource class, but may represent the instance. Both 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 section 2.1.2. These are used to select an instance if the ResourceURI represents a multi-instanced target.

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

When it is time to actually use the above address in a real SOAP message, WS-Addressing specifies that translations take place and the headers are flattened out. While this is described in WS-Addressing, it is worth repeating because of its critical nature.

As an example, consider the following endpoint reference definition:

```
<wsa:EndpointReference>
  <wsa:Address> Address </wsa:Address>
    <wsa:ReferenceProperties>
      <other:UserProp>prop-value</other:UserProp>
    </wsa:ReferenceProperties>
    <wsa:ReferenceParameters>
     <wsman:ResourceURI>resURI</wsman:ResourceURI>
     <wsman:SelectorSet>
        <wsman:Selector Name="Selector-name">
            Selector-value
        </wsman:Selector>
     </wsman:SelectorSet>
     <other:UserParam> param </otherUserParam>
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

This address definition becomes the following when actually used in a SOAP message, in which 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>
  <other:UserProp>prop-value</other:UserProp>
  <wsman:ResourceURI>resURI</wsman:ResourceURI>
  <wsman:SelectorSet>
   <wsman:Selector Name="Selector-name">
      Selector-value
   </wsman:Selector>
  </wsman:SelectorSet>
  <other:UserParam> param </otherUserParam>"
  ...
```

Note also that in addition to the WS-Management-defined values, the user may additionally specify client-specific reference parameters (see *other:UserParam* above) which also are included in the message if they are part of the wsa:EndpointReference.

Note that the wsa:To, wsman:ResourceURI, and wsman:SelectorSet 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.

Here is an example of WS-Addressing headers based on the default format in an actual 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://123.99.222.36/wsman</wsa:To>
```

```
(8)  <wsman:ResourceURI>http://acme.org/hardware/2005/02/storage/physDisk</wsman:ResourceURI>
(9)  <wsman:SelectorSet>
(10)   <wsman:Selector Name="LUN"> 2 </wsman:Selector>
(11)  </wsman:SelectorSet>
(12)  <wsa:Action> http://schemas.xmlsoap.org/ws/2004/09/transfer/Get </wsa:Action>
(13)  <wsa:MessageID>   urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91 </wsa:MessageID>
(14)    ...
(15)</s:Header>
(16)<s:Body> ...
(17)
```

The following definitions apply to the above 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 is not required.

R2.1.1-1: 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. While 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 work together to define the actual resource being targeted:

```
(18)<s:Header>
(19)  <wsa:To> http://123.15.166.67/wsman </wsa:To>
(20)  <wsman:ResourceURI> http//schemas.acme.org/2005/02/hardware/physDisk </wsman:ResourceURI>
(21)   ...
```

It is considered a good practice for vendor-specific or organization-specific URIs to contain the internet domain name in the first token sequence after the scheme, such as "acme.org" above.

R2.1.1-2: The wsman:ResourceURI reference parameter is REQUIRED in messages

509 with the following wsa:Action URIs:
510

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

511 Note that the following messages require the EPR to be returned in the
512 wse:SubscriptionManager element of the wse:SubscribeResponse
513 message (WS-Eventing), so the format of the EPR is determined by the
514 service and may or may not include the ResourceURI:
515

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

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

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

516

517 While the ResourceURI is required, it may be short and of a very simple form, such as

518 *http://sample.com/**

519 *http://sample.com/resource*

520 *...etc.*

521
522
523 R2.1.1-3: For the request message of custom actions (methods), the ResourceURI header MAY be
524 present in the message to help route the message to the correct handler.
525
526 R2.1.1-4: The ResourceURI SHOULD NOT appear in other messages, such as responses or
527 events.
528

529 In practice, the wsman:ResourceURI is only required in requests to reference the targeted resource class.
530 Responses are not accessing the WS-Management or SOAP space, so the wsman:ResourceURI has no
531 meaning in that context.

532
533 R2.1.1-5: If the wsman:ResourceURI is missing, the service MUST issue a
534 wsa:DestinationUnreachable fault with a detail code of
535 http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI.
536

R2.1.1-6: The wsman:ResourceURI MUST only be used to indicate the identity of a resource, but MAY NOT be used to indicate the action being applied to that resource, which is properly expressed using the wsa:Action URI.

R2.1.1-7: The ResourceURI MUST be unique and unambiguous within the scope of a service.

Otherwise, the service has no idea which resource is actually being referenced or accessed.

Note that custom-WSDL based methods should have both a ResourceURI identity from the perspective of addressing, and have a wsa:Action from the point of view of execution. In many cases, the ResourceURI is simply a pseudonym for the WSDL identity and Port, and the wsa:Action is the specific method within that Port (or Interface) definition.

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

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

See also the recommendations in section 4.3 regarding addressing uniformity.

R2.1.1-8: If the wsman:ResourceURI is of the incorrect form or absent, the service SHOULD return a wsa:DestinationUnreachable fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI.

Note that custom actions have two distinct identities: The ResourceURI which can identify the WSDL and Port (or Interface), and the wsa:Action which identifies the specific method. If there is only one method in the interface, in a sense the ResourceURI and wsa:Action are identical.

It is not an error to utilize the wsa:Action URI for the ResourceURI of a custom method, but both will still be required in the message for uniform processing on both clients and servers. For example, the following action to reset a network card may 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://acme.com/2005/02/networkcards/reset </wsman:ResourceURI>
(6)     <wsa:Action>
(7)       http://acme.com/2005/02/networkcards/reset
(8)     </wsa:Action>
(9)     ...
```

In many cases, the ResourceURI will be equivalent to a WSDL name and port, and the wsa:Action URI contain an additional token as a suffix:

```
(10)<s:Header>
```

```
(11) <wsa:To>
(12)   http://1.2.3.4/wsman
(13) </wsa:To>
(14) <wsman:ResourceURI>http://acme.com/2005/02/networkcards</wsman:ResourceURI>
(15) <wsa:Action>
(16)       http://acme.com/2005/02/networkcards/reset
(17) </wsa:Action>
(18)     ...
```

And the ResourceURI may be completely unrelated to the wsa:Action:

```
(19) <s:Header>
(20)  <wsa:To>
(21)   http://1.2.3.4/wsman
(22)  </wsa:To>
(23) <wsman:ResourceURI>http://acme.com/products/management/networkcards</wsman:ResourceURI>
(24)  <wsa:Action>
(25)       http://acme.com/2005/02/netcards/reset
(26)  </wsa:Action>
```

All of these are legal usage.

As with other WS-Management operations, the endpoint reference 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 may not be used as part of the endpoint reference.

2.1.2 Selectors

In the WS-Management default addressing format, selectors are optional elements used to reference instances within a resource class. For operations such as wxf:Get or wxf:Put, the selectors are used to reference a single instance of the resource class referenced by the ResourceURI. For operations such as wsen:Enumerate, the selectors may be used to identify a subset of instances within 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 reference a specific 'row' or 'instance'. If there is only one instance of a resource class implied by the *ResourceURI,* the SelectorSet may be omitted, as 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 to be interpreted as being separated by implied logical AND operators.

The Selectors act as a 'key' mechanism against the resource class space implied by the ResourceURI. However, there is no implication that the Selector values are part of the returned resource or that Selectors be unique across instances, only that the set of all Selectors in a given message results in a reference to a set of instances of the appropriate cardinality for the operation. A SelectorSet used with wxf:Get must result in a reference to a single instance, while a SelectorSet used with wsen:Enumerate may result in a set of multiple instances." This is critical for small footprint implementations that cannot afford a full XPath processor.

Similarly, Selectors may be used in other operations such as wse:Subscribe to scope the domain of emitted events. See sections 5.3 and 7.2.2 for more information on using Selectors in these operations.

Note that in some information domains, the values referenced by the Selectors are "keys" which 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 reference the resource, but are not part of the representation.

R2.1.2-1: 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, as required to identify the precise instance of the resource class. Selector names and values MAY be treated case-insensitively or case-sensitively by the service (see 10.8), as required by the underlying execution environment.

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

R2.1.2-2: All content within the SelectorSet element is to be treated as a single reference parameter with a scope relative to the ResourceURI.

R2.1.2-3: A service using the WS-Management default addressing model MUST examine all Selectors in the message and process them as if they were logically ANDed. If the set of Selectors is incorrect for the targeted resource instance, then a wsman:InvalidSelectors fault SHOULD be returned to the client with the following detail codes:

a) http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelectors if Selectors are missing

b) http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch if Selector values are the wrong types

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

d) http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelectors if the Name is not a recognized Selector name.

R2.1.2-4: The Selector Name attribute MUST NOT be duplicated at the same level of nesting. The service SHOULD return a wsman:InvalidSelectors fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DuplicateSelectors if this occurs.

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. However, most information domains will benefit from the separation of type and instance identities into separate addressing elements.

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)  ...
```

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

wsa:To
: Network address and ResourceURI suffix

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, then there is one Selector element 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 endpoint reference. In the example below, the Selector on line 23 is a part of a SelectorSet consisting of nested EPR (lines 24-32) with its own Address+ResourceURI and SelectorSet elements:

```
(15)<s:Envelope
(16)    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(17)    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(18)    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(19)  <s:Header>
(20)    ...
(21)    <wsman:SelectorSet>
(22)      <wsman:Selector Name="Primary"> 123 </wsman:Selector>
(23)      <wsman:Selector Name="EPR">
(24)         <wsa:EndpointReference>
(25)            <wsa:Address> address </wsa:Address>
(26)               <wsa:ReferenceParameters>
(27)                 <wsman:ResourceURI> resource URI </wsman:ResourceURI>
(28)                 <wsman:SelectorSet>
(29)                   <wsman:Selector Name="name"> value </wsman:Selector>
(30)                 </wsman:SelectorSet>
(31)               </wsa:ReferenceParameters>
(32)          </wsa:EndpointReference>
(33)      </wsman:Selector>
(34)    </wsman:SelectorSet>
(35)   ...
(36)
```

718
719 R2.1.2-5: For those services using the WS-Management default addressing model, the value of a
720 wsman:Selector MUST be one of
721 (a) A simple type as defined in the XML schema namespace
722 **http://www.w3.org/2001/XMLSchema**
723 (b) A nested wsa:EndpointReference using the WS-Management addressing model.
724 A service MAY fault Selector usage with wsman:InvalidSelectors if the Selector is not a simple type
725 or of a supported schema.
726
727 R2.1.2-6: A conformant service MAY reject any Selector or nested Selector with a nested endpoint
728 reference whose wsa:Address value is not the same as the primary wsa:To value or is not
729 **http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous**.

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

732 WS-Management SelectorSet values may be used in conjunction with enumerations to specify a scope
733 on the enumeration.

734 In some cases, the wsman:ResourceURI refers to a domain or class of items, and the wsman:Selector
735 values act as a scoping mechanism. For example, the ResourceURI may be used to enumerate
736 "Principals", and an additional "Selector" might scope the enumeration to a specific Kerberos realm,
737 allowing only "Principals" within the specified Realm to be the enumerable set before any filters are
738 applied.

739 Selectors may be used as a type of filtering mechanism per se without using a filter dialect. This is
740 permitted as long as the logical restriction of Selector values is observed: They are simple tests for
741 equality and if more than one selector is present, they are logically ANDed.

742 For instance consider the following examples for receiving a subset of items in an enumeration sequence,
743 the XPath

744
745
```
(1)    .[BackupDrive='true']
```

746 ...and the following SelectorSet would provide the same enumeration result set:

747
748
```
(2)   <wsman:SelectorSet>
(3)      <wsman:Selector Name="BackupDrive">true</wsman:Selector>
(4)   </wsman:SelectorSet>
```

751 In the first case, the XPath filters out all drives which are not "backup drives", and in the second case, the
752 "Selector" forces "BackupDrive" to be "true" as part of the selection criteria.

753 XPath and other filter dialects are typically much more powerful than using the wsman:SelectorSet for
754 filtering, but resource-constrained implementations may be able to benefit from using this simple
755 mechanism, since it requires no filter language processor and uses SOAP processing which is already
756 present for other WS-Management operations.

757
758 R2.1.2-7: A service MAY support the wsman:SelectorSet element and wsman:Selector values to
759 scope the events emitted by an event source, either alone or in conjunction with a wsman:Filter
760 element.

In some cases, the wsman:ResourceURI refers to a domain or class of items, and the wsman:Selector values can act as a scoping mechanism. For example, the ResourceURI may be used to provide all "hardware failure" events, and the "Selector" may scope this to "Storage" devices or "Display" devices.

This suggests that Selectors may be used as a type of filtering mechanism per se without using a filter dialect. This is permitted as long as the logical restriction of Selector values is observed: They are simple tests for equality and if more than one selector is present, they are logically ANDed.

XPath and other filter dialects are typically much more powerful than using the wsman:SelectorSet for filtering, but resource-constrained implementations may be able to benefit from using this simple mechanism, since it requires no filter language processor and uses SOAP processing which is already present for other WS-Management operations.

R2.1.2-8: A service MAY fail to process a Selector of more than 2048 characters.

R2.1.2-9: 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 which contains more than 8096 characters of content in the root <SelectorSet> element.

2.1.3 Faults for Default Addressing Format

Whenever faults based on the default format are generated, they often will contain specific fault detail codes. These are clearly called out separately in the master fault table in 11.6 and do not apply when service-specific addressing is in use.

2.1.4 Service-Specific Endpoint References

Although WS-Management specifies a default addressing model, there are cases where this model is not available or appropriate.

R2.1.4-1: A conformant service MAY not understand the header values used by the WS-Management default endpoint reference format. Since the wsman:ResourceURI should be marked *mustUnderstand="true"* by the client, the service MUST return an s:NotUnderstood fault.

R2.1.4-2: A conformant service MAY require additional header values to be present which are beyond the scope of this specification.

Services may thus use alternative EPR formats for addressing resources with WS-Management. These formats may or may not use ResourceURI or SelectorSet, and still be valid addressing models providing they conform to the rules of WS-Addressing.

In addition to a defined alternative format, a service may not explicitly define any format at all and may use an opaque EPR generated at runtime, which is to be handled according to the standard rules of WS-Addressing.

When such addressing models are in use, the client application will have to understand and interoperate with discovery methods for acquiring these endpoint references that are beyond the scope of this specification.

2.2 Other WS-Addressing Headers

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

R2.2-1: A conformant service MUST 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 usage of these header blocks is discussed in subsequent sections.

2.3 mustUnderstand Usage

The SOAP mustUnderstand attribute for SOAP headers is to be interpreted as a "must comply" instruction in WS-Management. For example, if a SOAP header which 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 the service treats a header as optional, the mustUnderstand attribute should be omitted.

Obviously, if the service cannot understand the primary endpoint reference of the resource (the ResourceURI), it will not be able to service the request in any case. Similarly, if the wsa:Action is not understood, the implementation will 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

R2.3-1: A conformant service MUST process any of the above elements identically whether *mustUnderstand="true"* is present or not.

As a corollary, clients may omit *mustUnderstand="true"* from any of the above elements with no change in meaning. Obviously, the client may safely always include *mustUnderstand="true"* on any of the above elements.

R2.3-2: If a service cannot comply with a header marked with *mustUnderstand="true"*, it MUST issue a s:NotUnderstood fault.

The goal is that the service should be tolerant of inconsistent mustUnderstand usage by clients when there is no real chance of the request being misinterpreted.

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

2.4 wsa:To

In request messages, the wsa:To address contains the network address of the service. In some cases, this is sufficient to locate the resource. In other cases, the service is a dispatching agent for multiple resources. In these cases the endpoint reference will typically contain additional fields (reference parameters) to allow the service to identify a resource within its scope.

Note that 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.

For example, when the default address format is in use, these additional fields are the ResourceURI and SelectorSet fields.

R2.4-1: The wsa:To header MUST be present in all messages, whether requests, responses, or events, and SHOULD reflect a valid URI. In the absence of other requirements, it is RECOMMENDED that the network address be suffixed by the token sequence **/wsman**:

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

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

Including the network transport address in the SOAP message may seem redundant, since the network connection would already have to be established by the client, but in cases where the message is routed through intermediaries, this would obviously be required so that the intermediary could examine the message and make the final connection to the actual endpoint.

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

Note that wsa:To must be present in *all* messages, including replies and faults, even though it appears redundant for many transports.

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

R2.4-2: 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 a detail code of 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.

2.5 wsa:ReplyTo

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

R2.5-1: A wsa:ReplyTo header MUST be present in all request messages when a reply is required. This MUST either be 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 is missing, a wsa:MessageInformationHeaderRequired fault is returned.

Note that some messages, such as event deliveries, wse:SubscriptionEnd, etc., do not require a response and may omit a wsa:ReplyTo element.

R2.5-2: 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 R2.5-1 MUST indicate this. Otherwise, the service MUST return a wsman:UnsupportedFeature fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode.

R2.5-3: When delivering events for which acknowledgement of delivery is required, the sender of the event MUST include a wsa:ReplyTo element, and observe the usage in section 7.8 of this specification.

R2.5-4: The service MUST fully duplicate the entire wsa:Address of the wsa:ReplyTo element in the wsa:To of the reply, even if some of the information is not understood by the service.

This is used in cases where the client includes suffixes on the HTTP or HTTPS address that the service does not understand. The service must return these suffixes nonetheless.

R2.5-5: Any reference parameters supplied in the wsa:ReplyTo address MUST 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, then 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 ReplyTo headers. The WS-Addressing specification requires that these 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 section 2.1. This allows clients to better correlate responses with the original requests. This step cannot be omitted. In the example below, the "user-defined content" must be 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)          user-defined content
(14)     </wsa:ReferenceParameters>
(15) </wsa:ReplyTo>
(16) ...
```

R2.5-6: If the wsa:ReplyTo address is not usable or is missing, the service should not reply to the request, as it has no way to reply properly, 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.

2.6 wsa:FaultTo

R2.6-1: A conformant service is NOT REQUIRED to support a wsa:FaultTo address which is distinct from the WS-Addressing:ReplyTo address. If such a request is made and is not supported by the service, a wsman:UnsupportedFeature fault MUST be returned with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode.

If a wsa:ReplyTo is omitted from a request (a gross error), then transport-level mechanisms are typically used to fail the request, since where to send the fault is uncertain. It is not an error for the service to simply shut down the connection in this case.

R2.6-2: If wsa:FaultTo is omitted, the service MUST return the fault to the wsa:ReplyTo address if a fault occurs.

R2.6-3: 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 MUST be **http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous** (see WS-Addressing). If services do not support separately-addressed fault delivery and the wsa:FaultTo is any other address, a wsman:UnsupportedFeature fault MUST be returned with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode.

Note that this specification does not restrict richer implementations from fully supporting wsa:FaultTo.

R2.6-4: Any reference parameters supplied in the wsa:FaultTo address SHOULD be included as top-level headers in the actual fault, as specified in WS-Addressing. 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 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 should not be omitted except in cases where the resulting fault would be large enough to exceed encoding limit restrictions elsewhere in this specification.

In the following example, the "user-defined content" MUST appear in the fault, if it occurs:

```
(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)         user-defined content
(14)    </wsa:ReferenceParameters>
(15)</wsa:FaultTo>
(16)...
```

R2.6-5: If the wsa:FaultTo address is not usable, the service should not reply to the request, as it has no way to return the fault properly. Similarly, if no 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.

R2.6-6: The service MUST 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 occurs in cases where the client includes private content suffixes on the HTTP or HTTPS address that the service does not understand. The service should not remove this information when constructing the address.

2.7 wsa:MessageID and wsa:RelatesTo

R2.7-1: The MessageID and RelatesTo URIs MAY be of any format, as long as they are valid URIs according to RFC 3986. Because URIs may differ only by case, two URIs with the same characters are considered different, regardless of what the URI represents.

The following two formats are endorsed by this specification. The first is considered a best-practice, as it is backed by IETF RFC 4122 [23]:

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

In the examples above, each **x** is an uppercase or lowercase hexadecimal digit (lowercase required by RFC 4122); there are no spaces or other tokens. It is NOT REQUIRED that this format be used or that it formally be a DCE-style GUID with provable uniqueness properties, although this is typical.

Regardless of format, the URI may not exceed the maximum defined in

R10.6-1.

Because GUIDs have a numeric meaning as well as a string meaning, this can lead to confusion. The same GUID in lowercase is a different URI than the same GUID in uppercase. This is because the value is being implied rather than the URI representation, and WS-Management works with the URI value itself, not the underlying value representation. Services are free to *interpret* the URI in any way, but must treat the value as a URI and not as its underlying value, and may not alter the case usage when repeating the message or any of the MessageID values in subsequent messages.

