
Web Services for Management (WS-Management) Specification

*Spécification des services Web pour le management
(WS-Management)*

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2013

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 17963 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 38, *Distributed application platforms and services (DAPS)*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013



Document Number: DSP0226

Date: 2012-08-28

Version: 1.1.1

Web Services for Management (WS- Management) Specification

Document Type: Specification

Document Status: DMTF Standard

Document Language: en-US

10 Copyright Notice

11 Copyright © 2006–2012 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

12 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
13 management and interoperability. Members and non-members may reproduce DMTF specifications and
14 documents, provided that correct attribution is given. As DMTF specifications may be revised from time to
15 time, the particular version and release date should always be noted.

16 Implementation of certain elements of this standard or proposed standard may be subject to third party
17 patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations
18 to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose,
19 or identify any or all such third party patent right, owners or claimants, nor for any incomplete or
20 inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to
21 any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize,
22 disclose, or identify any such third party patent rights, or for such party's reliance on the standard or
23 incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any
24 party implementing such standard, whether such implementation is foreseeable or not, nor to any patent
25 owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is
26 withdrawn or modified after publication, and shall be indemnified and held harmless by any party
27 implementing the standard from any and all claims of infringement by a patent owner for such
28 implementations.

29 For information about patents held by third-parties which have notified the DMTF that, in their opinion,
30 such patent may relate to or impact implementations of DMTF standards, visit
31 <http://www.dmtf.org/about/policies/disclosures.php>.

32

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013

CONTENTS

34	Foreword.....	7
35	1 Scope.....	10
36	2 Normative References.....	10
37	3 Terms and Definitions.....	12
38	4 Symbols and Abbreviated Terms.....	15
39	5 Addressing.....	16
40	5.1 Management Addressing.....	16
41	5.2 Versions of Addressing.....	25
42	5.3 Requirements for Compatibility.....	25
43	5.4 Use of Addressing in WS-Management.....	27
44	6 WS-Management Control Headers.....	44
45	6.1 wsman:OperationTimeout.....	44
46	6.2 wsman:MaxEnvelopeSize.....	45
47	6.3 wsman:Locale.....	46
48	6.4 wsman:OptionSet.....	47
49	6.5 wsman:RequestEPR.....	50
50	7 Resource Access.....	51
51	7.1 General.....	51
52	7.2 Addressing Uniformity.....	53
53	7.3 Get.....	54
54	7.4 Put.....	55
55	7.5 Delete.....	59
56	7.6 Create.....	61
57	7.7 Fragment-Level Access.....	64
58	7.8 Fragment-Level Get.....	66
59	7.9 Fragment-Level Put.....	67
60	7.10 Fragment-Level Delete.....	70
61	7.11 Fragment-Level Create.....	71
62	8 Enumeration of Datasets.....	73
63	8.1 General.....	73
64	8.2 Enumerate.....	75
65	8.3 Filter Interpretation.....	82
66	8.4 Pull.....	84
67	8.5 Release.....	88
68	8.6 Ad-Hoc Queries and Fragment-Level Enumerations.....	90
69	8.7 Enumeration of EPRs.....	90
70	8.8 Renew.....	92
71	8.9 GetStatus.....	94
72	8.10 EnumerationEnd.....	94
73	9 Custom Actions (Methods).....	95
74	10 Notifications (Eventing).....	96
75	10.1 General.....	96
76	10.2 Subscribe.....	97
77	10.3 GetStatus.....	117
78	10.4 Unsubscribe.....	118
79	10.5 Renew.....	119
80	10.6 SubscriptionEnd.....	120
81	10.7 Acknowledgement of Delivery.....	122
82	10.8 Refusal of Delivery.....	123
83	10.9 Dropped Events.....	124
84	10.10 Access Control.....	125