Note that RFC 4122 requires the digits to be lowercase. This is the responsibility of the client, as the service simply processes the values as URI values and is not required to analyze the URI for correctness or compliance. Of course, the service must replicate the client usage in the RelatesTo reply, and may not alter the case usage.

R2.7-2: The MessageID SHOULD be generated according to any algorithm which ensures that no two MessageIDs will repeat. Since the value is treated as case-sensitive (R2.7-1), confusion can arise if the same value is reused differing only in case, so the service MUST NOT create or employ MessageID values which differ only in case. For any message transmitted by the service, the MessageID MUST NOT be reused.

The client must ensure that MessageID values are not reused in requests. While services and clients should not issue different MessageIDs which differ only by case, the service is not required to detect this, nor is it required to analyze the URI for syntactic correctness or repeated use.

R2.7-3: The RelatesTo MUST be present in all response messages and faults, MUST contain the MessageID of the associated request message, and MUST match the original in case, being treated as a URI value and not as a binary GUID value.

R2.7-4: If the MessageID is not parsable or is missing, a wsa:InvalidMessageInformationHeader fault SHOULD be returned.

Examples:

```
<wsa:MessageID>
  uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91
</wsa:MessageID>

<wsa:MessageID>
  anotherScheme:ID/12310/1231/16607/25
</wsa:MessageID>
```

Note that mustUnderstand can be omitted for either wsa:MessageID or wsa:RelatesTo with no change in meaning.

2.8 wsa:Action

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

R2.8-1: The wsa:Action URI MUST NOT be used to identify the specific resource class or instance, but only to identity the operation to use against that resource.

R2.8-2: For all resource endpoints, a service MUST 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 is what will contain the "Get". The wsa:To, and potentially other SOAP headers, will indicate *what* is being accessed. When the default addressing format is used, for example, the ResourceURI will typically contain the reference to the "Disk" and the SelectorSet identify which disk. Other service-specific addressing formats may factor the identity of the resource in different ways.

Implementations are free to support additional custom methods which combine the notion of "Get" and "Disk into a single "GetDisk" action, as long as 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.

R2.8-3: If a service exposes any of the following types of capabilities, a conformant service MUST at least expose that capability using the definitions in the following table according to the rules of this specification. The service MAY OPTIONALLY expose additional similar functionality using a distinct wsa:Action URI.

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 the above "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 above "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 the above "Pull"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew	Renews an enumerator which may have timed out (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse	Response to the "Renew" (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus	Gets the status of the enumerator (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse	Response to the "GetStatus"

	request (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 the above "Release"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd	Notifies that an enumerator has terminated (not required in WS-Management)
http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe	Models a subscription to an event source
http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse	Response to the above "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 the "Renew" request
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 the "GetStatus"message
http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe	Removes an active subscription
http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse	Response to the "Unsubscribe" operation
http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd	Delivers a message to indicate that a subscription has terminated
http://schemas.dmtf.org/wbem/wsman/1/wsman/Events	Delivers batched events based on a subscription
http://schemas.dmtf.org/wbem/wsman/1/wsman/Heartbeat	A pseudo-event that models a heartbeat of an active subscription. Delivered when no real events are available, but used to indicate that the event subscription and delivery mechanism is still active.
http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents	A pseudo-event which 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 which does not define its own action.

1088 R2.8-4: A custom action MAY BE supported if the operation is a custom method
1089 whose semantic meaning is not present in the table, or if the item is an

event.

R2.8-5: All event deliveries MUST contain a unique action URI which 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 section 6 of this specification.

2.9 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 field provides no useful information in practice, but can be useful in cases where the response arrives on a different connection.

R2.9-1: A conformant service MAY include a wsa:From address in the message, but is NOT REQUIRED to do so. A conformant service SHOULD process any incoming message which has a wsa:From element.

R2.9-2: A conformant service SHOULD NOT fault any message with a wsa:From element, whether or not mustUnderstand is included.

Note that processing the wsa:From message is trivial since it has no effect on the meaning of the message. The *From* address is primarily for auditing and logging purposes. This may occur in a message of any type, whether a request, reply, singleton message, or event.

3 WS-Management Control Headers

3.1 Operation Timeout

Most management operations are time-critical due to quality-of-service constraints and obligations. If they cannot be completed in a specified time, the service must return a fault so that a client can comply with its obligations.

The following header value may be supplied with any WS-Management message to indicate that the client expects a response or a fault within the specified time:

```
<wsman:OperationTimeout> xs:duration </wsman:OperationTimeout>
```

R3.1-1: 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 can be generated.

R3.1-2: 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, then a wsman:InvalidHeader fault SHOULD be generated with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidTimeout.

R3.1-3: If the service does not support user-defined timeouts, a wsman:UnsupportedFeature fault SHOULD be returned with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OperationTimeout.

R3.1-4: If the wsman:OperationTimeout is omitted, the service MAY interpret this as an instruction to block indefinitely until a response is available, or MAY impose a default timeout.

These rules do not preclude services from supporting infinite or very long timeouts. Given that network connections seldom will block indefinitely with no traffic occurring, some type of transport timeout is likely in any case. 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 should 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 should undo the effects of any partially complete operations if possible, but this is not always practical. In such cases, the service should keep a local log of requests and operations which the client can query later.

R3.1-5: If mustUnderstand is applied to the wsman:OperationTimeout, then the service MUST observe the requested value or return the fault specified in R3.1-2. The service SHOULD attempt to complete the request within the specified time or issue a fault without any further delay.

Clients should 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.

An example of a correctly formatted 30-second timeout appears as follows in the SOAP header:

```
<wsman:OperationTimeout>PT30S</wsman:OperationTimeout>
```

If the transport timeout occurs before the actual wsman:OperationTimeout, the operation should be treated as specified in section 10.4, the same as a failed connection. In practice, these should be configured so that the network transport timeout is larger than any expected wsman:OperationTimeout.

This specification establishes no requirement regarding the internal behavior of services regarding incomplete operations if a timeout occurs. 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 may elect to include additional information in the fault (see section 11.5) regarding its internal policy in this regard.

The service should attempt to return to the state that existed before the operation was attempted, but this is not always possible. It is expected that clients will attempt to discover what happened by doing subsequent wxf:Get operations or by querying internal logs kept by the service.

3.2 Maximum Envelope Size

To prevent a response beyond the capability of the client, the request message may 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 body does not exceed the specified number of octets:

```
<wsman:MaxEnvelopeSize> xs:positiveInteger </wsman:MaxEnvelopeSize>
```

The limitation is on the entire envelope, as resource-constrained implementations need a reliable figure for the required amount of memory for all SOAP processing, not just the envelope Body, which leaves the Header totally ambiguous.

R3.2-1: 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, a wsman:EncodingLimit fault SHOULD be returned with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize.

R3.2-2: If mustUnderstand is set to true, the service MUST comply with the request. If the response would exceed the maximum size, then a wsman:EncodingLimit fault SHOULD be returned. Since a service may execute the operation prior to knowing the response size, the service SHOULD undo any effects of the operation prior to issuing the fault. If the operation cannot be reversed (such as a destructive wxf:Put or wxf:Delete, or a wxf:Create), the service MUST indicate that the operation succeeded in the wsman:EncodingLimit fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess.

R3.2-3: If mustUnderstand is set to false, the service MAY ignore the header.

R3.2-4: The service 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 a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MininumEnvelopeLimit.

Note that if the service exceeds its own encoding limit (independently of what the client specifies), the fault is wsman:EncodingLimit with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit.

Note: The definition of the wsman:MaxEnvelopeSize element in the schema contains a Policy attribute since 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. Clients should not add the attribute to the wsman:MaxEnvelopeSize element when it is used as a SOAP header.

R3.2-5: Services SHOULD ignore the Policy attribute if it appears in the wsman:MaxEnvelopeSize element when used as a SOAP header.

3.3 Locale

Management operations often span locales, and many items in responses can require translation. Typically, this applies to descriptive information intended for human readers which is sent back in the response. If the client requires such output to be translated into a specific language, it may 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" />
```

R3.3-1: If mustUnderstand is omitted or set to false, the service SHOULD utilize this value when composing the response message and adjust any localizable values accordingly. This is the RECOMMENDED usage for most cases. The locale is treated as a "hint" in this case.

R3.3-2: If mustUnderstand is set to true, then the service MUST ensure that the replies contain localized information where appropriate, or else the service must issue a wsman:UnsupportedFeature fault with a detail code of

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale. A service MAY always fault if wsman:Locale contains s:mustUnderstand, since 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.

R3.3-3: The value of the xml:lang attribute in the wsman:Locale header must be a valid RFC 3066 language code.

R3.3-4: 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 may be omitted if no descriptive content appears in the body. Including it is not an error, even if no descriptive content occurs:

```
(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>
(7)    ...
```

The xml:lang attribute may 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.

R3.3-5: For operations which span multiple message sequences, the wsman:Locale element is processed in the initial message only. It SHOULD be ignored in subsequent messages, since 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 applies primarily to wsen:Enumerate and wsen:Pull messages. The locale is clearly established once during the initial wsen:Enumerate request, so changing the locale during the enumeration would serve no purpose. The service ignores any wsman:Locale elements in subsequent wsen:Pull messages, but the client should 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 R3.3-1) that the wsman:Locale element never contain a mustUnderstand attribute. In this way, the client will not receive faults in unexpected places.

3.4 Options

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.

R3.4-1: Any request message MAY contain a *wsman:OptionSet* header, which wraps a set of optional switches or controls on the message. These help the service compose the desired reply or observe the required side-effect.

R3.4-2: The service SHOULD NOT send responses, unacknowledged events, or singleton messages containing wsman:OptionSet headers unless it is acting in the role of a client to another service. Those headers are intended

for request messages alone to which a subsequent response is expected, including acknowledged events.

R3.4-3: If the **mustUnderstand** attribute is omitted from the OptionSet block, then the service MAY ignore the entire wsman:OptionSet block. If it is present and the service does not support wsman:OptionSet, the service MUST return a s:NotUnderstood fault.

Services must be able to process an OptionSet block if it is present, but they are not required to understand or process individual options, as shown in R3.4-6. However, if mustComply is used on any given Option, then mustUnderstand must be set to "true". This avoids the incongruity of allowing the entire OptionSet block to be ignored while having mustComply on individual options.

R3.4-4: Each resource class MAY observe its own set of options, and individual instances of that resource class MAY further observe its own set of options. There is no requirement to support consistent option usage 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.

R3.4-5: Any number of individual Option elements may appear under the wsman:OptionSet wrapper. Option names MAY be repeated if appropriate. The content MUST 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. Case usage MUST be retained because the OptionSet 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. This is because the value must usually be passed through to real-world subsystems which inconsistently expose case usage. Where interoperation is a concern, the client should omit both mustUnderstand and mustComply.

R3.4-6: Individual option values may be advisory or may be required by the client. The service must observe and execute any option marked with the MustComply attribute set to "true", or return a wsman:InvalidOptions fault with a detail code of 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 MAY be ignored by the service. If any option is marked with MustComply set to "true", then mustUnderstand MUST 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 or not, and can only pass it along to the next subsystem.

R3.4-7: 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 types declared in the **http://www.w3.org/2001/XMLSchema** namespace are supported in this version of WS-Management.

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

R3.4-8: Options SHOULD NOT be used as a replacement for the documented parameterization technique for the message, but SHOULD only be used 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.

R3.4-9: The following faults SHOULD be returned by the service:

- wsman:InvalidOptions with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported in cases where Options are not supported.
- wsman:InvalidOptions with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName in cases where one or more options names were not valid or not supported by the specific resource.
- wsman:InvalidOptions with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue in cases where the value was not correct for the Option name.

R3.4-10: For operations which span multiple message sequences, the wsman:OptionSet element is processed in the initial message only. It SHOULD be ignored in subsequent messages, since 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 applies primarily to wsen:Enumerate and wsen:Pull messages. The set of options is clearly established once during the initial wsen:Enumerate request, so changing the options during the enumeration would constitute an error.

Options are intended to help 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 should be recomputed and that cached values should not 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 should not contain XML, but are limited to xs:string values. If XML-based input is required, then a custom operation (method) with its own wsa:Action is the correct model for the operation. These rules are intended to ensure that no backdoor parameters over well-known message types are introduced. For example, when issuing a wse:Subscribe request, the message already defines a technique for passing an event filter to the service, so the options should not be used to circumvent this and pass a filter using an alternate method.

The following is an example of wsman:OptionSet:

```
(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:xs="http://www.w3.org/2001/XMLSchema">
(6)    <s:Header>
(7)       ...
(8)       <wsman:OptionSet s:mustUnderstand="true">
(9)         <wsman:Option Name="VerbosityLevel" Type="xs:int">
(10)          3
```

```
(11)         </wsman:Option>
(12)         <wsman:Option Name="LogAllRequests" MustComply ="true" />
(13)       </wsman:OptionSet>
(14)   ...
(15)
```

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

wsman:OptionSet

Used to wrap individual option blocks. In this example, s:mustUnderstand is set to "true", indicating that the overall block must be processed using the rules above and cannot be ignored.

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 should be a short non-colliding URI which is vendor-specific.

wsman:OptionSet/wsman:Option/@MustComply

If set to "true", the option must be observed, otherwise, it is advisory or a hint.

wsman:OptionSet/wsman:Option/@Type

Optional attribute. If present, this indicates the data type of the element content, which helps the service to interpret the content. A service may require this to be present on any given option element.

wsman:OptionSet/wsman:Option

The content of the option. This may be any simple string value. If the option value is empty, then it should be interpreted as logical "true", and the option is "enabled". The following example enables the "Verbose" option:

```
        <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.

3.5 Returned EPR

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

In order to provide a means for the client to determine when this has happened, two SOAP headers as 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 .../>
```

R3.5-1: A service receiving a message containing the wsman:RequestEPR header block SHOULD return a response which contains a wsman:RequestedEPR header block. This block will contain the most recent endpoint reference of the resource being accessed or a status code if it cannot. This

EPR will reflect any identity changes that may have occurred as a result of the current operation, as set forth in the behavior below.

The header block in the corresponding response message has the following format:

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

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

wsman:RequestedEPR/wsa:EndpointReference

This element is one of a choice of three elements that can be returned as a child element of the wsman:RequestedEPR element. The use of this choice indicates 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 or side-effects of the associated request message. 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

This element is one of a choice 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 MUST NOT use the old wsa:EndpointReference in subsequent operations.

wsman:RequestedEPR/wsman:EPRUnknown

This element is one of a choice 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 use the old wsa:EndpointReference in subsequent operations. The result of using an old wsa:EndpointReference is unpredictable; a result may be a fault or a successful response.

4 Resource Access

4.1 Introduction

Resource access applies to all synchronous operations regarding getting, setting, and enumerating values. The WS-Transfer 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 must be sent as a series of single messages.

4.2 WS-Transfer

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

Following is a full example of a hypothetical wxf:Get request and associated response:

```
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>
8         http://1.2.3.4/wsman/
9       </wsa:To>
10      <wsman:ResourceURI>http://samples.org/2005/02/physicalDisk</wsman:ResourceURI>
11      <wsa:ReplyTo>
12        <wsa:Address>
13          http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
14        </wsa:Address>
15      </wsa:ReplyTo>
16      <wsa:Action>
17        http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
18      </wsa:Action>
19      <wsa:MessageID>
20        uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
21      </wsa:MessageID>
22      <wsman:SelectorSet>
23        <wsman:Selector Name="LUN"> 2 </wsman:Selector>
24      </wsman:SelectorSet>
25      <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
26
27    </s:Header>
28    <s:Body/>
29  </s:Envelope>
```

Note that the wsa:ReplyTo occurs on the same connection as the request (line 8), the action is a wxf:Get (line 12), and the ResourceURI (line 8) and wsman:SelectorSet (line 21) are used to address the requested management information. This example assumes that the WS-Management default addressing format is in use. The service should expect to complete the operation in 30 seconds or return a fault to the client (line 24).

Also note that the s:Body has no content in a wxf:Get request.

A hypothetical response could be:

```
0   <s:Envelope
1       xmlns:s="http://www.w3.org/2003/05/soap-envelope"
2       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
3       xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
4     <s:Header>
5       <wsa:To>
6           http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
7       </wsa:To>
8       <wsa:Action s:mustUnderstand="true">
```

```
9          http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
10       </wsa:Action>
11       <wsa:MessageID s:mustUnderstand="true">
12          uuid:d9726315-bc91-430b-9ed8-ce5ffb858a88
13       </wsa:MessageID>
14       <wsa:RelatesTo>
15           uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
16       </wsa:RelatesTo>
17     </s:Header>
18     <s:Body>
19       <PhysicalDisk xmlns="http://schemas.acme.com/2005/02/samples/physDisk">
20         <Manufacturer> Acme, Inc. </Manufacturer>
21         <Model> 123-SCSI 42 GB Drive </Model>
22         <LUN> 2 </LUN>
23         <Cylinders> 16384 </Cylinders>
24         <Heads> 80 </Heads>
25         <Sectors> 63 </Sectors>
26         <OctetsPerSector> 512 </OctetsPerSector>
27         <BootPartition> 0 </BootPartition>
28       </PhysicalDisk>
29     </s:Body>
30   </s:Envelope>
```

Note that the response uses the wsa:To address (line 35) that the original request had specified in wsa:ReplyTo. Also, the wsa:MessageID for this response is unique (line 40). The wsa:RelatesTo (line 43) contains the uuid of the wsa:MessageID of the original request to allow the client to correlate the response.

The Body (lines 48-59) contains the requested resource representation.

The same general approach exists for wxf:Delete, except that no response occurs in the s:Body. The wxf:Create and wxf:Put operations are similar, except that there is an s:Body on input to specify the values being created or updated.

4.3 Addressing Uniformity

In general, the service should expose addressing usage which is identical for the operations. Wherever practical, the endpoint reference of the resource should be the same whether a wxf:Get, wxf:Delete, wxf:Put, or wxf:Create is being used. This is not a strict requirement, but reduces the education and training required to construct and use WS-Management-aware tools

It is clear that wxf:Create is a special case, in that the endpoint reference of the newly created resource is often not known until the resource is actually created. For example, while it may 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 would not have the same addressing headers as the corresponding wxf:Get or wxf:Delete (which would presumably terminate the process).

If the 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 must be remembered that the s:Body of the messages must contain XML with correct and valid XML namespaces referring to XML Schemas which can validate the message. While most services and clients will not do real-time validation of messages in production environments due to performance constraints, during debugging or other systems verification, validation may be enabled, and such messages will be considered invalid.

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

4.4 WS-Transfer:Get

The wxf:Get operation retrieves resource representations. The message may 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.

R4.4-1: A conformant service SHOULD support wxf:Get 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.

R4.4-2: Execution of the wxf:Get SHOULD NOT in itself have side-effects on the value of the resource.

R4.4-3: 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 fragments or chunks of XML which correspond to real-world objects. To retrieve individual property values, the client must either postprocess the XML content for the desired value, or the service can support Fragment-Level WS-Transfer (4.8).

Fault usage is generally as described in chapters 2 and 3. Inability to locate or access the resource is equivalent to problems with the SOAP message when the endpoint reference is defective. There are no "get-specific" faults.

4.5 WS-Transfer:Delete

The WS-Transfer:Delete operation deletes resource instances.

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

R4.5-1: A conformant service is NOT REQUIRED to support wxf:Delete.

R4.5-2: A conformant service SHOULD support wxf:Delete using the same endpoint reference as a corresponding wxf:Get or other messages, but this is NOT REQUIRED if the deletion mechanism for a resource is semantically distinct.

R4.5-3: 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.

R4.5-4: 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 service and is a *logical* deletion.

The operation may result in an actual deletion, such as removal of a row from a database table, or it may 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 removing it from the access scope of the service, or "unbinding" 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, the client must either do a wxf:Put with those properties removed, or the service can support Fragment-Level WS-Transfer (section 4.8).

Fault usage is generally as described in chapters 2 and 3. Inability to locate or access the resource is equivalent to problems with the SOAP message when the endpoint reference is defective. There are no "delete-specific" faults.

4.6 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 endpoint reference assigned to a newly created instance for subsequent access is not necessarily part of the XML content used for creating the resource. Since the endpoint reference may often be assigned by the service or one of its underlying systems, the CreateResponse must contain the applicable endpoint reference of the newly created instance.

R4.6-1: A conformant service is NOT REQUIRED to support wxf:Create.

R4.6-2: A conformant service is NOT REQUIRED to support wxf:Create using the same endpoint reference as a corresponding wxf:Get, wxf:Put, or other messages for that resource.

R4.6-3: 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.

R4.6-4: If the supplied 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:

- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues if one or more values in the <Body> was not correct
- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues if one or more values in the <Body> was missing
- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace if the wrong XML schema namespace was used and is not recognized by the service

R4.6-5: A service MUST not use wxf:Create to perform an update on an existing representation. The targeted object must not already exist, or else the service SHOULD return a wsman:AlreadyExists fault.

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

WS-Transfer specifies that wxf:CreateResponse must contain 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 which 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 endpoint reference of the newly created resource.

If a service needs to support creation of individual values within a representation (fragment-level creation, array insertion, etc.), then it should support Fragment-Level WS-Transfer (section 4.8).

R4.6-6: The wxf:CreateResponse to a wxf:Create message MUST contain the new endpoint reference of the created resource in the wxf:ResourceCreated element.

R4.6-7: The response MUST 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 man SOAP processors cannot process multiple child elements within a SOAP s:Body. In general, clients can simply issue a wxf:Get to retrieve the representation, since they will have just acquired an endpoint reference to the new resource.

Following is a hypothetical example of a response for a newly created virtual drive:

```
(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:wxf="http://schemas.xmlsoap.org/ws/2004/09/transfer">
(6)  <s:Header>
(7)     ...
(8)    <wsa:Action>
(9)      http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
(10)   </wsa:Action>
(11)   ...
(12) </s:Header>
(13) <s:Body>
(14)   <wxf:ResourceCreated>
(15)     <wsa:Address>
(16)       http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous/
(17)     </wsa:Address>
(18)     <wsa:ReferenceParameters>
(19)       <wsman:ResourceURI>http://samples.org/2005/02/virtualDrive</wsman:ResourceURI>
(20)       <wsman:SelectorSet>
(21)         <wsman:Selector Name="ID"> F: </wsman:Selector>
(22)       </wsman:SelectorSet>
(23)     </wsa:ReferenceParameters>
```

```
(24)    </wxf:ResourceCreated>
(25)  </s:Body>
(26)</s:Envelope>
```

This example assumes the default address format is in use. Note that the response contains two sections, the wxf:ResourceCreated (lines 13-23) section, which contains the new endpoint reference of the created resource, including its ResourceURI and the SelectorSet. This is the address that would be used to retrieve the resource in a subsequent wxf:Get operation.

Note that the service may use a network address which is the same as the <wsa:To> address in the wxf:Create request, or may simply use the anonymous address as shown (lines 15-16).

R4.6-8: The service MAY ignore any values in the initial representation which are considered read-only from the point of view of the underlying real-world object.

This is to allow 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.

R4.6-9: If the success of an operation cannot be reported as described in this section and cannot be reversed, a wsman:EncodingLimit fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess SHOULD be returned.

4.7 WS-Transfer:Put

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

R4.7-1: A conformant service is NOT REQUIRED to support Wxf:Put.

R4.7-2: If a single resource instance can be updated (within the constraints of its schema) using a SOAP message, and that resource subsequently can be retrieved using wxf:Get, then a service SHOULD support updating the resource using wxf:Put. The service MAY additionally export a custom method for updates.

R4.7-3: 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, as long as 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.

Note that this 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 must contain the complete new representation for the instance. If the resource schema requires the presence of any given value (minOccurs is not zero), it must 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 which are optional (minOccurs=0), then the wxf:Put message may omit these values and rely on the defaults being observed during the update.

In short, the Body of the wxf:Put message must not violate the constraints of the associated XML schema. For example, say that wxf:Get returns the following:

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

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

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

In this case, the corresponding wxf:Put must have all three elements, since 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 R4.7-3, the service should ignore attempts to update values which are read-only with regard to the underlying real-world object.

To update isolated values without having to supply values which will not change, use the fragment-level transfer mechanism in section 4.8.

R4.7-4: A conformant service SHOULD support wxf:Put using the same endpoint reference as a corresponding wxf:Get or other messages, but this is NOT REQUIRED if the Put mechanism for a resource is semantically distinct.

R4.7-5: 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:

- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues if any values in the s:Body are not correct
- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues if any values in the s:Body are missing
- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace if the wrong XML schema namespace is used and is not recognized by the service

R4.7-7: If an object cannot be updated due to locking conditions, simultaneous access, or similar conflicts, a wsman:Concurrency fault SHOULD be returned.

R4.7-8: A wxf:Put operation MAY result in a change to the endpoint reference for the resource, since the values being updated may in turn cause an identity change.

Since WS-Management services typically delegate the wxf:Put to underlying subsystems, the service might not always be aware of an identity change. Clients should make use of the mechanism in section 3.5 to be informed of EPR changes that may have occurred as a side-effect of executing the wxf:Put.

R4.7-9: It is RECOMMENDED that the service return the new representation in all cases. Knowing whether the new representation is different than the requested update is often difficult because resource-constrained implementations may have insufficient resources to determine the equivalence of the requested update from the result.

The implication of this rule is that, if the new representation is not returned, it must have precisely matched what was submitted in the wxf:Put message. Since implementations can rarely assure this, they should always return the new representation.

R4.7-10: If the success of an operation cannot be reported as described in this section due to encoding limits or other reasons and it cannot be reversed, a wsman:EncodingLimit fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess SHOULD be returned.

R4.7-11: The Put operation may contain updates of multiple values. The service MUST successfully carry out an update of all the specified values or return the fault which 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.

4.8 Fragment-Level WS-Transfer

Because WS-Transfer works with entire instances and it may 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 via WS-Transfer operations. This is done using special usage of WS-Transfer.

Because of the XML schema limitations discussed in section 4.7, simply returning a subset of the XML defined for the object being accessed is often incorrect, as 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.

R4.8-1 A conformant service is NOT REQUIRED to support fragment-level WS-Transfer. If the service supports fragment-level access, the service MUST NOT behave as if normal WS-Transfer operations were in place, but MUST operate exclusively on the fragments specified. If the service does not support fragment-level access, it MUST return a wsman:UnsupportedFeature fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAccess.

R4.8-2 A conformant service which supports fragment-level WS-Transfer MUST accept the following SOAP header in all requests and include it in all responses which transport the fragments:

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

The default value of this header is the XPath 1.0 expression identifying the fragment being transferred (using **http://www.w3.org/TR/1999/REC-xpath-19991116**), with relation to the full representation of the object. If an expression other than XPath 1.0 is used, a Dialect attribute MUST be added to indicate this, as follows:

```
<wsman:FragmentTransfer s:mustUnderstand="true"
   Dialect="URIToNewSelectorDialect">
   dialect text
</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 special-cased due to its importance, but it is not mandated. 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.

R4.8-3 For WS-Transfer operations, the Dialect used is XPath 1.0 (**http://www.w3.org/TR/1999/REC-xpath-19991116),** so the following rules hold when interpreting the XPath:

Value of /s:Header/wsman:FragmentTransfer is an XPath [XPath 1.0] expression. The context of the expression is:

- Context Node: The body of the XML element that would be returned as a direct child s:Body if fragment transfer were not in use.
- 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 first child of /s:Envelope/s:Body.

This means that the XPath is to be interpreted relative to XML of the returned object 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.

Note that an XPath value may refer to the entire node, so the concept of a fragment can include the entire object, making fragment-level WS-Transfer a proper superset of normal WS-Transfer.

If the full XPath cannot be supported, a common subset for this purpose is described in Chapter 12 of this specification. However, in such cases, the Dialect value is still that of XPath.

R4.8-4 If a service understands fragment transfers but does not understand the specified fragment dialect or the default dialect, then the service MUST issue a wsman:FragmentDialectNotSupported fault.

R4.8-5 All transfers in either direction of the XML fragments must be wrapped with a <wsman:XmlFragment> wrapper containing a definition which suppresses validation and allows any content to pass. A service MUST 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 MUST fault the request using a wxf:InvalidRepresentation fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFragment.

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

R4.8-6 If fragment-level WS-Transfer is supported, a conformant service SHOULD support at least leaf-node, value-level access using an XPath using the **/text()** function. 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.

R4.8-7 For all fragment-level operations, partial successes are NOT permitted. The entire meaning of the XPath or other dialect MUST be fully observed by the service in all operations, and the entire fragment that is specified MUST 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 sections show how the underlying WS-Transfer operations change when transferring XML fragments.

4.9 Fragment-Level WS-Transfer:Get

Fragment-level gets are similar to full wxf:Get, except for the wsman:FragmentTransfer header (line 24). This example is drawn from the example in section 4.2:

```
<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://samples.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>
     uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
   </wsa:MessageID>
   <wsman:SelectorSet>
      <wsman:Selector Name="LUN"> 2 </wsman:Selector>
   </wsman:SelectorSet>
   <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
   <wsman:FragmentTransfer s:mustUnderstand="true">
     Manufacturer
   </wsman:FragmentTransfer>
  </s:Header>
```

```
    <s:Body/>
  </s:Envelope>
```

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

Note that the service MUST repeat the **wsman:FragmentTransfer** in the wxf:GetResponse (lines 47–49) 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 which would otherwise apply:

```
(27) <s:Envelope
(28)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(29)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(30)     xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(31)  <s:Header>
(32)    <wsa:To>
(33)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(34)    </wsa:To>
(35)    <wsa:Action s:mustUnderstand="true">
(36)       http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(37)    </wsa:Action>
(38)    <wsa:MessageID s:mustUnderstand="true">
(39)       uuid:d9726315-bc91-430b-9ed8-ce5ffb858a88
(40)    </wsa:MessageID>
(41)    <wsa:RelatesTo>
(42)        uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(43)    </wsa:RelatesTo>
(44)    <wsman:FragmentTransfer s:mustUnderstand="true">
(45)      Manufacturer
(46)    </wsman:FragmentTransfer>
(47)  </s:Header>
(48)  <s:Body>
(49)    <wsman:XmlFragment>
(50)      <Manufacturer> Acme, Inc. </Manufacturer>
(51)    </wsman:XmlFragment>
(52) </s:Body>
(53) </s:Envelope>
```

The output (lines 52-54) is like that supplied by a typical XPath processor and may or may not contain XML namespace information or attributes.

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

```
(54) <wsman:FragmentTransfer s:mustUnderstand="true">
(55)    Manufacturer/text()
(56) </wsman:FragmentTransfer>
```

The above request will yield this XML:

```
(57)    <wsman:XmlFragment>
(58)      Acme, Inc.
```

```
(59)    </wsman:XmlFragment>
```

4.10 Fragment-Level WS-Transfer:Put

Fragment-level WS-Transfer:Put works like regular wxf:Put except that it only transfers the part being updated. While the fragment may 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.

It is important to note that 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.

Use the following XML for illustrative purposes:

```
1       <a>
2         <b>
3           <c> </c>
4           <d> </d>
5         </b>
6         <e>
7           <f> </f>
8           <g> </g>
9         </e>
10      </a>
```

While <a> is the entire representation of the resource instance, if the operation references the a/b node during the wxf:Put, 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 only to update <d>, then the XPath expression used is ”b/d”.

Continuing from the example in sections 4.2, if the client wanted to update the <BootPartition> value from 0 to 1, then the following wxf:Put fragment could be sent to the service:

```
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://samples.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/Put
17      </wsa:Action>
18      <wsa:MessageID>
19        uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
20      </wsa:MessageID>
21      <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>
```

Note that the <BootPartition> wrapper is present because the XPath value specifies this. If "BootPartition/text()" were used as the expression, then the Body could contain just the value, as follows:

```
  ...
  <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 only needs to supply the parts that are different than what is requested. This would generally not occur for single values, since a failure to honor the new value would result in a wxf:InvalidRepresentation fault.

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">
        uuid:d9726315-bc91-430b-9ed8-ce5ffb858a88
     </wsa:MessageID>
     <wsa:RelatesTo>
```

```
          uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
      </wsa:RelatesTo>
      <wsman:FragmentTransfer s:mustUnderstand="true">
        BootPartition/text()
     </wsman:FragmentTransfer>
    </s:Header>
    <s:Body>
      <wsman:XmlFragment>
         1
       </wsman:XmlFragment>
  </s:Body>
  </s:Envelope>
```

R4.10-1 As for normal wxf:Put, the service MAY ignore any read-only values supplied as part of the fragment for updating.

R4.10-2 If the service encounters an attempt to update a read-only value, a wsman:ActionNotSupported fault is returned with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ReadOnly.

Note that the fragment-level Put 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 (since 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.

WS-Transfer states that if the new representation differs from the input, then it should be returned in the response. With fragment-level wxf:Put, this only applies 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:

```
(60) <wsman:FragmentTransfer s:mustUnderstand="true">
(61)    AssetLabel
(62) </wsman:FragmentTransfer>
(63) <s:Body>
(64)    <wsman:XmlFragment xmlns:xsi="www.w3.org/2001/XMLSchema-instance">
(65)      <AssetLabel xsi:nil="true"/>
(66)    </wsman:XmlFragment>
(67) </s:Body>
```

4.11 Fragment-Level WS-Transfer:Create

Use of wxf:Create for fragments only applies if the XML schema for the targeted object supports optional elements which are not currently present, or supports arrays with varying numbers of elements and the client wishes to insert an element in an array (a repeated element). If entire array replacement is needed, then Fragment-level wxf:Put should be used. For array access, the XPath array access notation (the [ ] operators) can be conveniently used. Note that wxf:Create may only be used to add new content, but cannot update existing content.

To insert a value which may be legally added (according to the rules of the schema for the object), the wsman:FragmentTransfer identifies the path of the item to be added. For instance, assume the following

message fragment is sent to a LogicalDisk resource:

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

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

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

This would presumably result in the creation of a <VolumeLabel> element where none existed before.

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

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

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>
</s:Envelope>
```

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 must know the number of elements in the array so that the new index can be used. 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:

```
<s:Body>
   <wsman:XmlFragment>
      <BlockedIPAddress> 123.12.188.44 </BlockedIPAddress>
    </wsman:XmlFragment>
</s:Body>
</s:Envelope>
```

This will presumably result in adding a third IP address to the <BlockedIPAddress> array (a repeated element), assuming at least two elements are at that level already.

R4.11-1: A service MUST not use wxf:Create to perform an update on an existing representation. The targeted object must not already exist, or else the service SHOULD return a wsman:AlreadyExists fault.

R4.11-2: If the wxf:Create fails because the result would not conform to the schema in some way, a wxf:InvalidRepresentation fault is returned.

As defined in section 4.6, the wxf:CreateResponse contains the EPR of the created resource. In the case of fragment-level Create, the response additionally contains the wsman:FragmentTransfer block, including the path (line 12) in a SOAP header. Note that the wxf:ResourceCreated EPR continues to refer to the entire object, not just the fragment:

```
(68) <s:Envelope
(69)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(70)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(71)     xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
(72)     xmlns:wxf="http://schemas.xmlsoap.org/ws/2004/09/transfer">
(73)  <s:Header>
(74)     ...
(75)    <wsa:Action>
(76)     http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
(77)    </wsa:Action>
(78)       ...
(79)      <wsman:FragmentTransfer s:mustUnderstand="true">
(80)          Path To Fragment
(81)     </wsman:FragmentTransfer>
(82)
(83)     <s:Body>
(84)      <wxf:ResourceCreated>
(85)         <wsa:Address> ... </wsa:Address>
(86)         <wsa:ReferenceParameters>
(87)          <wsman:SelectorSet>
(88)            <wsman:Selector ...> </wsman:Selector>
(89)          </wsman:SelectorSet>
(90)         </wsa:ReferenceParameters>
(91)        </wxf:ResourceCreated>
(92)      </s:Body>
(93) </s:Envelope>
```

As discussed in section 4.6, 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.

4.12 Fragment-Level WS-Transfer:Delete

Use of wxf:Delete for fragments only applies if the XML schema for the targeted object supports optional elements which may be removed from the representation object, or supports arrays (repeated elements) with varying numbers of elements and the client wishes to remove an element in an array. If entire array replacement is needed, then Fragment-level Put should be used. For array access, the XPath array access notation can be conveniently used.

To delete a value which may be legally removed (according to the rules of the schema for the object), the wsman:FragmentTransfer identifies the path of the item to be removed:

```
<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. The following example deletes the third <User> element in the representation:

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

Note that the <Body> is empty for all wxf:Delete operations, even with fragment-level access, and all normal faults for wxf:Delete apply.

R4.12-1: If a value cannot be deleted due to locking conditions or similar phenomena, a wsman:AccessDenied fault SHOULD be returned.

5 WS-Enumeration

5.1 Introduction

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

R5.1-1: A service is NOT REQUIRED to support WS-Enumeration if enumeration of any kind is not supported.

R5.1-2: If simple, unfiltered enumeration of resource instances is exposed through Web services, a conformant service MUST support WS-Enumeration to expose this. The service MAY also support other techniques for enumerating the instances.

R5.1-3: 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: An initial wsen:Enumerate is issued to establish the enumeration context and wsen:Pull operations are used to iterate over the result set. When the enumeration iterator is no longer required and not yet exhausted, a wsen:Release is issued to release the enumerator and associated resources. As with other WS-Management methods, the enumeration may make use of wsman:OptionSet.

If the default addressing model is used, the service may make use of the SelectorSet to scope the enumeration, although typically the ResourceURI would be used alone.

R5.1-4: A service is NOT REQUIRED to implement any of the following messages from WS-Enumeration and implementing them is NOT RECOMMENDED:

Renew, GetStatus, or EnumerationEnd or any associated responses. Since these messages are OPTIONAL, it is RECOMMENDED that the service fault both Renew and GetStatus requests with a wsa:ActionNotSupported fault.

R5.1-5: If a service is exposing enumeration, it MUST 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 Release may be a simple no-op in reality, so it is trivial to implement (it always succeeds when the operation is valid). But it must be supported in any case to allow for the uniform construction of clients.

R5.1-6: The wsen:Pull and wsen:Release operations are a continuation of the original wsen:Enumerate. 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.

Note that some transports like HTTP may 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 may require that the same connection be used. This specification establishes no requirements in this regard. However, R5.1-6 establishes that the user credentials may not change during the entire enumeration sequence.

5.2 WS-Enumeration:Enumerate

Enumerations are initiated by the wsen:Enumerate message.

5.2.1 General

R5.2.1-1: A conformant service is NOT REQUIRED to accept a wsen:Enumerate message with an EndTo address as R5.1-4 recommends not supporting the EndEnumerate message, and may issue a wsman:UnsupportedFeature fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode.

R5.2.1-2: A conformant service MUST accept a wsen:Enumerate message with an Expires timeout or fault with wsman:UnsupportedFeature and a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime.

R5.2.1-3: The wsman:Filter element (see section 5.3) in the wsen:Enumerate body MUST be either simple text or a single complex XML element. A conformant service MUST NOT accept mixed content of both text and elements, or multiple peer XML elements under the wsman:Filter element.

While 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 **http://www.w3.org/TR/1999/REC-xpath-19991116**, which is XPath 1.0. Resource-constrained implementations may have difficulty exporting full XPath processing and yet still wish 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 may be described using that Dialect value.

No rule mandates the use of XPath or any subset as a filtering dialect. Note that if no Dialect is specified, the default interpretation is that the Filter value is in fact XPath (as specified in WS-Enumeration).

R5.2.1-4: A conformant service is NOT REQUIRED to 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, a wsen:CannotProcessFilter fault SHOULD be returned with some text indication as to what went wrong.

Some services REQUIRE filters to function, as their search space is so large that simple enumeration is meaningless or impossible.

R5.2.1-5: A conformant service MUST fault any request without a wsman:Filter if a wsman:Filter is actually required, using a wsman:UnsupportedFeature fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired.

R5.2.1-6: 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 may include or exclude the results of these operations in real-time, or may produce an initial snapshot of the enumeration and execute the wsen:Pull requests from this snapshot, or may deny access to other operations while enumerations are in progress.