85	10.11 Implementation Considerations.....	126
86	10.12 Advertisement of Notifications.....	126
87	11 Metadata and Discovery	126
88	12 Security	129
89	12.1 General.....	129
90	12.2 Security Profiles	130
91	12.3 Security Considerations for Event Subscriptions	130
92	12.4 Including Credentials with a Subscription	131
93	12.5 Correlating Events with a Subscription	132
94	12.6 Transport-Level Authentication Failure	132
95	12.7 Security Implications of Third-Party Subscriptions.....	132
96	13 Transports and Message Encoding.....	133
97	13.1 SOAP.....	133
98	13.2 Lack of Response.....	134
99	13.3 Replay of Messages.....	134
100	13.4 Encoding Limits	134
101	13.5 Binary Attachments	135
102	13.6 Case-Sensitivity.....	135
103	14 Faults	136
104	14.1 Introduction	136
105	14.2 Fault Encoding	136
106	14.3 NotUnderstood Faults	137
107	14.4 Degenerate Faults	138
108	14.5 Fault Extensibility	138
109	14.6 Master Faults.....	139
110	ANNEX A (informative) Notational Conventions	160
111	A.1 XML Namespaces	160
112	ANNEX B (normative) Conformance	162
113	ANNEX C (normative) HTTP(S) Transport and Security Profile	163
114	C.1 General.....	163
115	C.2 HTTP(S) Binding	163
116	C.3 HTTP(S) Security Profiles	165
117	C.4 IPSec and HTTP	170
118	ANNEX D (informative) XPath Support	171
119	D.1 General.....	171
120	D.2 Level 1	172
121	D.3 Level 2.....	174
122	ANNEX E (normative) Selector Filter Dialect	177
123	ANNEX F (informative) Identify XML Schema.....	179
124	ANNEX G (informative) Resource Access Operations XML Schema and WSDL	182
125	ANNEX H (informative) Enumeration Operations XML Schema and WSDL	187
126	ANNEX I (informative) Notification OperationsXML Schema and WSDL	196
127	ANNEX J (informative) Addressing XML Schema.....	204
128	ANNEX K (informative) WS-Management XML Schema.....	207
129	ANNEX L (informative) Change Log.....	217
130		

131 Figures

132	Figure 1 – Message Information Header Blocks	20
-----	--	----

133

134 Tables

135	Table 1 – Relationship Type	21
136	Table 2 – Interoperability Requirements	25
137	Table 3 – WSA Versions in Exchanges	26
138	Table 4 – wsa:Action URI Descriptions	42
139	Table 5 – wsman:AccessDenied	139
140	Table 6 – wsa:ActionNotSupported	140
141	Table 7 – wsman:AlreadyExists	140
142	Table 8 – wsman:CannotProcessFilter	141
143	Table 9 – wsman:CannotProcessFilter	141
144	Table 10 – wsman:Concurrency	142
145	Table 11 – wsme:DeliveryModeRequestedUnavailable	142
146	Table 12 – wsman:DeliveryRefused	143
147	Table 13 – wsa:DestinationUnreachable	143
148	Table 14 – wsman:EncodingLimit	144
149	Table 15 – wsa:EndpointUnavailable	145
150	Table 16 – wsman:EventDeliverToUnusable	145
151	Table 17 – wsme:EventSourceUnableToProcess	146
152	Table 18 – wsman:FilterDialectRequestedUnavailable	146
153	Table 19 – wsme:FilteringNotSupported	146
154	Table 20 – wsman:FilteringNotSupported	147
155	Table 21 – wsme:FilteringRequestedUnavailable	147
156	Table 22 – wsman:FragmentDialectNotSupported	148
157	Table 23 – wsman:InternalError	148
158	Table 24 – wsman:InvalidBookmark	149
159	Table 25 – wsman:InvalidEnumerationContext	149
160	Table 26 – wsme:InvalidExpirationTime	150
161	Table 27 – wsman:InvalidExpirationTime	150
162	Table 28 – wsme:InvalidMessage	151
163	Table 29 – wsa:InvalidMessageInformationHeader	151
164	Table 30 – wsman:InvalidOptions	152
165	Table 31 – wsman:InvalidParameter	152
166	Table 32 – wsmt:InvalidRepresentation	153
167	Table 33 – wsman:InvalidSelectors	153
168	Table 34 – wsa:MessageInformationHeaderRequired	154
169	Table 35 – wsman:NoAck	154
170	Table 36 – wsman:QuotaLimit	154
171	Table 37 – wsman:SchemaValidationError	155