5.2.2 Enumeration "Count" Option

In order 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 headers for use in wsen:Enumerate or wsen:Pull is

```
(1) <s:Header>
(2) ...
(3)     <wsman:RequestTotalItemsCountEstimate …/>
(4) </s:Header>
(5)
```

The header used by the service to return the value is

```
(6)       <s:Header>
(7)        ...
(8)          <wsman:TotalItemsCountEstimate>
(9)            xs:nonNegativeInteger
(10)  </wsman: TotalItemsCountEstimate>
(11)</s:Header>
```

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

wsman:RequestTotalItemsCountEstimate

The presence of this element (no value is required) as a SOAP header on an wsen:Enumerate or wsen:Pull message indicates that the client is requesting that the associated response message

should include 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

The presence of this element 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 services understands the TotalItemsCountEstimate feature, but cannot determine the number of items for some reason, the service will respond with the wsman:TotalItemsCountEstimate element having an xsi:nil attribute with value 'true', and having no value:

```
(12) <wsman:TotalItemsCountEstimate xsi:nil="true"/>
```

R5.2.2-1: A conformant service MAY support the ability to return an estimate of the number of items in an enumeration sequence. If a service receives an wsen:Enumerate or wsen:Pull message without the wsman:RequestTotalItemsCountEstimate SOAP header, the service MUST NOT return the wsman:TotalItemsCountEstimate SOAP header on the associated response message.

R5.2.2-2: 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.

5.2.3 Optimization For Enumerations With Small Result Sets

In order to optimize the number of round-trip messages required to enumerate the items in an enumerable resource, a client may request optimized enumeration behavior. This 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 may be used even for large enumerations in order 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.

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

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

wsman:OptimizeEnumeration

The presence of this element as a child of the wsen:Enumerate element indicates that the client is requesting an optimized enumeration.

wsman:MaxElements

This OPTIONAL element (of type xs:positiveInteger) indicates the maximum number of items the consumer is willing to accept in the wsen:EnumerateResponse. When this element is absent, its implied value is 1. Implementations MUST NOT return more than this number of elements in the Enumerate response message. Implementations MAY return fewer than this number based on either the wsman:OperationTimeout SOAP header, wsman:MaxEnvelopeSize SOAP header, or implementation-specific constraints.

R5.2.3-1: A conformant service MAY support enumeration optimization. If a service receives the wsman:OptimizeEnumeration element in an 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.

Note that 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.

R5.2.3-2: A conformant service receiving an wsen:Enumerate message without the wsman:OptimizeEnumeration element MUST NOT return any enumeration items in the wsen:EnumerateResponse message and MUST return an wsen:EnumerationContext initialized to return the first items when the first wsen:Pull message is received.

Clearly, the service must not implement the optimization if it was not requested, or clients unaware of the optimization will incorrectly process the enumeration result.

R5.2.3-3: A conformant service receiving an wsen:Enumerate message with the wsman:OptimizeEnumeration element and without a wsman:MaxElements element MUST interpret the request as being for zero or one item in the optimized response. The service MUST not return a count of items larger than one unless specifically requested.

When requested by the client, a service implementing the optimized enumeration will respond with the following additional content in an wsen:EnumerateResponse message:

```
(9) <s:Body>
(10)    <wsen:EnumerateResponse>
(11)      <wsen:EnumerationContext> ... </wsen:EnumerationContext>
(12)      <wsman:Items>
(13)        ...same as for wsen:Items in wsen:PullResponse
(14)     </wsman:Items> ?
(15)     <wsman:EndOfSequence/> ?
(16)    <wsen:EnumerateResponse> ... </wsen:EnumerateResponse>
(17)</s:Body>
```

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

wsman:Items

The optional element 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

This optional element 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

The wsen:EnumerateResponse message requires the presence of a wsen:EnumerationContext. If the wsman:EndOfSequence is also present, clearly 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. Note that while the wsen:EnumerationContext MUST be present, there is no value required, so in cases where the wsman:EndOfSequence is present, the value for wsen:EnumerationContext SHOULD be empty:

```
(18) ...
(19) <s:Body>
(20)   <wsen:EnumerateResponse>
(21)    <wsen:EnumerationContext/>
(22)     <wsman:Items>
(23)        Items
(24)     </wsman:Items>
(25)     <wsman:EndOfSequence/>
(26)   <wsen:EnumerateResponse> ... </wsen:EnumerateResponse>
(27) </s:Body>
(28)
```

R5.2.3-4: A conformant service that supports optimized enumeration and is responding with an wsen:EnumerateResponse message MUST include either or both the wsman:Items or wsman:EndOfSequence elements 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.

R5.2.3-5: A conformant service that supports optimized enumeration and has not returned all items of the enumeration sequence in the wsen:EnumerateResponse message MUST return an wsen:EnumerationContext 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 MUST 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. An wsen:EnumerateResponse message with optimized enumeration will include either or both the wsman:Items and wsman:EndOfSequence elements.

Clients concerned about the size of the initial response, irrespective of the number of items should use the wsman:MaxEnvelopeSize mechanism in 3.2.

5.3 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 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.

```
<wsman:Filter Dialect="xs:anyURI"?> xs:any </wsman:Filter>
```

The Dialect attribute is OPTIONAL. When not specified it has an implied value of "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.

R5.3-1: If a service supports filtered enumeration using wsen:Filter, it MUST also support filtering using wsman:Filter. This 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.

R5.3-2: If a service supports filtered enumeration using wsman:Filter, it SHOULD also support filtering using wsen:Filter. This allows clients coded to WS-Enumeration to interact with a WS-Management service.

R5.3-3: If a wsen:Enumerate request contains both wsen:Filter and wsman:Filter, the service MUST 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 is selected (within the capabilities of XPath, for example)
4) Composition of new output (XQuery), including simple projection

Most services will fall into (1) or (2). However, if a service wishes to support fragment-level enumeration to complement fragment-level WS-Transfer (section 4.8), then the service should implement (3) as well. Only rarely will services implement (4).

Note that XPath 1.0 can be used simply for filtering, or may 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 section 5.6 applies.

If full XPath cannot be supported, a common subset for this purpose is described in section 13.2 of this specification.

A typical example of the use of XPath in a filter is shown below. Assume 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 as such. The XPath is evaluated as if each item in the wsen:Items block were a separate document.

When used for simple document processing, the following two XPath expressions would both "select" the entire <DiskInfo> node:

```
   ../DiskInfo
----- OR -----
   .
```

If used as as a "filter", this XPath 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 only selects the XML node if the test expression in brackets evaluates to logical "true":

```
   ../DiskInfo[LogicalDisk="C:"]
```

In this case, the item must refer to disk drive "C:" or the XML node is not selected. This XPath filters out all <DiskInfo> instances for other drives. 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 section 7.2.2 on filtering over WS-Eventing subscriptions.

5.4 WS-Enumeration:Pull

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

Since 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 should 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.

In general, since wsman:MaxEnvelopeSize size can be requested for any response in WS-Management, wsen:MaxCharacters is generally redundant and preferably is omitted from the wsen:Pull message, with wsman:MaxEnvelopeSize used instead. However, if present, it has the following characteristics:

R5.4-1: If a service is exposing enumeration and supports wsen:Pull with the wsen:MaxCharacters element, the service SHOULD implement this as a general guideline or hint, but MAY ignore it if wsman:MaxEnvelopeSize is present, since that takes precedence. The service SHOULD NOT fault in the case of a conflict but SHOULD observe the wsman:MaxEnvelopeSize value.

R5.4-2: 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 MUST NOT violate wsman:MaxEnvelopeSize in any case.

R5.4-3: If a wsen:PullResponse would violate the wsman:MaxEnvelopeSize request, the service MUST return a wsman:EncodingLimit fault with a detail code of one of the following:

- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize if the client's requested maximum would have been exceeded
- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit if the service's internal limit was exceeded

In general, wsen:MaxCharacters is a hint, and wsman:MaxEnvelopeSize is a strict rule and may not be exceeded.

R5.4-4: 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.

Note that if a fault occurs with a wsen:Pull request, the service generally should not cancel the entire enumeration, but should simply freeze the cursor and allow the client to try again.

The EnumerationContext from only the latest response is considered to be valid. While the service may return the same EnumerationContext values with each wsen:Pull, it is not required to do so and may in fact change the EnumerationContext unpredictably.

R5.4-5: A conformant service MAY ignore wsen:MaxTime if wsman:OperationTimeout is also specified, as wsman:OperationTimeout takes precedence. These have precisely the same meaning and may be used interchangeably. If both are used, the service SHOULD only observe

the wsman:OperationTimeout.

Clients should omit wsen:MaxTime and use only wsman:OperationTimeout and should use wman:MaxEnvelopeSize in preference to wsen:MaxCharacters.

Note that any fault issued for wsen:Pull applies to the wsen:Pull itself, not the underlying enumeration that is in progress. The most recent EnumerationContext is still considered valid, and the service should try to allow a retry of the most recent wsen:Pull so that the client can continue. However, the service may terminate (as specified in R5.3-3) early upon encountering any kind of problem.

R5.4-6: The service MUST accept a wsen:Pull message with an endpoint reference identical to that specified for the original wsen:Enumerate. 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 Pull may be retried:

R5.4-7: 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 occur and that the client is not likely to succeed by simply issuing another wsen:Pull. 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:

```
(1) ...
(2) <s:Body>
(3)   <wsen:PullResponse>
(4)      <wsen:EnumerationContext> ...possibly updated... </wsen:EnumerationContext>
(5)      <wsen:Items/>
(6)   </wsen:PullResponse>
(7) </s:Body>
(8)
```

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 will know which one to return based on its internal state. For example, on the very first wsen:Pull if the service is waiting for another component, then a wman:TimedOut fault may be likely. If the enumeration is continuing with no problem and after 50 requests a particular wsen:Pull times out, the service may simply send back zero items in the expectation that the client should continue with another wsen:Pull.

R5.4-8: 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 which are too large, other faults may also be returned. In all such cases, the service SHOULD invalidate the enumeration context as well.

R5.4-9: If the wsen:EndOfSequence marker occurs in the wsen:PullResponse, then the wsen:EnumerationContext MUST be omitted, as the enumeration has completed. The client does not need to subsequently issue a wsen:Release.

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 should not issue a

wsen:Release if the wsen:EndOfSequence actually occurs, as the enumeration is then completed and the enumeration context is invalid.

R5.4-10: If no wsen:MaxElements is specified, the batch size is 1, as specified in WS-Enumeration.

R5.4-11: If wsen:MaxElements is larger than the service supports, the service MAY ignore the value and use any default maximum of its own.

The service should export its maximum wsen:MaxElements value in metadata, but the format and location of such metadata is beyond the scope of this specification.

R5.4-12: The wsen:EnumerationContext MUST be present in all wsen:Pull requests, even if the service uses a constant value for the lifetime of the enumeration sequence. This is mandated by WS-Enumeration and repeated here for clarity.

5.5 WS-Enumeration:Release

As previously stated, wsen:Release MUST be implemented. It is only used to perform an early cancellation of the enumeration.

In cases where it is not actually needed, the implementation can expose a dummy implementation which always succeeds. This promotes completely uniform client-side messaging.

R5.5-1: The service MUST recognize and process the wsen:Release message if the enumeration is terminated early. Note that if a wsen:EndOfSequence marker occurs in a wsen:PullResponse, the enumerator is already completed and a wsen:Release cannot be issued, as there is no up-to-date wsen:EnumerationContext to use.

R5.5-2: The client may fail to deliver the wsen:Release 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 MUST result in a wsen:InvalidEnumerationContext fault.

R5.5-3: The service MUST accept a wsen:Release message with an endpoint reference identical to that specified for the original wsen:Enumerate, assuming the enumeration is still active and the wsen:EndOfSequence has not occurred. A wsa:MessageInformationHeaderRequired fault SHOULD be returned if the EPR is missing or different.

R5.5-4: The service MAY accept a wsen:Release 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 a detail code of 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.

In most cases, it is desirable to be able to asynchronously cancel an outstanding wsen:Pull. This capability requires the service to be able to receive the wsen:Release asynchronously while still processing a pending wsen:Pull. Further, it requires that the EnumerationContext contain information which is constant between wsen:Pull operations. Note that if the 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 EnumerationContext contains a value which 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.

5.6 Ad-Hoc Queries and Fragment-Level Enumerations

As discussed in section 4.8, 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 wish to execute XPath or XQuery operations over the representation to receive ad-hoc results.

Since SOAP messages must conform to known schemas, and since ad-hoc queries return results which are dynamically generated and may conform to no schema, the wsman:XmlFragment wrapper from section 4.8 must be used to wrap the responses.

R5.6-1: 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 (or wsen:EnumerateResponse if OptimizedEnuemration is used) SHOULD be wrapped with wsman:XmlFragment wrappers as follows:

```
...
<s:Body>
  <wsen:PullResponse>
     <wsen:EnumerationContext> ...possibly updated... </wsen:EnumerationContext>
     <wsen:Items>
       <wsman:XmlFragment>
                XML content
       </wsman:XmlFragment>
       <wsman:XmlFragment>
                XML content
       </wsman:XmlFragment>
         ...etc.
     </wsen:Items>
  </wsen:PullResponse>
</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.

Note that XPath 1.0 and XQuery 1.0 already support returning subsets or compositions of representations, so they are suitable for use in this regard.

R5.6-2: If the service does not support fragment-level enumeration, it should return a wsen:FilterDialectRequestedUnavailable fault, the same as for any other unsupported dialect.

Note that the XPath expression used for filtering is still that described in WS-Enumeration. The wsman:XmlFragment wrappers are applied after the XPath is evaluated to prevent schema violations if the XPath selects node sets which are fragments and not legal according to the original schema.

5.7 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 endpoint references of objects rather than the objects themselves. Such EPRs should 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 endpoint references.

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.

R5.7-1: 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.

Here is an example:

```
<s:Header>
    ...
   <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
  </wsa:Action>
  ...

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

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.

The hypothetical response would appear as follows:

```
<s:Body>
  <wsen:PullResponse>
    <wsen:Items>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      <wsa:EndpointReference> ... </wsa:EndpointReference>
      ...
    </wsen:Items>
  </wsen:PullResponse>
```

R5.7-2: 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 which juxtaposes two XML representations: the payload representation followed by the associated wsa:EndpointReference.

As an example, the wsman:EnumerationMode header appears as follows:

```
<s:Header>
    ...
   <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
  </wsa:Action>
```

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

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 -->
      </wsman:Item>
      ...
    </wsen:Items>
  </wsen:PullResponse>
```

In the above example, each item is wrapped in a wsman:Item wrapper (line 46), which itself contains the representation object (47) followed by its EPR (48). As many wsman:Item objects may be present as is consistent with other encoding limitations.

R5.7-3: If a service does not support the wsman:EnumerationMode modifier, it MUST return a fault of wsman:UnsupportedFeature with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EnumerationMode.

6 Custom Actions (Methods)

6.1 General

Custom actions or "methods" are ordinary SOAP messages with unique Actions. An implementation may support resource-specific methods in any form, subject to the addressing model and restrictions described in section 2.0 of this specification.

R6.1-1: A conformant service is NOT REQUIRED to expose any custom actions or methods.

R6.1-2: If custom methods are exported, WS-Addressing rules as described elsewhere in this specification MUST be observed and each custom method MUST have a unique wsa:Action.

A custom method must be a dedicated WS-Addressing Action URI directed to a specific Resource and may not be described using a wsa:To address alone.

In general, Options should not be used to implement custom methods or in combination with them, since options are a parameterization technique for message types which are not user-extensible, such as WS-Transfer. Custom methods defined in WSDL expose any required parameters and thus expose naming and type checking in a stringent way, so mixing Options with parameters is likely to lead to confusion.

7 Eventing

7.1 General

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

R7.1-1: If a Resource can emit events and allows clients to subscribe to and receive event messages, it MUST do so by implementing WS-Eventing as specified in this specification.

R7.1-2: If WS-Eventing is supported, the Subscribe, Renew, and Unsubscribe messages MUST be supported. SubscriptionEnd is OPTIONAL, and GetStatus is NOT RECOMMENDED.

7.2 Subscribe

7.2.1 General

WS-Management uses wse:Subscribe substantially as documented in WS-Eventing, except that the WS-Management endpoint reference model is incorporated as described in section 2.1.

R7.2.1-1: The identity of the event source MUST be based on the WS-Addressing endpoint reference.

R7.2.1-2: A service is NOT REQUIRED to support distinct addresses and distinct security settings for NotifyTo and 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 a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode.

The service should verify that the address will actually be usable. For example, if the address cannot be reached due to firewall configuration and the service can detect this, then it should issue a fault.

R7.2.1-3: Because many delivery modes require a separate connection to deliver the event, the service SHOULD comply with the security profiles defined in section 9 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 wse:UnsupportedFeature fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress.

Since clients may need to have client-side context sent back with each event delivery, the wse:NotifyTo address in the wse:Delivery block should be used for this purpose. This wse:NotifyTo operation may contain any number of client-defined reference parameters.

R7.2.1-4: 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 the address is not valid or permissions cannot be acquired, it should emit a wse:EventSourceUnableToProcess fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnusableAddress.

This can occur either when the address is incorrect or if the event source cannot acquire permissions to deliver events properly.

R7.2.1-5: A service is REQUIRED to deliver any reference parameters in the wse:NotifyTo address with each event delivery as specified in section 2.2 of this specification. If EndTo is supported, this behavior applies as well.

When the default addressing model is in use by the service. the ResourceURI is often used to reference the logical event source, and Selector values may 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 logically can overlap with the Filter mechanism in the subscription body itself, so due consideration must be given to the interplay between 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.

R7.2.1-6: If the events contain localized content, the service SHOULD accept a subscription with a wsman:Locale block acting as a "hint" (see section 3.3) 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 SHOULD attempt to localize any descriptive content to the specified language when delivering such events, 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 (already defined in section 3.3) is not a SOAP header and mustUnderstand cannot be used.

R7.2.1-7: The service SHOULD accept a subscription with a wsman:Content-Encoding block within the wse:DeliveryBlock of the wse:Subscribe message. This block acts as a hint to indicate how the delivered events should 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, as in the following example:

```
(7) <wse:Subscribe>
(8)   <wse:Delivery>
(9)     ...
(10)  <wse:NotifyTo>  ... </wse:NotifyTo>
(11)  <wsman:ContentEncoding> UTF-16 </wsman:ContentEncoding>
(12) </wse:Delivery>
(13)</wse:Subscribe>
```

7.2.2 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.

```
<wsman:Filter Dialect="xs:anyURI"?> xs:any </wsman:Filter>
```

The Dialect attribute is OPTIONAL. When not specified it has an implied value of "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 wse:Subscribe element.

If the filter dialect used for the wsen:Subscribe message is "http://www.w3.org/TR/1999/REC-xpath-19991116", the context node is the same as that specified by WS-Eventing (the SOAP Envelope element).

R7.2.2-1: If a service supports filtered subscriptions using wse:Filter, it MUST also support filtering using wsman:Filter. This 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.

R7.2.2-2: If a service supports filtered subscriptions using wsman:Filter, it SHOULD also support filtering using wse:Fitler. This allows clients coded to WS-Eventing to interact with a WS-Management service.

R7.2.2-3: If a wse:Subscribe request contains both wse:Filter and wsman:Filter, the service MUST return a wse:InvalidMessage fault.

The default wse:Filter dialect as defined by WS-Eventing is http://www.w3.org/TR/1999/REC-xpath-19991116, which is XPath 1.0. However, this defines the context node as being the SOAP Envelope. In order 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 the context node is defined as being the element that would be returned as the first child of the SOAP Body element if the Push DeliveryMode were used. The URI for this filter dialect is: **http://schemas.dmtf.org/wbem/wsman/1/wsman/filter/eventRootXPath**. The context node for this expression is:

- Context Node: any XML element that could be returned as a direct child of the s:Body element if the wsman:DeliveryMode 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.

R7.2.2-4: 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 section 5.3 regarding the XPath 1.0 filter dialect also apply to the eventing filter just described.

Resource-constrained implementations may have difficulty providing full XPath processing and yet still wish to use a subset of XPath syntax. This does not require the addition of a new dialect as long as 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 which would result in the XML fragment violating the schema of the event.

A typical example of the use of XPath in a subscription follows. Assume each event that would be delivered has the following XML content:

```
<s:Body>
  <LowDiskSpaceEvent xmlns="...">
    <LogicalDisk>C:</LogicalDisk>
    <CurrentMegabytes>12</CurrentMegabytes>
    <Megabytes24HoursAgo>17</Megabytes24HoursAgo>
  </LowDiskSpaceEvent>
</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 within the s:Body, and does not reference the <s:Body> element as such. The XPath is evaluated as if the content were a separate XML document.

When used for simple document processing, the following two XPath expressions would both "select" the entire <LowDiskSpaceEvent> node:

```
   ../LowDiskSpaceEvent
----- OR -----
   .
```

If used as a "filter", this XPath does not filter out any instances and is the same as selecting all instances of the event, or omitting the filter entirely. However, using the following syntax, the XPath only selects the XML node if the test expression in brackets evaluates to logical "true":

```
   ../LowDiskSpaceEvent[LogicalDisk="C:"]
```

In this case, the event must refer to disk drive "C:" or the XML node is not selected. This XPath would filter out all <LowDiskSpaceEvent> events for other drives. Full XPath implementations may support more complex test expressions:

```
   ../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 may 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 section 5.6 applies.

If full XPath cannot be supported, a common subset for this purpose is described in Chapter 12 of this specification.

R7.2.2-5: The wsman:Filter element MUST 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.

R7.2.2-6: A conformant service is NOT REQUIRED to 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, a wse:EventSourceUnableToProcess fault SHOULD be returned with text explaining why the filter was problematic.

R7.2.2-7: If a service requires complex initialization parameters in addition to the filter, these SHOULD be part of the wsman:Filter block as 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 MUST be devised for the event source and the schema and usage published.

R7.2.2-8: 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 transfer (section 4.8 of this specification).

Note that events, regardless of how they are filtered or reduced, must conform to some kind of XML schema definition when they are actually delivered. Simply sending out unwrapped XML fragments during delivery is not legal.

R7.2.2-9: If the service requires specific initialization XML in addition to the filter to formulate a subscription, this initialization XML MUST form part of the filter body and be documented as part of the filter dialect.

This promotes a consistent location for initialization content, which may be logically seen as part of the filter anyway. The filter XML schema should cleanly separate the initialization and filtering parts into separate XML elements.

See the related section 5.3 on filtering over WS-Enumeration.

7.2.3 Connection Retries

Due to the nature of event delivery, at event-time the subscriber may not be reachable. Rather than terminate all subscriptions immediately, typically the service will attempt to connect several times with suitable timeouts before giving up.

R7.2.3-1: A service MAY observe any connection retry policy, or allow the subscriber to define it by including the following wsman:ConnectionRetry instruction in a subscription. The service is NOT REQUIRED to accept wsman:ConnectionRetry and may return a wsman:UnsupportedFeature fault with a detail code of 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:

```
<wse:Subscribe>
  <wse:Delivery>
```

```
        <wse:NotifyTo>  ... </wse:NotifyTo>
        <wsman:ConnectionRetry Total="count"> xs:duration </wsman:ConnectionRetry>
    </wse:Delivery>
  </wse:Subscribe>
```

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

wsman:ConnectionRetry
: An xs:interval 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.

R7.2.3-2: If the retry counts are exhausted, the subscription SHOULD be considered expired and any normal operations that would occur upon expiration should occur.

Note that the retry mechanism only applies to attempts to connect. Failures to deliver on an established connection should 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.

7.2.4 wse:SubscribeResponse

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

R7.2.4-1: In the 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.

R7.2.4-2: A conformant service is NOT REQUIRED to return the wsen:Expires field in the response, but as specified in WS-Eventing, this implies the subscription does not expire until explicitly canceled.

7.2.5 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 which 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. Should the heartbeat event not arrive, then the client knows that connectivity is bad or that the subscription has expired and it may 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>
```

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

wsman:Heartbeats

Specifies that heartbeat events are added to the event stream at the specified interval.

R7.2.5-1: A service is NOT REQUIRED to support heartbeat events, but SHOULD do so. If the service does not support them, a wsman:UnsupportedFeature fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats MUST be returned to the client. 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 may 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.

R7.2.5-2: While a subscription with heartbeats is active, the service MUST 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, due to colliding time windows and race conditions, 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.

R7.2.5-3: 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.* It is NOT A REQUIREMENT for a service to fail to produce heartbeats at the specified interval if real events have been delivered.

R7.2.5-4: A conformant service MUST NOT send out heartbeats asynchronously to any event deliveries already in progress. They must 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 may never be delivered.

Heartbeats need to be acknowledged like any other event.

The client will assume that the subscription is no longer active if no heartbeats are received within the specified interval, so the service should proceed to cancel the subscription and send any requested SubscriptionEnd messages, as the client will likely resubscribe shortly. Used in combination with bookmarks, 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:

```
(1) s:Envelope ...>
(2)    <s:Header>
(3)     <wsa:To> .... </wsa:To>
```

```
(4)     <wsa:Action s:mustUnderstand="true">
(5)       http://schemas.dmtf.org/wbem/wsman/1/wsman/Heartbeat
(6)     </wsa:Action>
(7)     ...
(8)
```

7.2.6 Bookmarks

Reliable delivery of events is difficult to achieve, so management subscribers should 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.

For this to succeed, event sources must be backed by logs, whether short-term or long-term. The client subscribes as normal for WS-Eventing, and specifies that bookmarks should be used. The service then sends a new bookmark along 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 may 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.

Note that 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.

R7.2.6-1: A conformant service is NOT REQUIRED to support the WS-Management Bookmark mechanism. If the service does not support bookmarks, it should return a wsman:UnsupportedFeature fault with a detail code of 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:

```
(1) <s:Body>
(2)   <wse:Subscribe>
(3)     <wse:Delivery>
(4)       ...
(5)     </wse:Delivery>
(6)     <wsman:SendBookmarks/>
(7)   </wse:Subscribe>
(8) </s:Body>
```

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

wsman:SendBookmarks

This is an element with no value that instructs the service to send a bookmark with each event delivery. Bookmarks apply to all delivery modes.

The bookmark is a token which 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 since the

token is supplied to the same implementation during a subsequent wse:Subscribe operation. The service may 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. The format of the bookmark is entirely determined by the service and should be treated as an opaque value:

```
<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 s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
   ...
   <wsman:Bookmark> xs:any </wsman:Bookmark>
   ...
</s:Header>
<s:Body>
   ...event content...
</s:Body>
</s:Envelope>
```

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

wsman:Bookmark

XML content supplied by the service which indicates the logical position of this event or event batch in the event stream implied by the subscription.

R7.2.6-2: If bookmarks are supported, the wsman:Bookmark element content MUST be either simple text or a single complex XML element. A conformant service MUST NOT accept mixed content of both text and elements, or multiple peer XML elements under the wsman:Bookmark element.

R7.2.6-3: If bookmarks are supported, the service MUST use a wsman:Bookmark element in the Header to send an updated bookmark with each event delivery. Bookmarks only accompany event deliveries and are not part of any SubscriptionEnd message.

Once the subscription has terminated, for whatever reason, a subsequent wse:Subscribe on the part of the client may include the bookmark in the subscribe 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:

```
...
<s:Body>
  <wse:Subscribe>
    <wse:Delivery>  ...  </wse:Delivery>
    <wse:Expires> ... </wse:Expires>
    <wsman:Filter> ... </wsman:Filter>
    <wsman:Bookmark>
     ...last known bookmark from a previous delivery...
    </wsman:Bookmark>
    <wsman:SendBookmarks/>
  </wse:Subscribe>
```

```
    </s:Body>
```

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

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.

R7.2.6-4: The bookmark is a pointer to the last event delivery or batched delivery. The service MUST resume delivery at the first event or events after the event represented by the bookmark. The service MUST NOT replay events associated with the bookmark or skip any events since the bookmark.

R7.2.6-5: 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.

R7.2.6-6: If the bookmark cannot be honored, the service MUST fault with a wsman:InvalidBookmark, with one of the following detail codes:

- http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired: The 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/InvalidFormat: The format is unknown.

If multiple new subscriptions are made using a previous bookmark, the service MAY allow multiple reuse or MAY limit bookmarks to a single subscriber, and may even restrict how long bookmarks may be used before becoming invalid.

A predefined, reserved bookmark value indicates that the subscription should start 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 should replay all possible events which match the filter and any events which subsequently occur for that event source. Absence of any bookmark means "begin at the next available event".

R7.2.6-7: 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.

7.2.7 Delivery Modes

A WS-Management implementation may support a variety of event delivery modes.

In essence, delivery consists of the following:

- 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 Section 9 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, then a wse:DeliveryModeRequestedUnavailable fault should be issued with additional detail to this effect.

R7.2.7-1: 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. Note that a delivery mode implies a specific SOAP message format, so a message which deviates from that format will require a new delivery mode.

R7.2.7-2: The wse:NotifyTo address in the wse:Subscribe message MUST support only a single delivery mode.

This is actually a requirement for the client, since the service cannot verify that this is true. However, if not observed by the client, the service may not operate correctly. If the subscriber supports multiple delivery modes, then the wse:NotifyTo address must be differentiated in some way, such as by adding an additional reference parameter.

7.2.8 Event Action URI

Typically, each event type has its own wsa:Action to quickly identify and route the event. If an event type does not define its own action URI, then the following URI should be used as a default:

```
http://schemas.dmtf.org/wbem/wsman/1/wsman/Event
```

This URI may be required 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 Action URI. This specification places no restrictions on the wsa:Action URI for events. The URI should be as specific as possible in most cases so that it can act as a reliable dispatching point. In many cases, a fixed schema may 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 may reflect the schema and be undifferentiated for all of the various event IDs which may occur, or it may 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 should be made when designing the URIs for events.

7.2.9 Delivery Sequencing and Acknowledgement

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 MAY deliver events for the subscription asynchronously without regard to any events already in transit. This is primarily 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 timestamp 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 may be sent. The service MUST queue all undelivered events for the subscription and only deliver each new event 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 may be sent. The service MUST queue all events for the subscription and deliver 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. Since the receiver uses wsen:Pull to synchronously retrieve the events, acknowledgement is implicit. The order of delivery must be maintained.

There is no implication that ordering of events occurs across subscriptions.

The acknowledgement model is discussed in section 7.7.

7.2.10 Push Mode

The standard mode from WS-Eventing is **http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push,** in which each delivery consists of a single event. There is no acknowledgement, so the delivery cannot be faulted to cancel the subscription.

Therefore, subscriptions made with this delivery mode should 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 should be distinct for each event type, regardless of how strongly typed the event body is.

R7.2.10-1: A service MAY support the http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push delivery mode.

R7.2.10-2: To precisely control how to deal with events which are too large, the service MAY accept the following additional instruction in a subscription:

```
<wse:Delivery>
  <wsa:Address> ... </wsa:Address>
  ...
  <wsman:MaxEnvelopeSize Policy="enumConstant">
     xs:positiveInteger
  </wsman:MaxEnvelopeSize>
  ...
```

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

wsman:MaxEnvelopeSize
: The maximum number of octets for the entire SOAP envelope in a single event delivery.

wsman:MaxEnvelopeSize/@Policy
: An OPTIONAL value with one of the following enumeration values:

a) **CancelSubscription:** cancel on the first oversized event.

b) **Skip:** silently skip oversized events.

c) **Notify:** notify the subscriber that events were dropped as specified in section 7.9.

R7.2.10-3: If wsman:MaxEnvelopeSize is requested, the service MUST NOT send an event body larger than the specified limit. The default behavior is to notify the subscriber as specified in section 7.9, unless otherwise instructed in the subscription, and attempt to continue delivery. If the event exceeds any internal default maximums, the service SHOULD also attempt to notify as specified in section 7.9 rather than terminate the subscription, unless otherwise specified in the subscription. If wsman:MaxEnvelopeSize is too large for the service, the service MUST return a wsman:EncodingLimit fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize.

Note that, in the absence of any other Policy instructions, services should deliver notifications of dropped events to subscribers, as specified in section 7.9.

7.2.11 PushWithAck Mode

This is identical to the standard "Push" mode except that each delivery is acknowledged. Each delivery still has one event, and the wsa:Action indicates the event type. However, a SOAP-based acknowledgement as described in section 7.7 must occur.

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 section 7.2.10.

R7.2.11-1: A service SHOULD support the http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck delivery mode. If the delivery mode is not supported, a fault of wse:DeliveryModeRequestedUnavailable SHOULD be returned.

For management, acknowledged delivery is typically more useful than unacknowledged delivery.

7.2.12 Batched Delivery Mode

Batching events is an effective way of minimizing 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 section 7.7.

R7.2.12-1: A service MAY support the http://schemas.dmtf.org/wbem/wsman/1/wsman/Events delivery mode. If the delivery mode is not supported, a fault of wse:DeliveryModeRequestedUnavailable SHOULD be returned.

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

```
<wse:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
  <wse:NotifyTo>
    wsa:EndpointReferenceType
  </wse:NotifyTo>
  <wsman:MaxElements> xs:positiveInteger </wsman:MaxElements> ?
  <wsman:MaxTime> xs:duration </wsman:MaxTime> ?
```

```
    <wsman:MaxEnvelopeSize Policy="enumConstant"> xs:positiveInteger </wsman:MaxEnvelopeSize> ?
  </wse:Delivery>
```

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

wse:Delivery/@Mode
: MUST be http://schemas.dmtf.org/wbem/wsman/1/wsman/Events.

wse:Delivery/wse:NotifyTo
: This required element MUST contain the endpoint reference to which event messages should be sent for this subscription.

wse:Delivery/wsman:MaxElements
: This optional element MAY contain a positive integer that indicates the maximum number of event bodies to batch into a single SOAP envelope. The resource MUST NOT deliver more than this number of items in a single delivery, although it MAY deliver fewer.

wse:Delivery/wsman:MaxEnvelopeSize
: This optional element MAY contain a positive integer that indicates the maximum number of octets in the SOAP envelope used to deliver the events. Note that wsman:MaxEnvelopeSize only applies to the response to the current message (wse:Subscribe) and does not apply to the resulting delivery stream of a subscription.

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 section 7.9.

wse:Delivery/wsman:MaxTime
: This optional element MAY contain a duration that indicates the maximum amount of time the service should allow to elapse while batching EVENT bodies. That is, 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 should 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, but if present, it serves as an override on anything defined within the filter.

Note that, in the absence of any other instructions in any part of the subscription, services should deliver notifications of dropped events to subscribers, as specified in section 7.9.

If a client wants to discover the appropriate values for wsman:MaxElements or wsman:MaxEnvelopeSize, the client should query for service-specific metadata. The format of such metadata is beyond the scope of this particular specification.

R7.2.12-2: If batched mode is requested in a Subscribe message, and none of MaxElements, MaxEnvelopeSize, and MaxTime are present, the service may pick any applicable defaults. The following faults apply:

- wman:Unsupported with a fault detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements if MaxElements is not supported or is excessive.

- wman:Unsupported with a fault detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize if MaxEnvelopeSize is not supported or is excessive.
- wman:Unsupported with a fault detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime if MaxTime is not supported or is excessive.
- wman:Unsupported with a fault detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy if MaxEnvelopeSize/@Policy is not supported.

R7.2.12-3: If wsman:MaxEnvelopeSize is requested, the service MUST NOT send an event body larger than the specified limit. The default behavior is to notify the subscriber as specified in section 7.9, unless otherwise instructed in the subscription, and attempt to continue delivery. If the event exceeds any internal default maximums, the service SHOULD also attempt notification as specified in section 7.9 rather than terminate the subscription, unless otherwise specified in the subscription.

If a subscription has been created using batched mode, all event delivery messages MUST have the following format:

```
<s:Envelope ...>
  <s:Header>
    ...
    <wsa:Action>
        http://schemas.dmtf.org/wbem/wsman/1/wsman/Events
    </wsa:Action>
    ...
  </s:Header>
  <s:Body>
    <wsman:Events>
      <wsman:Event Action="event action URI"> +
        ...event body...
      </wsman:Event>
    </wsman:Events>
  </s:Body>
</s:Envelope>
```

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

s:Envelope/s:Header/wsa:Action

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

s:Envelope/s:Body/wsman:Events/wsman:Event

Each of these required elements MUST 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

This required attribute MUST contain the Action URI that would have been used for the contained event message.

R7.2.12-4: If batched mode is requested, deliveries MUST be acknowledged as described in section 7.7.

Dropped events (as specified in section 7.9) are encoded along with any other events.

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:

```
...
<wse:Delivery
    Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
  <wse:NotifyTo>
    <wsa:Address>http://2.3.4.5/client</wsa:Address>
  </wse:NotifyTo>
  <wsman:MaxElements>10</wsman:MaxElements>
  <wsman:MaxTime>PT20S</wsman:MaxTime>
  <wsman:MaxEnvelopeSize>8192</wsman:MaxEnvelopeSize>
</wse:Delivery>
```

Following is an example of batched delivery that conforms to this specification:

```
<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:wse="http://schemas.xmlsoap.org/ws/2004/09/eventing">
  <s:Header>
    <wsa:To s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
    <wsa:Action>
        http://schemas.dmtf.org/wbem/wsman/1/wsman/Events
    </wsa:Action>
    ...
  </s:Header>
  <s:Body>
    <wsman:Events>
      <wsman:Event
          Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
        <DiskChange
            xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
          <Drive> C: </Drive>
          <FreeSpace> 802012911 </FreeSpace>
        </DiskChange>
      </wsman:Event>
      <wsman:Event
          Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
        <DiskChange
            xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
          <Drive> D: </Drive>
          <FreeSpace> 1402012913 </FreeSpace>
        </DiskChange>
      </wsman:Event>
```

```
      </wsman:Events>
    </s:Body>
  </s:Envelope>
```

Note the use of the generic Action in line 40, which specifies that this is a batch containing distinct events. The individual event bodies are at lines 48–52 and lines 56–60. Note that the actual Action attribute for the individual events is an attribute of the wsman:Event wrapper.

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

```

  <wse:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull">
    ...
  </wse:Delivery>

```

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

wse:Delivery/@Mode

MUST be "http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull".

R7.2.13-1: A service is NOT REQUIRED to support the http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull delivery mode. If requested but not supported, the service MUST return a fault of wse:DeliveryModeRequestedUnavailable.

Note that wsman:MaxElements, wsman:MaxEnvelopeSize, and wsman:MaxTime do not apply in the wse:Subscribe message when using this delivery mode, as the wsen:Pull message contains all of the necessary functionality for controlling the batching and timing of the responses.

R7.2.13-2: If a subscription incorrectly specifies parameters that are not compatible with "pull mode", then the service SHOULD issue a wsman:UnsupportedFeature fault with a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch.

R7.2.13-3: If pull mode is requested in a Subscribe message and the event source accepts the subscription request, the SubscribeResponse element in the REPLY message MUST contain a wsen:EnumerationContext element suitable for use in a subsequent wsen:Pull operation:

```
  <s:Body ...>
    <wse:SubscribeResponse ...>
      <wse:SubscriptionManager>
        wsa:EndpointReferenceType
      </wse:SubscriptionManager>
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires>
      <wsen:EnumerationContext>...</wsen:EnumerationContext>
```

```
      ...
    </wse:SubscribeResponse>
  </s:Body>
```

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

R7.2.13-4: If pull mode is active, wsen:Pull messages MUST contain the EPR of the subscription manager obtained from the wse:SubscribeResponse message. The EPR reference parameters are of a service-specific format, but may use the WS-Management default addressing model if it is suitable.

R7.2.13-5: 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.

R7.2.13-6: If pull mode is active, and a wsen:Pull request returns events, the service MUST return an updated wsen:EnumerationContext as specified for wsen:Pull, and the subscriber is expected to use the update in the subsequent wsen:Pull, as specified for WS-Enumeration. Bookmarks, if active, may also be returned in the header and must also be updated by the service.

In practice, the service may not actually change the EnumerationContext, but the client should not count on it remaining constant. It is conceptually updated, whether in reality or not.

Note that 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.

R7.2.13-7: If pull mode is active, the service MUST NOT return a wsen:EndOfSequence element in the event stream, as there is no concept of a "last event". Rather, the enumeration context should become invalid if the subscription expires or is canceled for any reason.

R7.2.13-8: If pull mode is used, the service MUST accept the wsman:MaxEnvelopeSize used in the wsen:Pull as the limitation on the event size that can be delivered.

Note that 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.

7.3 GetStatus

This message is optional for WS-Management.

R7.3-1: A conformant service is NOT REQUIRED to implement the GetStatus message or its response. 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 should be implemented rather than GetStatus.

7.4 Unsubscribe

Unsubscribe cancels a subscription.