172	Table 38 – wsman:TimedOut	155
173	Table 39 – wsman:TimedOut	155
174	Table 40 – wsme:UnableToRenew	156
175	Table 41 – wsme:UnsupportedExpirationType	156
176	Table 42 – wsman:UnsupportedExpirationType	156
177	Table 43 – wsman:UnsupportedFeature	157
178	Table 44 – wsme:UnsupportedExpirationType	158
179	Table 45 – wsman:UnableToRenew	158
180	Table 46 – wsa:InvalidMessage	158
181	Table 47 – wsme:CannotProcessFilter	159
182	Table A-1 – Prefixes and XML Namespaces Used in This Specification	161
183	Table C-1 – Basic Authentication Sequence	165
184	Table C-2 – Digest Authentication Sequence	166
185	Table C-3 – Basic Authentication over HTTPS Sequence	166
186	Table C-4 – Digest Authentication over HTTPS Sequence	167
187	Table C-5 – HTTPS with Client Certificate Sequence	167
188	Table C-6 – Basic Authentication over HTTPS with Client Certificate Sequence	168
189	Table C-7 – SPNEGO Authentication over HTTPS Sequence	169
190	Table C-8 – SPNEGO Authentication over HTTPS with Client Certificate Sequence	169
191	Table D-1 – XPath Level 1 Terminals	173
192	Table D-2 – XPath Level 2 Terminals	175
193		

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013

Foreword

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

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

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

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

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

Acknowledgements

The authors wish to acknowledge the following people.

Chairpersons:

- Josh Cohen – Microsoft
- Larry Lamers (Vice-Chairman) – VMware

Editors:

- Nathan Burkhart – Microsoft
- Doug Davis – IBM
- Raymond McCollum – Microsoft
- Bryan Murray – HP
- Brian Reistad – Microsoft

Authors:

- Akhil Arora – Sun Microsystems
- Vince Brunssen – IBM
- Mark Carlson – Sun Microsystems
- Jim Davis – WBEM Solutions
- Tony Diczio – Oracle
- Mike Dutch – Symantec
- Zulah Eckert – BEA Systems
- George Ericson – EMC
- Wassim Fayed – Microsoft
- Chris Ferris – IBM
- Bob Freund – Hitachi Ltd.
- Eugene Golovinsky – BMC Software
- Yasuhiro Hagiwara – NEC

- 233 • Steve Hand – Olocity
- 234 • Jackson He – Intel
- 235 • David Hines – Intel
- 236 • Reiji Inohara – NEC
- 237 • Christane Kämpfe – Fujitsu-Siemens Computers
- 238 • Paul Knight – Nortel Networks
- 239 • Vincent Kowalski – BMC Software
- 240 • Heather Kreger – IBM
- 241 • Vishwa Kumbalimutt – Microsoft
- 242 • Sunil Kunisetty – Oracle
- 243 • Richard Landau – Dell
- 244 • Paul Lipton – CA
- 245 • James Martin – Intel
- 246 • Milan Milenkovic – Intel
- 247 • Jeff Mischkinsky – Oracle
- 248 • Paul Montgomery – AMD
- 249 • Jishnu Mukurji – HP
- 250 • Alexander Nosov – Microsoft
- 251 • Abhay Padlia – Novell
- 252 • Gilbert Pilz – Oracle
- 253 • Roger Reich – Symantec
- 254 • Larry Russon – Novell
- 255 • Tom Rutt – Fujitsu Ltd.
- 256 • Jeffrey Schlimmer – Microsoft
- 257 • Dr. Hemal Shah – Broadcom
- 258 • Sharon Smith – Intel
- 259 • Enoch Suen – Dell
- 260 • Vijay Tewari – Intel
- 261 • William Vambenepe – HP
- 262 • Andrea Westerinen – CA, Inc.
- 263 • Kirk Wilson – CA, Inc.
- 264 • Dr. Jerry Xie – Intel

Contributors:

- 266 • Paul C. Allen – Microsoft
- 267 • Rodrigo Bomfim – Microsoft
- 268 • Don Box – Microsoft
- 269 • Jerry Duke – Intel
- 270 • David Filani – Intel
- 271 • Kirill Gavrylyuk – Microsoft
- 272 • Omri Gazitt – Microsoft
- 273 • Frank Gorishek – AMD
- 274 • Lawson Guthrie – Intel
- 275 • Arvind Kumar – Intel
- 276 • Brad Lovering – Microsoft

- 277 • Pat Maynard – Intel
- 278 • Steve Millet – Microsoft
- 279 • Matthew Senft – Microsoft
- 280 • Barry Shilmover – Microsoft
- 281 • Tom Slaight – Intel
- 282 • Marvin Theimer – Microsoft
- 283 • Dave Tobias – AMD
- 284 • John Tollefsrud – Sun
- 285 • Anders Vinberg – Microsoft
- 286 • Megan Wallent – Microsoft

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013

287
288

Web Services for Management (WS-Management) Specification

1 Scope

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

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

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

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

This specification intends to meet the following requirements:

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

2 Normative References

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

IETF RFC 2616, R. Fielding et al, *Hypertext Transfer Protocol (HTTP 1.1)*, June 1999, <http://www.ietf.org/rfc/rfc2616.txt>

IETF RFC 2818, E. Rescorla, *HTTP over TLS (HTTPS)*, May 2000, <http://www.ietf.org/rfc/rfc2818.txt>

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

- 323 IETF, RFC 4122, P. Leach et al, *A Universally Unique Identifier (UUID) URN Namespace*, July 2005,
 324 <http://www.ietf.org/rfc/rfc4122.txt>
- 325 IETF RFC 4178, L. Zhu et al, *The Simple and Protected Generic Security Service Application*
 326 *Program Interface (GSS-API) Negotiation Mechanism*, October 2005,
 327 <http://www.ietf.org/rfc/rfc4178.txt>
- 328 IETF, RFC 4559, K. Jaganathan et al, *SPNEGO-based Kerberos and NTLM HTTP Authentication in*
 329 *Microsoft Windows*, June 2006, <http://www.ietf.org/rfc/rfc4559.txt>
- 330 IETF RFC 5646, A. Phillips et al, *Tags for Identifying Languages*, September 2009,
 331 <http://tools.ietf.org/rfc/rfc5646.txt>
- 332 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
 333 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>
- 334 OASIS, A. Nadalin et al, *Web Services Security Username Token Profile 1.0*, March 2004,
 335 <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>
- 336 OASIS, A. Nadalin et al, *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*,
 337 March 2004, [http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
 338 [1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
- 339 OASIS, S. Anderson et al, *Web Services Trust Language (WS-Trust)*, December 2005,
 340 <http://schemas.xmlsoap.org/ws/2005/02/trust>
- 341 The Unicode Consortium, *The Unicode Standard Version 3.0*, January 2000,
 342 <http://www.unicode.org/book/u2.html>
- 343 The Unicode Consortium, *Byte Order Mark (BOM) FAQ*,
 344 http://www.unicode.org/faq/utf_bom.html#BOM
- 345 W3C, M. Gudgin, et al, *SOAP Version 1.2 Part 1: Messaging Framework*, June 2003,
 346 <http://www.w3.org/TR/soap12-part1/>
- 347 W3C, M. Gudgin, et al, *SOAP Version 1.2 Part 2: Adjuncts*, June 2003,
 348 <http://www.w3.org/TR/2003/REC-soap12-part2-20030624>
- 349 W3C, M. Gudgin, et al, *SOAP Message Transmission Optimization Mechanism (MTOM)*,
 350 November 2004, <http://www.w3.org/TR/2004/PR-soap12-mtom-20041116/>
- 351 W3C, J. Clark et al, *XML Path Language Version 1.0 (XPath 1.0)*, November 1999,
 352 <http://www.w3.org/TR/1999/REC-xpath-19991116>
- 353 W3C, J. Cowan et al, *XML Information Set Second Edition (XML Infoset)*, February 2004,
 354 <http://www.w3.org/TR/2004/REC-xml-20040204/>
- 355 W3C, H. Thompson et al, *XML Schema Part 1: Structures (XML Schema 1)*, May 2001,
 356 <http://www.w3.org/TR/xmlschema-1/>
- 357 W3C, P. Biron et al, *XML Schema Part 2: Datatypes (XML Schema 2)*, May 2001,
 358 <http://www.w3.org/TR/xmlschema-2/>
- 359 W3C, *Web Services Addressing 1.0 – Core*, W3C Recommendation, May 2006,
 360 <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/>
- 361 W3C, *Web Services Addressing 1.0 – SOAP Binding*, W3C Recommendation, May 2006,
 362 <http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/>
- 363 W3C, *Web Services Addressing 1.0 – Metadata*, W3C Recommendation, September 2007,
 364 <http://www.w3.org/TR/2007/REC-ws-addr-metadata-20070904/>
- 365 W3C, *Extensible Markup Language (XML) 1.0*, W3C Recommendation, October 2000,
 366 <http://www.w3.org/TR/2000/REC-xml-20001006>