R7.4-1: If a service supports wse:Subscribe, it MUST implement the Unsubscribe message and ensure that event delivery will be terminated if the message is accepted as valid. It is NOT REQUIRED that the service stop event flow prior to responding to the Unsubscribe message as an atomic operation, only that the event traffic stops at some point.

R7.4-2: A service MAY unilaterally cancel a subscription for any reason, including internal timeouts, reconfiguration, or unreliable connectivity.

Note that clients must be prepared to receive any events already in transit even though they have issued a wse:Unsubscribe message. Clients may fault any such deliveries or accept them, at their option.

Note that the EPR to use for this message was received from the wse:SubscribeResponse in the wse:SubscriptionManager element.

7.5 Renew

According to WS-Eventing, processing the wse:Renew message is not optional, but there is no requirement that it actually must succeed.

R7.5-1: While a service MUST support the wse:Renew message in terms of accepting it as a valid action, a conformant service MAY always fault the request with a wse:UnableToRenew fault, forcing the client to subscribe from scratch.

Renew has no effect on deliveries in progress, bookmarks, heartbeats or other ongoing activity. It simply extends the lifetime of the subscription.

Note that the EPR to use for this message was received from the wse:SubscribeResponse in the wse:SubscriptionManager element.

7.6 SubscriptionEnd

This message is optional for WS-Management. In effect, it is the "last event" for a subscription. Since its primary purpose is to warn a subscriber that a subscription has ended, it is not really suitable for use with pull-mode delivery.

R7.6-1: A conformant service is NOT REQUIRED to implement the SubscriptionEnd message. If implemented, the service MAY fail to accept a subscription with any address differing from the NotifyTo address.

R7.6-2: A conformant service MUST NOT implement the SubscriptionEnd when event delivery is done using pull mode as defined in section 7.2.12.

R7.6-3: If SubscriptionEnd is supported, the message MUST contain any reference parameters specified by the subscriber in the EndTo address in the original subscription.

R7.6-4: 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 since the connection itself will close or terminate.

When the delivery connection is distinct from the subscribe connection, a SubscriptionEnd message is highly recommended, or else the client has no immediate way of knowing that a subscription is no longer active.

7.7 Acknowledgement of Delivery

To ensure delivery is acknowledged at the application level, the original subscription may request that the subscriber physically acknowledge event deliveries, rather than relying entirely on transport-level guarantees.

In other words, the transport may have accepted delivery of the events but not forwarded them to the actual subscriber 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

R7.7-1: A conformant service is NOT REQUIRED to support any specific delivery mode. However, if either of the above delivery modes is requested, to maintain an ordered queue of events, the service MUST wait for the acknowledgement from the client before delivering the next event or events which match the subscription.

R7.7-2: If an acknowledged delivery mode is selected for the subscription, the service MUST include the following SOAP headers in each event delivery:

```
<s:Header>
   <wsa:ReplyTo> where to send the acknowledgement </wsa:ReplyTo>
   <wsman:AckRequested/>
```

wsa:ReplyTo

This will always be present in the event delivery as a consequence of the wsman:AckRequested. The client must extract this address and send the acknowledgement to the specified EPR.

wsman:AckRequested

No content. This requires that the subscriber acknowledge all deliveries as described below.

The client must then reply to the delivery with an acknowledgement or a fault.

R7.7-3: 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 subscriber (receiver) to the service is identical for all delivery modes. It contains a unique wsa:Action, and MUST contain the wsa:RelatesTo field set to the MessageID of the event delivery to which it applies:

```
<s:Envelope ...>
  <s:Header>
     ...
     <wsa:To> endpoint reference from the event delivery ReplyTo field </wsa:To>
     <wsa:Action> http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack </wsa:Action>
```

```
        <wsa:RelatesTo> message ID of original event delivery </wsa:RelatesTo>
        ...
    </s:Header>
    <s:Body/>
  </s:Envelope>
```

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

s:Envelope/s:Header/wsa:Action
MUST be http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack.

s:Envelope/s:Header/wsa:RelatesTo
MUST contain the wsa:MessageID of the event delivery to which it refers.

s:Envelope/s:Header/wsa:To
The endpoint reference address extracted from the ReplyTo field in the event delivery. All reference parameters must be extracted and added to the SOAP header as well.

Note that wsa:RelatesTo may not be omitted as it is the critical item which ensures that the correct delivery is being acknowledged.

In spite of the request to acknowledge, the client may refuse delivery with a fault or fail to respond with the acknowledgement. In this case, the service should terminate the subscription and send any applicable SubscriptionEnd messages.

If the client does not support acknowledgement, it may respond with a wsman:UnsupportedFeature fault with a detail code http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack.

However, this is as difficult as acknowledging the delivery, so most clients should 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.

7.8 Refusal of Delivery

With all acknowledged delivery modes as described in section 7.7, a subscriber may 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 service must be prepared to end the subscription even though a wse:Unsubscribe message is not issued by the subscriber.

R7.8-1: During event delivery, if the receiver faults the delivery with a wsman:DeliveryRefused fault, the service MUST 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 MAY issue the fault as a way to cancel the subscription when it does not have the wse:SubscriptionManager information.

7.9 Dropped Events

Events which cannot be delivered should not 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 which cannot be delivered with push modes or which are too large to fit within the delivery constraints requested by the subscriber:

a) Terminate the subscription

b) Silently skip such events

c) Send a special event in place of the dropped events

These options are discussed in sections 7.2.10 and 7.2.11.

During delivery, the service may have to drop events for a number of reasons: they exceed the maximum size requested by the subscriber, the client cannot keep up with the event flow and there is a backlog, or the service might have been reconfigured or restarted and the events permanently lost. In these cases, a service should inform the client that events have been dropped.

R7.9-1: If a service drops events, it SHOULD issue an **http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents** event which indicates this to the client. Any reference parameters specified in the wse:NotifyTo address in the subscription MUST also be copied into this message. This is a normal event and implicitly considered part of any subscription.

R7.9-2: If an **http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents** event is issued, it MUST take the ordinal position of the original dropped event in the delivery stream. If batched delivery mode is in use, the event takes the place in the batch of the event it represents.

Note that this event is considered the same as any other event with regard to its location and other behavior (bookmarks, acknowledged delivery, location in batch, etc.). It simply takes the place of the dropped event.

```
<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 describes additional, normative constraints on the outline listed above:

s:Envelope/s:Header/wsa:Action

MUST be http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents.

s:Body/wsman:DroppedEvents/@Action

The Action URI of the event which was dropped.

s:Body/wsman:DroppedEvents

A positive integer which represents the total number of dropped events since the subscription was created.

Note that wse:Renew has no effect on the running total of dropped events. Dropped events are like any other events and may require acknowledgement, affect the bookmark location, and so on.

Here 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 dropped event is an event in itself.

R7.9-3: 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.

Since this 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.

8 Metadata and Discovery

The WS-Management protocol is compatible with many techniques for discovery of resources available through a service and composable with many such protocols.

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 *a priori* knowledge of the protocol version or versions supported by the implementation. This message is used to discover the presence of a WS-Management compatible service assuming that a transport address is known over which the message can be delivered. 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 which 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:

```
(1) <s:Envelope
(2)      xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)      xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd"
(4)  <s:Header>
(5)    ...
(6)  </s:Header>
```

```
(7)  <s:Body>
(8)    <wsmid:Identify>
(9)       ...
(10)    </wsmid:Identify>
(11) </s:Body>
(12)</s:Envelope>
```

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

wsmid:Identify
: The body of the Identify request operation. This 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:

```
(13)<s:Envelope
(14)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(15)
   xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsdIdentity/1.0/wsmanIdentity.xsd"
(16)  <s:Header>
(17)   ...
(18)  </s:Header>
(19)  <s:Body>
(20)   <wsmid:IdentifyResponse>
(21)     <wsmid:ProtocolVersion> xs:anyURI </wsmid:ProtocolVersion> +
(22)     <wsmid:ProductVendor> xs:string </wsmid:ProductVendor> ?
(23)     <wsmid:ProductVersion> xs:string </wsmid:ProductVersion> ?
(24)      ...
(25)   </wsmid:IdentifyResponse>
(26)  </s:Body>
(27)</s:Envelope>
```

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

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 MUST be equal to the core XML namespace identifying a supported version of the WS-Management specification. One element MUST be provided for each supported version of the protocol.

wsmid:IdentifyResponse/wsmid:ProductVendor
: An OPTIONAL element that identifies the vendor of the WS-Management service implementation using a widely recognized name or token, such as the official corporate name of the vendor or its stock symbol. Alternately, 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 may follow these standardized elements.

R8.9-1: A WS-Management service SHOULD support the wsmid:Identify operation. A service implementation that supports the operation MUST do so irrespective of the version(s) of WS-Management supported by that service. The operation MUST 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. It SOAP header elements are present, the *s:mustUnderstand* attribute on all such elements should be set to "false". Doing otherwise reduces the likelihood of a successful, version-independent response from the service.

R8.9-2: A service supporting the wsmid:Identify operation MUST 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 containing unexpected or unsupported header content with *mustUnderstand* set to "false", the service MUST NOT fault the request and MUST process the body of the request as though the header elements were not present.

R8.9-3: A service 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.

Note that 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.

In order to provide that a client does not need to have any a-priori knowledge about a service before calling Identify, it is desirable to allow a service to process an Identify message without requiring authentication.

R8.9-4: A service supporting the wsmid:Identify operation MAY expose this operation without requiring client or server authentication in order to process the message.

9 Security

9.1 Introduction

In general, management operations and responses should be protected against attacks such as snooping, interception, replay, and modification during transmission. Generally, authenticating the user who has sent a request is also necessary so that access control rules can be applied to determine whether or not 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 which expect to interoperate should adopt one or more of the transport and security models defined in this chapter and are free to define any additional profiles under different URI-based designators.

9.2 Security Profiles

For this specification, a profile is any arbitrary mix of transport or SOAP behavior which describes a common security need. In some cases, the profile is defined for documentation and metadata purposes, but may not be part of the actual message exchange. Rather, it *describes* the message exchange involved.

Metadata retrieval should 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 are used for descriptive and metadata purposes and do not actually show up in the SOAP traffic, with the exception of the wse:Subscribe message when using any delivery mode which causes a new connection to be created from the publisher to the subscriber (push and batched modes, for example). When a subscription is created, the authentication technique for event-delivery needs to be specified by the subscriber, since the subscriber will have to authenticate the service (acting as publisher) at event-time.

In this specification, security profiles are identified by a URI. As profiles are defined for transports, they should be assigned a URI and published. WS-Management defines a set of standardized security profiles for the common transports HTTP and HTTPS, which are described in 12.4.

9.3 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, then 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 may not know which authentication model to choose or which credentials to use without being told in the subscription.

Because of the wide variety of capabilities of services, no mechanism is defined at the message level for negotiating which security profiles may be supported by the service. Instead, the service should export metadata which describes the available options. The format of such metadata is beyond the scope of this particular specification.

WS-Management defines an additional field in the wse:Delivery block which can communicate authentication information:

```
<s:Body>
  <wse:Subscribe>
    <wse:Delivery>
      <wse:NotifyTo>  Delivery EPR </wse:NotifyTo>
      <wsman:Auth Profile="authentication-profile-URI"/>

```

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

wsman:Auth

This block contains authentication information to be used by the service (acting as publisher) when authenticating to the subscriber (the client) at event delivery time. This block contains a simple string which encodes a token to be used. The Profile attribute indicates the format of the token.

wsman:Auth/@Profile

A URI which indicates which security profile to use when making the connection to deliver events.

If the wsman:Auth block is not present, then the service must infer what to do by using the wse:NotifyTo address using any preconfigured policy or settings it has available. If it is present and no security-related tokens are communicated, the service must know which credentials to use by its own internal configuration.

For example, if the service is already configured to use a specific certificate when delivering events, the subscriber can request standard mutual authentication:

```
<s:Body>
  <wse:Subscribe>
    <wse:Delivery>
      <wse:NotifyTo>  HTTPS address </wse:NotifyTo>
      <wsman:Auth
Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual"/>
```

Similarly, if the service knows how to retrieve a proper username and password for event delivery, simple HTTP Basic or Digest authentication can be used:

```
<s:Body>
  <wse:Subscribe>
    <wse:Delivery>
      <wse:NotifyTo>  HTTP address </wse:NotifyTo>
      <wsman:Auth
Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest"/>
```

There is no requirement that the service support any specific profile. The rest of this section defines special-case profiles for event delivery in which the service needs additional information to select the proper credentials to use when delivering events.

9.4 Including Credentials with a Subscription

In addition to specifying the authentication profile using the wsman:Auth block, the subscriber may wish to send additional tokens to the service to indicate which credentials to use when making the connection to deliver events. As stated in the previous section, if no tokens are specified, the service must be preconfigured to know which credentials to use. However, the service may 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 username, password, or certificate identity could be used with a variety of transports.

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. Using other mechanisms introduces implementation-specific behavior and makes writing compatible client-side implementations more difficult.

In the following example, the user name token is conveyed in the headers to the wse:Subscribe message in a wst:IssuedTokens block (lines 9–27):

```
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-token-profile-
14  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>
```

This wst:IssuedTokens block is divided into three sections:

- The type of token or credential being passed: the wst:TokenType wrapper on lines 9–11. This may refer to user names, X.509 certificates, or other token types.
- The actual security token in a wst:RequestedSecurityToken wrapper (lines 13–17).

- What the tokens apply to: the wsp:AppliesTo block from WS-Policy. In this case, the tokens apply to the wse:NotifyTo address in the subscription. The wse:NotifyTo EPR and the wsp:AppliesTo MUST be identical.

Note that the wst:IssuedTokens block must 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 will pass one of the following to the service using WS-Trust:

- A username reference (the service must know the password or other related credentials and only uses the username as a hint to know which credential to use).
- An X.509 certificate identifier (thumbprint or "hash"). The service has more than one certificate to use and needs to know which one.
- A username and password combination (rarely) which will be directly used to make the connection in the other direction at event-time. This has security implications and should not be delivered to the service over an unencrypted network transport.
- Some combination of the above token types (such as a username 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 may be communicated as well, but they are beyond the scope of this specification.

R9.4-1: Whenever a user name is communicated to the service, the following WS-Trust usage SHOULD be observed. The wst:TokenType MUST 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 must be a wsse:UsernameToken which contains the user name:

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

The wsse:UsernameToken is defined in **Web Services Security Username Token Profile 1.0 [20].** WS-Management does not require the use or presence of the other fields in wsse:UsernameToken, although the implementation should return appropriate errors if other submitted fields are not supported, such as wsse:Nonce.

The password may be optionally supplied in clear text as specified in the above specification, but it should be delivered over an encrypted transport:

```
<wst:RequestedSecurityToken>
   <wsse:UsernameToken>
```

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

R9.4-2: Whenever an X.509 certificate identity is communicated to the service, the following WS-Trust usage SHOULD be observed. The wst:TokenType MUST be the following URI:

```
  <wst:TokenType>
     http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#x509v3
  <wst:/TokenType>
```

The wst:RequestedSecurityToken MUST be a wsse:BinarySecurityToken with a ValueType of **wsse:X509v3** using a standard encoding type of **wsse:Base64Binary:**

```
  <wst:RequestedSecurityToken>
     <wsse:BinarySecurityToken
     ValueType=
  "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#x509v3"
     EncodingType=
  "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
  1.0#Base64Binary">
        MIEZzCCA9CgAwIBAgIQEmtJZc0BAgIQEmtJZc0
  </wsse:BinarySecurityToken>
  </wst:RequestedSecurityToken>
```

This token type is defined in the **Web Services Security X.509 Certificate Token Profile [21]**.

R9.4-3: A conformant service which accepts username, password, or X.509 certificate references within a wse:Subscribe message for later use in event delivery MUST at least support the mechanisms in R9.15-1 and R9.15-2, and MAY support any additional mechanisms.

While WS-Trust and the standard WS-Security profiles referenced above provide other options and mechanisms, their use is optional and beyond the scope of this version of WS-Management.

9.5 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.

For 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>
  ...
```

This definition requires that the service include the MyNamespace:uuid value as a SOAP header with each delivery (see section 2.1). The service can use this 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 if the event delivery is authorized or not, even though the connection itself may have been authenticated.

This mechanism still may 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 should receive at least one unique reference parameter which is never reused, such as the illustrate uuid, for this mechanism to be of value.

Other reference parameters may of course be present to help route and correlate the event delivery as required by the subscriber.

9.6 Transport-Level Authentication Failure

Since transports typically go through their own authentication mechanisms prior to any SOAP traffic occurring, the first attempt to connect may 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.

9.7 Security Implications of Third-Party Subscriptions

Without proper authentication and authorization, WS-Management implementations may be vulnerable to distributed denial-of-service attacks through third party subscriptions to events. WS-Management implementations should be mindful of the discussion in section 6.2 ("Access Control") of the WS-Eventing specification to address this vulnerability.

10 Transports and Message Encoding

10.1 Introduction

This section covers encoding rules which apply to all transports.

10.2 SOAP

R10.2-1: A SERVICE MUST at least receive and send SOAP 1.2 [SOAP 1.2] SOAP ENVELOPEs.

R10.2-2: A SERVICE MAY reject a TEXT SOAP ENVELOPE with more than 32,767 octets.

R10.2-3: A SERVICE SHOULD NOT send a TEXT SOAP ENVELOPE with more

than 32,767 octets unless the client has specified a wsman:MaxEnvelopeSize header overriding this limit.

Large SOAP ENVELOPEs are expected to be serialized using attachments.

R10.2-4: Any REQUEST MESSAGE MAY be encoded using either UNICODE 3.0 (UTF-16) or UTF-8 encoding. A service MUST accept either encoding type for all operations and emit RESPONSES using the same encoding as the original request.

Some SOAP-enabled systems only have UNICODE available, and some only have UTF-8. To maximize interoperation, a server can easily support both encodings, since R10.3-5 places limits on the required character set.

R10.2-5: A service IS REQUIRED to support characters from U+0000 to U+007F inclusive with both UTF-8 and UTF-16 encodings, and MAY support characters outside this range. If the message contains unsupported characters above U+007F, the service MUST return a wsman:EncodingLimit fault.

R10.2-6: For UTF-8 encodings, the service MAY fail to process any message beginning with the UTF-8 BOM (0xEF 0xBB 0xBF) at the beginning of the message and MUST send UTF-8 responses without the BOM.

The presence of BOM in 8-bit characters encodings reduces interoperation. Where extended characters are a requirement, UTF-16 SHOULD be used.

Since the only required subrange is U+0000 to U+007F, supporting both UTF-16 and UTF-8 encoding for characters is trivial since every other octet in UNICODE UTF-16 is a zero.

R10.2-7: If UTF-16 is the encoding, the service MUST 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 http://www.unicode.org/faq/utf_bom.html#BOM.)

R10.2-8: Duplicate headers SHOULD NOT be processed. The service SHOULD issue a wsa:InvalidMessageInformationHeaders fault if they are detected. However, a conformant service MAY ignore any duplicate headers if it assumes the first occurrence is the valid one.

Duplicate headers are considered a defect originating on the client side of the conversation. Returning a fault helps identify faulty clients. However, an implementation may be resource-constrained and unable to detect duplicate headers, so they may be ignored.

R10.2-9: 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 a detail code of http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Whitespace.

Clients should not send messages with leading or trailing whitespace in the values, but services should eliminate unneeded "cosmetic" whitespace on both sides of the element value without faulting.

(See also http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#dt-whiteSpace.)

R10.2-10: Services SHOULD NOT fault messages containing XML comments, as this is part of the XML standard. Services MAY emit messages containing comments relating to the origin and processing of the message or add comments for debugging purposes.

10.3 Lack of Response

If an operation succeeds, but a response cannot be computed or actually delivered due to runtime difficulties or transport problems, no response should be sent and the connection should be terminated.

Specific transports may have specific techniques for terminating the connection; for example, see R10.2-9.

This behavior is preferable to attempting a complex model for sending responses in a delayed fashion. Implementations should 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 they fail to get a response. The format and behavior of such a log is beyond the scope of this specification. Since, in any case, the client must be coded to take into account a lack of response, all abnormal message conditions can safely revert to this scenario.

R10.4-1: If correct responses or faults cannot be computed or generated due to internal service failure, a response to any operation SHOULD NOT be sent.

Regardless, the client has to deal with cases of no response, so the service should simply force the client into that mode rather than send a response or fault not defined in this specification.

10.4 Replay of Messages

A service should not resend messages which have not been acknowledged at the transport level.

R10.4-1: A service MUST 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.

10.5 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. In addition to any requirements or limits established by the WS-I Basic Profile, the service should observe the following:

R10.5-1 A service MAY fail to process any URI with more than 2048 characters.

R10.5-2: A service SHOULD NOT generate a URI with more than 2048 characters.

R10.5-3: A service MAY fail to process an Option Name of more than 2048 characters.

R10.5-4: A service MAY fail to process an Option value of more than 4096 characters.

R10.5-6: A service MAY fault any operation that would require a single reply exceeding 32,767 octets.

R10.5-7: 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 should always be prepared for this minimum in case of an error.

R10.5-8: When the default addressing model is in use, a service MAY fail to process a Selector Name of more than 2048 characters.

10.6 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:

R10.6-1: A conformant service MAY OPTIONALLY support binary attachments to any operation using the SOAP MTOM proposal (http://www.w3.org/TR/2004/PR-soap12-mtom-20041116).

R10.6-2: If a service supports attachments, the service MUST support the Abstract Transmission Optimization Feature.

R10.6-3: If a service supports attachments, the service MUST support the Optimized MIME Multipart Serialization Feature.

Other attachment types are not prohibited. Specific transports may impose additional encoding rules.

10.7 Case-Sensitivity

While XML and SOAP are intrinsically case-sensitive with regard to schematic elements, WS-Management will service many underlying systems which are not intrinsically case-sensitive. This primarily applies to values, but may also apply to schemas which are automatically and dynamically generated from other sources.

A service may observe any case usage required by the underlying execution environment.

The only requirement is that messages must be able to pass validation tests against any schema definitions. At any time, a validation engine may possibly 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 should be retained throughout the lifetime of that message.

11 Faults

11.1 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 will occur.

Only SOAP 1.2 faults (or later) should be supported.

Generally, faults should not be issued unless they are expected as part of a call-response pattern. It would not be valid for a client to issue a wxf:Get, receive the wxf:GetResponse, and then *fault* that response.

11.2 Fault Encoding

This section discusses XML fault encoding.

R11.2-1: A conformant service MUST use the fault encoding format and normative constraints defined below for faults in the WS-Management space or any of its dependent specifications:

```
<s:Envelope
      xmlns:s="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <s:Header>
     <wsa:Action>
        http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
     <wsa:Action>
     <wsa:MessageID>
        uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
     </wsa:MessageID>
     <wsa:RelatesTo>
        uuid:d9726315-bc91-430b-9ed8-ce5ffb858a85
     </wsa:RelatesTo>
  </s:Header>

  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value> [Code] </s:Value>
        <s:Subcode>
          <s:Value> [Subcode] </s:Value>
        </s:Subcode>
      </s:Code>
      <s:Reason>
         <s:Text xml:lang="en">  [Reason] </s:Text>
      </s:Reason>
      <s:Detail>
         [Detail]
      </s:Detail>
   </s:Fault>
  </s:Body>
```

R11.2-2: The following describes additional, normative constraints on the outline listed above:

s:Envelope/s:Header/wsa:Action

MUST be a valid fault Action URI from the relevant specification which defined the fault.

s:Envelope/s:Header/wsa:MessageId

MUST be present for the fault, like any non-fault message.

s:Envelope/s:Header/wsa:RelatesTo
MUST, like any other reply, contain the MessageID of the original request which caused the fault.

s:Body/s:Fault/s:Value
MUST be either s:Sender or s:Receiver, as specified in the Master Fault Table under the "Code" entry.

s:Body/s:Fault/s:Subcode/s:Value
For WS-Management-related messages, MUST be one of the subcode QNames defined in the Master Fault Table. If the service exposes custom methods or other messaging, this of course may be another QName not in the Master Fault Table.

s:Body/s:Fault/s:Reason
This OPTIONAL element SHOULD contain localized text explaining the fault in more detail. Typically, this is extracted from the "Reason" field of the Master Fault Table. However, the text may be adjusted to reflect a specific circumstance. This element may be repeated for each language. Note that the xml:lang attribute MUST be present.

s:Body/s:Fault/s:Detail
This OPTIONAL element SHOULD reflect the RECOMMENDED content from the Master Fault Table.

The above fault template is populated by examining entries from the Master Fault Table in section 11.6, which includes all relevant faults from WS-Management and its underlying specifications.

Note that s:Reason and s:Detail are always optional, but recommended, and that s:Reason must contain an xml:lang attribute to indicate the language used in the descriptive text.

R11.2-3: Note that fault wsa:Action URI values vary from fault to fault. The service MUST issue a fault using the correct URI, based on the specification which defined the fault. Faults defined in this specification must have the URI value:

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

The Master Fault Table in section 11.6 contains the relevant wsa:Action URIs which apply. The URI values are directly implied by the QName for the fault.

11.3 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 section 11.2. See 5.4.8 in reference **[4]**. In practice, the fault only varies in indicating the SOAP header that was not understood, the QName, and the namespace (line 3):

```
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        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>s:MustUnderstand</s:Value>
     </s:Code>
     <s:Reason>
        <s:Text xml:lang="en-US">Header not understood</s:Text>
     </s:Reason>
    </s:Fault>
  </s:Body>

  </s:Envelope>
```

The fault template shown above may 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 specifying that the current message is a fault.

The wsa:RelatesTo should be included so that the client can correlate the fault with the original request. Over transports other than HTTP in which requests may be interlaced, this may 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 MAY attempt to send back a fault without the wsa:MessageID and wsa:RelatesTo fields, or may simply fail to respond, as discussed in section 11.4.

11.4 Degenerate Faults

In rare cases, the SOAP message may 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 which references the original message. Some transports may not be able to reference the sender to return the fault.

If the transport guarantees a simple request-response pattern, then the service MAY 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, the wsa:FaultTo contains an invalid address, so there is no way to deliver the fault).

In all cases, the service SHOULD revert to the rules of section 10.4, in which no response is sent. The service SHOULD attempt to log the requests in some way to help identify the defective client.

11.5 Fault Extensibility

A service may 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 may appear at any location within this element, provided that it is properly mapped to an XML namespace which defines the schema for that content. WS-Management makes use of this extension technique for the wsman:FaultDetail URI values:

```
(1) <s:Detail>
(2)   <wsman:FaultDetail>... </wsman:FaultDetail>
(3)   <ExtensionData xmlns="vendor-specific-namespace">...</ExtensionData>
(4)    ...
(5) </s:Detail>
```

The extension data elements may appear before or after any WS-Management-specific extensions mandated by this specification. More than one extension element is permitted.

11.6 Master Fault Table

This section includes all faults from this specification and all underlying specifications. This list should be taken as the normative fault list for WS-Management.

R11.6-1: A service MUST 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 which 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", constructing interoperable clients will be impossible. In the tables that follow, 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.

11.6.1 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 is returned generically for all access denials relating to authentication or authorization failures. This should not be used to indicate locking or concurrency conflicts or other types of denials unrelated to security by itself.
Applicability	Any message
Remedy	Client must acquire the correct credentials and retry the operation.

11.6.2 wsa:ActionNotSupported

Fault Subcode	wsa:ActionNotSupported
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	The action is not supported by the service.
Detail	<s:Detail> <wsa:Action> *Incorrect Action URI* </wsa:Action> </s:Detail> <!-- The unsupported Action URI is returned, if possible -->

Comments	This means that the requested action is not supported by the implementation. As an example, read-only implementations (supporting only wxf:Get, wsen:Enumerate) will return this for any other operations besides these two. If the implementation never supports the action, the fault should 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 may be added to the fault, as follows: <s:Detail> <wsa:Action> *The offending Action URI* </wsa:Action> <wsman:FaultDetail> **http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ActionMismatch** </wsman:FaultDetail> </s:Detail> This situation can occur when the implementation supports wxf:Put, for example, but the client attempts to update a read-only resource.
Applicability	All messages
Remedy	Client must consult metadata provided by the service to determine which operations are supported.

11.6.3 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 which already exists.
Detail	none
Comments	This is returned in cases where the user attempted to create a resource which already exists.
Applicability	wxf:Create
Remedy	Client use wxf:Put or create a resource with a different identity.

11.6.4 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> <s:Text xml:lang="en"> Explanation of why filter cannot be processed </s:Text> </s:Detail>

Comments	This is typically returned for syntax errors or other semantic problems with the filter. 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 should be returned, such as wsman:QuotaLimit or wsman:InternalError.
Applicability	wsen:Enumerate
Remedy	Client fixes the filter problem and tries again.

11.6.5 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	
Comments	This means that the requested action could not be carried out due to either internal concurrency or locking problems or because another user is accessing the resource. This may 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	Client must wait and retry.

11.6.6 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> <wse:SupportedDeliveryMode>... </wse:SupportedDeliveryMode> <wse:SupportedDeliveryMode>...</wse:SupportedDeliveryMode> ... </s:Detail> <!-- This is a simple, optional list of one or more supported delivery mode URIs. It may be left empty. -->
Comments	This is returned for unsupported delivery modes for the specified resource. If the stack supports the delivery mode in general, but not for the specific resource, this fault is still returned. Other resources may support the delivery mode. The fault does not imply that the delivery mode is not supported by the implementation.

Applicability	wse:Subscribe
Remedy	Client should select one of the supported delivery modes.

11.6.7 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 is returned by event receivers to force a cancellation of a subscription. This 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 it no longer owns. This can help cleanup 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 should cease delivering events for the subscription and cancel the subscription, sending any applicable wse:SubscriptionEnd messages.

11.6.8 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.
Detail	<s:Detail> <s:Text xml:lang="en"> Explanation of why endpoint cannot be reached </s:Text> <!-- The following elements are optional --> <wsman:FaultDetail> one of the URI values below </wsman:FaultDetail> ...any service-specific additional XML content... </s:Detail> When the default addressing model is in use, the wsman:FaultDetail field may contain **http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI.**

Comments	This is returned as the general "Not Found" case for a resource, in which the resource endpoint reference cannot be mapped to the real-world resource. This fault is NOT used merely to indicate the resource is temporarily offline, which is indicated by wsa:EndpointUnavailable.
Applicability	All request messages
Remedy	Client should attempt to diagnose the version of the service, query any metadata, and perform other diagnostic operations to determine why the request cannot be routed.

11.6.9 wsman:EncodingLimit

Fault Subcode	wsman:EncodingLimit
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	An internal encoding limit was exceeded in a request or would be violated if the message were processed.
Detail	<s:Detail> <wsman:FaultDetail> Optional; one of the enumeration values below </wsman:FaultDetail> <s:Text> Optional text description of the limit violation </s:Text> ...any service-specific additional XML content... </s:Detail>

Possible enumeration values in the <wsman:FaultDetail> element:

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/URILimitExceeded
(URI was too long)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize
(The requested maximum was too large)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSizeExceeded
(The computed response is too large based on the client limit, but operation was read-only or never executed to start with)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit
(Service reached its own internal limit when computing response)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/SelectorLimit
(Used when the default addressing format is in use and indicates that too many Selectors were used for the corresponding ResourceURI)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionLimit
(Too many Options)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/CharacterSet
(Unsupported character set)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess
(Operation succeeded and cannot be reversed, but result is too large to send)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Whitespace
(Client-side whitespace usage is not supported)

http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EncodingType
(Used for unsupported MTOM or other encoding types)

Comments	This is returned when a system limit is exceeded, whether a published limit or a service-specific limit.
Applicability	All request messages
Remedy	Client should be reconfigured to send messages which fit the encoding limits of the service.

11.6.10 wsa:EndpointUnavailable

Fault Subcode	wsa:EndpointUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Receiver
Reason	The specified endpoint is currently unavailable.
Detail	<s:Detail> <wsa:RetryAfter> *xs:duration* </wsa:RetryAfter> <!-- optional --> ...optional service-specific XML content <wsman:FaultDetail> *A detail URI value* </wsman:FaultDetail> </s:Detail> In cases where the default addressing model is in used, the following fault detail codes may be returned: **http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI** Used when the ResourceURI is not known. **http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues**

	Used when the Selector values do not match a known resource.
Comments	This is returned if the message was correct and the EPR was valid, but the specified resource is offline. In practice , it is difficult for a service to distinguish between "Not Found" cases and "Offline" cases. In general, wse:DestinationUnreachable is preferable.
Applicability	All request messages
Remedy	Client can retry later, after the resource is again online.

11.6.11 wse:EventSourceUnableToProcess

Fault Subcode	wse:EventSourceUnableToProcess
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The event source cannot process the subscription.
Detail	<s:Detail> <s:Text> Text description of why subscription cannot be processed </s:Text> <wsman:FaultDetail> .... </wsman:FaultDetail> ...any service-specific additional XML content... </s:Detail> The wsman:FaultDetail code may be set to **http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnusableAddress.**
Comments	This should be limited to cases where the event filter contains syntax or semantic errors, or the wse:NotifyTo address is not usable because it is incorrect or permissions cannot be acquired for event delivery. It should *not* be used to report other internal failures such as resource limits, internal service errors, "Server Busy", "Access Denied", or any other more specific faults which provide more information to the client.
Applicability	wse:Subscribe
Remedy	Client should repair the filter syntax.

11.6.12 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> <wsen:SupportedDialect> .... </wsen:SupportedDialect> + </s:Detail>
Comments	This is returned when the client requests a filter type or query language not supported by the service. The filter dialect may vary from resource to resource, or may apply to the entire service.
Applicability	wsen:Enumerate
Remedy	Client must switch to a supported dialect or perform a simple enumeration with no filter.

11.6.13 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	Returned when the service does not support filtered subscriptions for the specified event source, but only supports simple delivery of all events for the resource. Note that the service may support filtering over a different event resource, or may not support filtering for *any* resource. The same fault applies.
Applicability	wse:Subscribe
Remedy	Client must subscribe using unfiltered delivery.

11.6.14 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	
Comments	Returned when the service does not support filtering of enumerations at all, but only supports simple enumeration. If enumeration as a whole is not supported, then the correct fault is wsa:ActionNotSupported. Note that the service may support filtering over a different enumerable resource, or may not support filtering for *any* resource. The same fault applies.
Applicability	wsen:Enumerate
Remedy	Client must switch to a simple enumeration.

11.6.15 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> <wse:SupportedDialect>.. </wse:SupportedDialect> + <wsman:FaultDetail> ..the URI below, if applicable </wsman:FaultDetail> </s:Detail> Possible URI value: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired
Comments	This is returned when the client requests a filter dialect not supported by the service. In some cases, a subscription *requires* a filter because the result of an unfiltered subscription may be infinite or extremely large. In these cases, the http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired needs to be included in the s:Detail element.
Applicability	wse:Subscribe
Remedy	Client must switch to a supported filter dialect or use no filtering.

11.6.16 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> <s:Text> <!-- Text description of internal failure or system-specific error codes and text --> </s:Text> ...service-specific extension XML elements.... <s:Detail>
Comments	This 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 must be reconfigured, restarted, or reinstalled, so merely asking the client to retry will not succeed.
Applicability	All messages

Remedy	Client must repair the service out-of-band to WS-Management.

11.6.17 wsman:FragmentDialectNotSupported

Fault Subcode	wsman:FragmentDialectNotSupported
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Receiver
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 above optional URI values indicate supported dialects.*
Comments	This 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 may 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	Client must use a supported filtering dialect or no filtering.

11.6.18 wsman:InvalidBookmark

Fault Subcode	wsman:InvalidBookmark
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The bookmark supplied with the subscription is not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values:

	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired
	http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Invalid
Comments	This is returned if a bookmark has expired, is corrupt, or is otherwise unknown. If the service cannot detect "Expired" bookmarks, "Invalid" may always be returned.
Applicability	wsen:Subscribe
Remedy	Client must issue a new subscription without bookmarks at all or locate the correct bookmark.

11.6.19 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 will happen with a wsen:Pull. The enumeration context may be invalid due to expiration, an invalid format, or reuse of an old context no longer being tracked by the service. The service also can return this for any case where the enumerator has been terminated unilaterally on the service side, although one of the more descriptive faults is preferable, since 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	Client must abandon the enumeration and let the service time it out, as wsen:Release will fail as well.

11.6.20 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	Expiration time is not valid at all or within the limits of the service. Used for outright errors (expirations in the past, etc.) or expirations too far into the future. If the service does not support expiration times at all, then a wsman:UnsupportedFeature fault should be returned with the correct detail code.
Applicability	wse:Subscribe

Remedy	Client issues a new subscription with a supported expiration time.

11.6.21 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	Since WS-Management recommends against implementing the wsen:Expiration feature, this fault should not occur with most implementations. Consult the WS-Enumeration specification for more information.
Applicability	wsen:Enumerate
Remedy	N/A

11.6.22 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	<s:Detail> <s:Text> ...identify the problem either with text or XML fragments </s:Text> </s:Detail>
Comments	Generally not used in WS-Management, although it MAY be used for cases not covered by other faults. If the content violates the schema, a wsman:SchemaValidationError fault should be sent. If specific errors occur in the subscription body, one of the more descriptive faults should be used. This should not be used to indicate unsupported features, only unexpected or unknown content in violation of this specification.
Applicability	WS-Eventing request messages
Remedy	Client has a defect which should be corrected to issue valid messages which comply with this specification.

11.6.23 wsa:InvalidMessageInformationHeader

Fault Subcode	wsa:InvalidMessageInformationHeader
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	A message information header is not valid and the message cannot be processed.

Detail	<s:Detail> ...the invalid header... </s:Detail>
Comments	This may occur with any type of SOAP header error. The header may be invalid in terms of schema or value, or may constitute a semantic error. This should not be used to indicate an invalid resource address (a "not found" condition for the resource), but to indicate actual structural violations of the SOAP header rules in this specification. Examples are repeated MessageIDs, missing RelatesTo on a response, badly formed addresses, or any other missing header content.
Applicability	All messages
Remedy	Major client defect. The SOAP packets are not correctly formed.

11.6.24 wsman:InvalidOptions

Fault Subcode	wsman:InvalidOptions
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	One or more options are not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue
Comments	This 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	Client should retrieve the catalog entry for the resource and determine how to correct the invalid option values.

11.6.25 wsman:InvalidParameter

Fault Subcode	wsman:InvalidParameter
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender

Reason	An operation parameter is not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName
Comments	Returned when a parameter to a custom action is not valid. This is a default for new implementations which need to have a generic fault for this case. The method may also return any specific fault of its own.
Applicability	All messages with custom actions
Remedy	Client should consult the WSDL for the operation and determine how to supply the correct parameter.

11.6.26 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	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFragment
Comments	This may be returned when the input XML is not valid semantically or uses the wrong schema for the resource. However, a wsman:SchemaValidationError fault should be returned if the error is related to XML schema violations as such, as opposed to invalid semantic values. Note the anomalous case where a schema violation does not occur, but the

	namespace is simply the wrong one, in which http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace is returned.
Applicability	wxf:Put, wxf:Create
Remedy	This is a client defect. The client should correct the input XML.

11.6.27 wsman:InvalidSelectors

Fault Subcode	wsman:InvalidSelectors
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The Selectors for the resource are not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelectors http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelectors http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AmbiguousSelectors http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DuplicateSelectors
Comments	This covers all cases where the specified Selectors were incorrect or unknown for the specified resource.
Applicability	All request messages
Remedy	Client should retrieve documentation or metadata and correct the Selectors.

11.6.28 wsa:MessageInformationHeaderRequired

Fault Subcode	wsa:MessageInformationHeaderRequired
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	A required header is missing.
Detail	<s:Detail> The XML QName of the missing header </s:Detail>
Comments	A required message information header, To, MessageID, or Action, is not present.

Applicability	All messages
Remedy	This is a major client defect. The SOAP packets are not correctly formed.

11.6.29 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 is returned when the client (subscriber) receives an event with a wsman:AckRequested header and does not (or cannot) acknowledge the receipt. The service should cease sending events and terminate the subscription.
Applicability	Any event delivery action (including heartbeats, dropped events, etc.) in any delivery mode
Remedy	For subscribers, the subscription must be resubmitted without the acknowledgement option. For services delivering events, the service should cancel the subscription immediately.

11.6.30 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	<s:Detail> <s:Text> reason </s:Text> </s:Detail>
Comments	This is returned when the SOAP message is otherwise correct, but the service has reached a resource or quota limit.
Applicability	All messages
Remedy	Client can retry later.