- 367 W3C, *Namespaces in XML*, W3C Recommendation, January 1999,
368 <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- 369 W3C, E. Christensen et al, *Web Services Description Language Version 1.1 (WSDL/1.1)*, March
370 2001, <http://www.w3.org/TR/wsdl>
- 371 W3C, S. Boag et al, *XQuery 1.0: An XML Query Language (XQuery 1.0)*, January 2007,
372 <http://www.w3.org/TR/2007/REC-xquery-20070123/>

373 **3 Terms and Definitions**

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

377 **3.1**

378 **can**

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

380 **3.2**

381 **cannot**

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

383 **3.3**

384 **conditional**

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

387 **3.4**

388 **mandatory**

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

391 **3.5**

392 **may**

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

394 **3.6**

395 **need not**

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

397 **3.7**

398 **optional**

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

400 **3.8**

401 **shall**

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

404 **3.9**

405 **shall not**

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

- 408 **3.10**
409 **should**
410 indicates that among several possibilities, one is recommended as particularly suitable, without
411 mentioning or excluding others, or that a certain course of action is preferred but not necessarily
412 required
- 413 **3.11**
414 **should not**
415 indicates that a certain possibility or course of action is deprecated but not prohibited
- 416 **3.12**
417 **client**
418 the application that uses the Web services defined in this document to access the management
419 service
- 420 **3.13**
421 **consumer**
422 the Web service that is requesting the data enumeration from the data source
- 423 **3.14**
424 **data source**
425 a Web service that supports traversal using enumeration contexts via the Enumerate operation
426 defined in this specification
- 427 **3.15**
428 **delivery mode**
429 the mechanism by which notification messages are delivered from the source to the sink
- 430 **3.16**
431 **enumeration context**
432 a session context that represents a specific traversal through a logical sequence of XML element
433 information items using the Pull operation defined in this specification
- 434 **3.17**
435 **event sink**
436 a Web service that receives notifications
- 437 **3.18**
438 **event source**
439 a Web service that sends notifications and accepts requests to create subscriptions
- 440 **3.19**
441 **managed resource**
442 an entity that can be of interest to an administrator
443 It may be a physical object, such as a laptop computer or a printer, or an abstract entity, such as a
444 service.
- 445 **3.20**
446 **notification**
447 a message sent to indicate that an event has occurred

448 **3.21**449 **push mode**

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

452 **3.22**453 **resource**

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

457 **3.23**458 **resource class**

459 an abstract representation (type) of a managed resource

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

463 **3.24**464 **resource factory**

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

467 **3.25**468 **resource instance**

469 an instantiation of a resource class

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

472 **3.26**473 **selector**

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

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

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

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

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

484 **3.27**485 **service**

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

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

490 **3.28**491 **subscriber**

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

493 **3.29**
494 **subscription manager**
495 a Web service that accepts requests to manage, get the status of, renew, and/or delete subscriptions
496 on behalf of an event source