11.6.31 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	<s:Detail>

	<s:Text> Service-specific error messages about the schema violation. </s:Text> </s:Detail>
Comments	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 may 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 should be returned for *all* errors noted by the validating parser.
Applicability	All messages
Remedy	Client corrects the message.

11.6.32 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 should not be used in WS-Management due to overlap with the wsman:TimedOut, which covers all the other messages.
Applicability	wsen:Pull
Remedy	The client can retry the wsen:Pull.

11.6.33 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 else an internal override timeout was reached by the service while trying to process the request. This is also returned in all enumerations when no content is available for the current wsen:Pull request. Clients may simply retry the wsen:Pull again until a different fault is returned.
Applicability	All requests
Remedy	Client may retry the operation.

If the operation was a write (delete, create, execute), the client should 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.

11.6.34 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	<s:Detail> <s:Text> Optional service-specific error messages about why the Renew failed </s:Text> </s:Detail>
Comments	This is returned in all cases where the subscription cannot be renewed, but is otherwise valid.
Applicability	wse:Renew
Remedy	Client must issue a new subscription.

11.6.35 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. This fault should not be used if the value itself is incorrect, only if the *type* is not supported.
Applicability	wse:Subscribe
Remedy	Client corrects the expiration to use a duration time.

11.6.36 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 may not be supported (as opposed to a duration-based expiration

	type). This fault should not be used if the value itself is incorrect, but only if the *type* is not supported.
Applicability	wsen:Enumerate
Remedy	Client corrects the expiration time or omits it and retries.

11.6.37 wsman:UnsupportedFeature

Fault Subcode	wsman:UnsupportedFeature
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The specified feature is not supported.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AuthorizationMode http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OperationTimeout http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAccess http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DeliveryRetries http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Bookmarks http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatSecurityToken http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AsynchronousRequest http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionSet

Comments	Indicates that an unsupported feature was attempted.
Applicability	any message
Remedy	Client corrects or removes the unsupported feature request and retries.

12 Appendix A: HTTP(S) Transport and Security Profile

12.1 Introduction

While WS-Management is a SOAP protocol and not tied to a specific network transport, interoperation requires some common standards to be established. This specification 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 should also acquire a URI for referencing purposes, and any new authentication mechanisms that they expose should also be assigned URIs for publication and identification purposes in XML documents.

For interoperability, the standard transports are HTTP 1.1 and HTTPS (using TLS 1.0), designated as follows:

- http://www.ietf.org/rfc/rfc2616.txt (HTTP 1.1)
- http://www.ietf.org/rfc/rfc2818.txt (HTTPS)

The requirements mentioned in this section are for ensuring interoperation between WS-Management implementations using HTTP 1.1 and HTTPS transports. As new transports are standardized for WS-Management, the associated transport-specific requirements should be defined and published to ensure interoperability.

The SOAP and HTTP encoding models specified in the following base specifications are used for WS-Management encoding over HTTP and HTTPS:

- **SOAP Version 1.2 Part 2: Adjuncts**, SOAP HTTP binding described in section 7 of http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapinhttp
- **WS-I Basic Profile Version 1.1**
  http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

  R12.1-1: A service SHOULD conform to any encoding requirements in the WS-I Basic Profile Version 1.1, in addition to any requirements in this specification.

This increases the probability of successful interoperation between implementations.

12.2 HTTP(S) Encoding

R12.2-1: A service MUST support **Transfer-Encoding: chunked.**

This requires the service to be able to receive incoming SOAP messages in several parts or deliver them in several parts when they are very large or the size is unknown. The limits are service-specific.

R12.2-2 A service MUST at least support the SOAP HTTP binding.

R12.2-3: A service MUST 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**/).

R12.2-4 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**/).

R12.2-5: A service MAY choose not to support the SOAP Web Method Feature.

R12.2-6: A service MUST 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).

This is used to carry SOAP messages which require no response.

R12.2-7: A service MUST at least support Request Message SOAP ENVELOPEs and one-way SOAP ENVELOPEs delivered using HTTP POST.

R12.2-8: 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.

R12.2-9: For services which support HTTPS (TLS 1.0), the service MUST at least implement TLS_RSA_WITH_RC4_128_SHA. It is RECOMMENDED that the service also support TLS_RSA_WITH_AES_128_CBC_SHA.

R12.2-10: 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.

R12.2-11: It is NOT a REQUIREMENT that the URL used with the HTTP-POST operation to deliver the SOAP message have the same content as the wsa:To URI used in the SOAP addressing. 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.

R12.2-12: If the SOAPAction header is present in an HTTP/HTTPS-based request which 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.

Since WS-Management is based on SOAP 1.2, the optional SOAPAction header is merely used as an optimization. If present, it MUST 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.

R12.2-13: If a service supports attachments, the service MUST support the HTTP Transmission Optimization Feature.

R12.2-14: If a service cannot process a message with an attachment or unsupported encoding type, and the transport is HTTP or HTTPS, it MUST return HTTP error 415 as its response (unsupported media).

R12.2-15: 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 a detail code of wsman:EncodingType.

12.3 IPSec and HTTP

A special note regarding the use of **IPSec** should be made. While HTTP with Basic authentication is weak on an unsecured network, if IPSec is in use, this is no longer an issue. IPSec provides high-quality cryptographic security, data origin authentication, and anti-replay services.

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/rechallenge** (using HTTPS) is equivalent to simple **http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic** or **http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest** if the traffic is actually secured by IPSec.

Because IPSec is intended for machine-level authentication and network traffic protection, it is insufficient for real-world management in many cases, which may 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.

12.4 HTTP(S) Security Profiles

This specification does not mandate that conformant services must provide HTTP or HTTPS-based access. However, if HTTP or HTTPS is used, it does mandate that at least one of the predefined profiles for that transport must be supported so that clients can reliably access the service.

R12.4-1: A conformant service which supports HTTP MUST support one of the predefined HTTP-based profiles.

R12.4-2: A conformant service which supports HTTPS MUST support one of the predefined HTTPS-based profiles.

R12.4-3: A conformant service MUST NOT expose WS-Management over a completely unauthenticated HTTP channel.

There is no requirement that the service only export a single HTTP or HTTPS address. The service may 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 which supports HTTP and/or HTTPS.

12.5 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic

This essentially the "standard" profile, but limited to Basic authentication.

The typical sequence is:

	Client		Service
1	Client connects with no auth header.	➔	Service sees no header.
2		←	Service sends 401, listing Basic as the auth mode.
3	Client provides Basic authorization header.	➔	Service authenticates the client.

This is normal behavior for HTTP. If the client connects with a Basic authorization header initially and it is valid, then of course 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, then the risk is substantially reduced.

Similarly, Basic authentication is suitable when performing testing, prototyping, or diagnosis.

12.6 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest

This essentially the same as the "standard" profile, but limited to the use of Digest authentication.

The typical sequence is:

	Client		Service
1	Client connects with no auth header.	➔	Service sees no header.
2		←	Service sends 401, listing Digest as the auth mode.
3	Client provides Digest authorization header.	➔	
4		←	Service begins auth sequence of secure token exchange.
5	Client continues auth sequence....	➔	Service authenticates client.

This is normal behavior for HTTP. If the client connects with a Digest authorization header initially and it is valid, then the token exchange sequence begins.

12.7 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/basic

This profile establishes the use of Basic authentication over HTTPS. This is used when only a server-side certificate encrypts the connection, but the service still needs to authenticate the client.

The typical sequence is:

	Client		Service
1	Client connects with no auth header using HTTPS.	➔	Service sees no header, but establishes an encrypted connection.
2		➔	Service sends 401, listing Basic as the auth mode.
3	Client provides Basic authorization header.	➔	Service authenticates the client.

If the client connects with a Basic authorization header initially and it is valid, then of course the request immediately succeeds.

12.8 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/digest

This profile establishes the use of Digest authentication over HTTPS. This is used when only a server-side certificate encrypts the connection, but the service still needs to authenticate the client.

The typical sequence is:

	Client		Service
1	Client connects with no auth header using HTTPS.	➔	Service sees no header, but establishes an encrypted connection.
2		➔	Service sends 401, listing Digest as the auth mode.
3	Client provides Digest authorization header.	➔	
4		➔	Service begins auth sequence of secure token exchange.
5	Client continues auth sequence....	➔	Service authenticates client.

This is normal behavior for HTTP. If the client connects with a Digest authorization header initially and it is valid, then the token exchange sequence begins.

12.9 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(S) authorization header is required in the HTTP POST request.

However, as a hint to the service, the following HTTP(S) authorization header may be present:

```
Authorization: http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual
```

Since the service can be configured to always look for the certificate, this is not required.

The sequence is simple:

	Client		Service
1	Client connects with no auth header but supplies an X.509 cert.	➔	Service ignores the authorization header and retrieves the client-side cert used in the TLS 1.0 handshake.
2		←	Service accepts or denies access with 403.7 or 403.16, etc.

12.10 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 may act as a simple, secondary authentication mechanism with no authorization semantics.

	Client		Service
1	Client connects with cert and special auth header.	➔	Service queries for client cert and authenticates. If cert 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, listing available Basic auth mode as a requirement.
3	Client selects Basic as the auth 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 is 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), since the transmission of the username and password is performed over a TLS 1.0 encrypted connection.

12.11 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/digest

This 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
```

12.12 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 the internet informational draft **[22]**, which describes the use of GSSAPI SPNEGO over HTTP. This mechanism allows HTTP to carry out the negotiation protocol of RFC 2478 to authenticate the user based on Kerberos Version 5.

	Client		Service
1	Client connects with no auth header using HTTPS.	➔	Service sees no header, but establishes an encrypted connection.
2		←	Service sends 401, 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.

12.13 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. See **[22]** for details.

	Client		Service
1	Client connects with no auth header using HTTPS.	➔	Service queries for client cert and authenticates. If cert 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, 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.

12.14 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/spnego-kerberos

This 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 **[22]** for details.

While 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 should not be used in environments with a requirement for a high level of security.

13 Appendix B: XPath Support

Implementations will typically need to support XPath for several purposes, such as Fragment Transfer (section 4.8), WS-Enumeration (section 5.0), and WS-Eventing filters (section 7.2.2). Since the full XPath 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 should provide to promote maximum interoperability. In most cases, implementations will provide large subsets of full XPath, but need additional definitions to ensure that the subset meets minimum requirements. The Level 1 and Level 2 BNF definitions below 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 section 4.8), 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 which meet the syntax requirements should provide a mode in which the elements are processed without regard to XML namespaces, but may of course provide more powerful, namespace-aware processing.

The default execution context of the XPath is specified explicitly for WS-Enumeration in section 5.3 of this specification, and in WS-Eventing subscription filters in section 7.2.2.

For the following dialects, XML namespaces and QNames are not expected to be supported by default and may be silently ignored by the implementation.

These are for informational purposes only and SHOULD NOT be identified as Filter Dialects in actual SOAP messages. Since 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

13.1 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 (section 4.8) of this specification:

```
<path> ::= <root_selector> TOKEN_END_OF_INPUT;

<root_selector> ::= TOKEN_SLASH <element_sequence>;
<root_selector> ::= <attribute>;
<root_selector> ::= <relpath> <element_sequence>;
<root_selector> ::=  TOKEN_DOT

<relpath> ::= <>;
<relpath> ::= TOKEN_DOT TOKEN_SLASH;
<relpath> ::= TOKEN_DOT_DOT TOKEN_SLASH;

<element_sequence> ::= <element> <optional_filter_expression> <more>;

<more> ::= TOKEN_SLASH <follower>;
<more> ::= <>;

<follower> ::= <attribute>;
<follower> ::= <text_function>;
<follower> ::= <element_sequence>;

<optional_filter_expression> ::=
  TOKEN_OPEN_BRACKET <filter_expression> TOKEN_CLOSE_BRACKET;

<optional_filter_expression> ::= <>;

<attribute> ::= TOKEN_AT_SYMBOL <name>;

<element> ::= <name>;

<text_function> ::=
  TOKEN_TEXT TOKEN_OPEN_PAREN TOKEN_CLOSE_PAREN;

<name> ::= TOKEN_XML_NAME;

<filter_expression> ::= <array_location>;

<array_location> ::= TOKEN_UNSIGNED_INTEGER;
```

This allows selecting any XML node based on its name or array position, or any attribute by its name. Optionally, the text() function can trail the entire expression to select only the raw value of the name or attribute, excluding the XML attribute or element name wrapper.

Using the following XML fragment, some examples are shown assuming that the element “a” is the context node:

```
(1) <Envelope>
(2)   <Body>
(3)     <a>
(4)       <b x="y"> 100 </b>
(5)       <c>
(6)         <d> 200 </d>
(7)       </c>
(8)       <c>
(9)         <d> 300 </d>
(10)        <d> 400 </d>
(11)      </c>
(12)    </a>
(13)  <Body>
(14)<Envelope>
```

Examples:

```
(15)   .                     // Selects <a> and all its content
(16)   ../a                  // Selects <a> and all its content
(17)   /Envelope/Body/a      // Selects <a> and all its content
(18)   b                     // Selects <b x="y"> 100 </b>
(19)   c                     // Selects both <c> nodes, one after the other
(20)   c[1]                  // Selects <c><d>200</d></c>
(21)   c[2]/d[2]             // Selects <d> 400 </d>
(22)   c[2]/d[2]/text()      // Selects 400
(23)   b/text()              // Selects  100
(24)   b/@x                  // Selects x="y"
```

Note that the only filtering expression capability is an array selection. Also note that XPath can return a node set. In section 4.8 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 (19) 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.

R12.1-1: A service which supports fragment transfer as described in section 4.8 of this specification SHOULD support a subset of XPath at least as powerful as that described in Level 1.

Clearly, the service may expose full XPath 1.0 or any other subset which meets or exceeds the requirements defined here.

R12.1-2: A service which 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, then the array notation MUST be used to distinguish them.

13.2 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 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:

```
<path> ::= <root_selector> TOKEN_END_OF_INPUT;

<root_selector> ::= TOKEN_SLASH <element_sequence>;

<root_selector> ::= <relpath> <element_sequence>;

<root_selector> ::= <attribute>;

<root_selector> ::= TOKEN_DOT;

<relpath> ::= <> ;
<relpath> ::= TOKEN_DOT TOKEN_SLASH;
<relpath> ::= TOKEN_DOT_DOT TOKEN_SLASH;

<element_sequence> ::= <element> <optional_filter_expression> <more>;

<more> ::= TOKEN_SLASH <follower>;

<more> ::= <>;

<follower> ::= <attribute>;

<follower> ::= <text_function>;

<follower> ::= <element_sequence>;

<optional_filter_expression> ::= TOKEN_OPEN_BRACKET <filter_expression> TOKEN_CLOSE_BRACKET;

<optional_filter_expression> ::= <>;

<attribute> ::= TOKEN_AT_SYMBOL <name>;

<element> ::= <name>;

<text_function> ::= TOKEN_TEXT TOKEN_OPEN_PAREN TOKEN_CLOSE_PAREN;

<name> ::= TOKEN_XML_NAME;

<filter_expression> ::= <array_location>;

<array_location> ::= TOKEN_UNSIGNED_INTEGER;


<filter_expression> ::= <or_expression>;

<array_location> ::= UNSIGNED_INTEGER;


 // Next level, simple OR expression

<or_expression> ::= <and_expression> <or_expression_rest>;

<or_expression_rest> ::= TOKEN_OR <and_expression> <or_expression_rest>;

<or_expression_rest> ::= <>;


 // Next highest level, AND expression
```

```
<and_expression> ::= <rel_expression> <and_expression_rest>;
<and_expression_rest> ::= TOKEN_AND <rel_expression> <and_expression_rest>;
<and_expression_rest> ::= <>;

 // Next level of precedence >, <, >=, <=, =, !=
<rel_expression> ::= <sub_expression> <rel_expression_rest>;
<rel_expression_rest> ::= <name> <rel_op> <const>;
<rel_expression_rest> ::= <>;

  // Identifier, literal, or identifier + param_list (function call)
<sub_expression> ::= TOKEN_OPEN_PAREN <filter_expression> TOKEN_CLOSE_PAREN;
<sub_expression> ::= TOKEN_NOT TOKEN_OPEN_PAREN <filter_expression> TOKEN_CLOSE_PAREN;

// Relational operators
<rel_op> ::= TOKEN_GT;    // >
<rel_op> ::= TOKEN_LT;    // <
<rel_op> ::= TOKEN_GE;    // >=
<rel_op> ::= TOKEN_LE;    // <=
<rel_op> ::= TOKEN_EQ;    // =
<rel_op> ::= TOKEN_NE;    // !=
<const> ::=  QUOTE TOKEN_STRING QUOTE;
```

This 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)
(6)  b[.="100" and @x="z"]  // Select <b> if it is 100 and has @x="z"
(7)  c[d="200" or d="300"]  // Select all <c> with d=200 or d=300
(8)
(9)  c[2][not(.="400" or  @x="100")]
(10) // Select second <c> provided that:
(11) //   its value is not 400 and it does not have an attribute x set to 100
(12)
(13) c/d[.="100" or  (@x="400" and .="500")]
(14) // Select <d> provided that:
(15) //   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.

R12.2-1: A service which supports XPath-based filtering dialects as described in this specification SHOULD support a subset of XPath at least as powerful as that described in Level 2.

Clearly, the service may expose full XPath 1.0 or any other subset which 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

14 Appendix C: WS-Management XSD

A normative copy of the XML schemas [XML Schema Part 1, Part 2] for this specification may be retrieved by resolving the XML namespace URIs for this specification (listed in section 1.5).

15 Appendix D: Authors and Acknowledgements

15.1 Additional Contributors

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

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
Bryan Murray, Hewlett-Packard
Brian Reistad, Microsoft
Matthew Senft, Microsoft
Barry Shilmover, Microsoft
Tom Slaight, Intel
Marvin Theimer, Microsoft
Dave Tobias, AMD
John Tollefsrud, Sun
Anders Vinberg, Microsoft

This specification has been validated by an Interoperability Workshop. The details of this workshop can be found here:

http://msdn.microsoft.com/webservices/community/workshops/mgmtinterop052005.aspx

The authors would like to thank the participants for their efforts.

16 Appendix E: References

[1] **HTTP 1.1**
R. Fielding et al, "IETF RFC 2616: Hypertext Transfer Protocol – HTTP/1.1," June 1999.

[2] **HTTPS**
E. Rescorla, "RFC 2818: HTTP over TLS," May 2000.

[3] **RFC 2119**
S. Bradner, "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels," March 1997.

[4] **SOAP 1.2**
M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003.

[5] **WS-I Basic Profile 1.1**
http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

[6] **WS-Addressing**
D. Box et al, "Web Services Addressing (WS-Addressing),"August 2004.

[7] **WS-Transfer**
J. Alexander et al, "Web Services Transfer (WS-Transfer)," March 2006.

[8] **WS-Enumeration**
J. Alexander et al, "Web Services Enumeration (WS-Enumeration)," March 2006.

[9] **WS-Eventing**
D. Box et al, "Web Services Eventing (WS-Eventing)," March 2006.

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

[11] **WS-SecureConversation**
G. Della-Libera et al, "Web Services Secure Conversation Language (WS-SecureConversation)," May, 2004.

[12] **WSDL 1.1**
E. Christensen et al, "Web Services Description Language (WSDL) 1.1," March 2001.

[13] **XML Schema, Part 1**
H. Thompson et al, "XML Schema Part 1: Structures," May 2001.

[14] **XML Schema, Part 2**
P. Biron et al, "XML Schema Part 2: Datatypes," May 2001.

[15] **RFC 3986: Uniform Resource Identifiers (URI): Generic Syntax**
http://www.ietf.org/rfc/rfc3986.txt

[16] **MTOM: SOAP Message Transmission Optimization Mechanism**
http://www.w3.org/TR/soap12-mtom/

[17] **RFC 3066: Tags for the Identification of Languages**
http://www.ietf.org/rfc/rfc3066.txt

[18] **Web Services Security (WS-Security 2004)]**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[19] **Web Services Trust Language February 2005**
http://msdn.microsoft.com/ws/2005/02/ws-trust/

[20] **Web Services Security Username Token Profile 1.0**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

[21] **Web Services Security X.509 Certificate Profile**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

[22] **Kerberos based HTTP Authentication in Windows, Internet-Draft, June 2005**
http://www.ietf.org/internet-drafts/draft-jaganathan-kerberos-http-00.txt

[23] **RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace**
http://www.ietf.org/rfc/rfc4122.txt