497 **4 Symbols and Abbreviated Terms**

498 The following symbols and abbreviations are used in this document.

499 **4.1**

500 **BNF**

501 Backus-Naur Form (<http://foldoc.org/foldoc/?Backus-Naur+Form>)

502 **4.2**

503 **BOM**

504 byte-order mark

505 **4.3**

506 **CQL**

507 CIM Query Language

508 **4.4**

509 **EPR**

510 Endpoint Reference

511 **4.5**

512 **GSSAPI**

513 Generic Security Services Application Program Interface

514 **4.6**

515 **SOAP**

516 Simple Object Access Protocol

517 **4.7**

518 **SPNEGO**

519 Simple and Protected GSSAPI Negotiation Mechanism

520 **4.8**

521 **SQL**

522 Structured Query Language

523 **4.9**

524 **URI**

525 Uniform Resource Identifier

526 **4.10**

527 **URL**

528 Uniform Resource Locator

529 **4.11**

530 **UTF**

531 UCS Transformation Format

532 **4.12**

533 **UUID**

534 Universally Unique Identifier

535 **4.13**

536 **WSDL**

537 Web Services Description Language

538 **4.14**

539 **WS-Man**

540 Web Services Management

541 **5 Addressing**

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

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

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

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

557 **5.1 Management Addressing**

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

563 **5.1.1 Introduction**

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

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

- 571 • Endpoint references are suitable for conveying the information needed to access a Web
572 service endpoint.

- 573 • Endpoint references are also used to provide addresses for individual messages sent to
574 and from Web services.

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

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

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

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

```
588 (001) <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
589         xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
590 (002)   <S:Header>
591 (003)     <wsa:MessageID>
592 (004)       uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
593 (005)     </wsa:MessageID>
594 (006)     <wsa:From>
595 (007)       <wsa:Address>http://business456.example/client1</wsa:Address>
596 (008)     </wsa:From>
597 (009)     <wsa:ReplyTo>
598 (010)       <wsa:Address>http://business456.example/client1</wsa:Address>
599 (011)     </wsa:ReplyTo>
600 (012)     <wsa:To>http://fabrikam123.example/Purchasing</wsa:To>
601 (013)     <wsa:Action>http://fabrikam123.example/SubmitPO</wsa:Action>
602 (014)   </S:Header>
603 (015)   <S:Body>
604 (016)     ...
605 (017)   </S:Body>
606 (018) </S:Envelope>
```

607 5.1.2 Endpoint References

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

609 5.1.2.1 Format of Endpoint References

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

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

```
614 <wsa:EndpointReference>
615   <wsa:Address>xs:anyURI</wsa:Address>
616   <wsa:ReferenceProperties>... </wsa:ReferenceProperties> ?
617   <wsa:ReferenceParameters>... </wsa:ReferenceParameters> ?
618   <wsa:PortType>xs:QName</wsa:PortType> ?
619   <wsa:ServiceName PortName="xs:NCName"?>xs:QName</wsa:ServiceName> ?
620   <wsp:Policy> ... </wsp:Policy>*
621 </wsa:EndpointReference>
```

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

623 `wsa:EndpointReference`

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

627 `wsa:EndpointReference/wsa:Address`

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

630 `wsa:EndpointReference/wsa:ReferenceProperties/`

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

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

636 `wsa:EndpointReference/wsa:ReferenceProperties/{any}`

637 Each child element of `ReferenceProperties` represents an individual reference property.

638 `wsa:EndpointReference/wsa:ReferenceParameters/`

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

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

645 `wsa:EndpointReference/wsa:ReferenceParameters/{any}`

646 Each child element of `ReferenceParameters` represents an individual reference parameter.

647 `wsa:EndpointReference/wsa:PortType`

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

650 NOTE: The use of `wsa:PortType` is deprecated.

651 `wsa:EndpointReference/wsa:ServiceName`

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

656 NOTE: The use of `wsa:ServiceName` is deprecated.

657 `wsa:EndpointReference/wsa:ServiceName/@PortName`

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

660 wsa:EndpointReference/wsp:Policy

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

662 NOTE: The use of wsp:Policy is deprecated.

663 wsa:EndpointReference/{any}

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

665 wsa:EndpointReference/@{any}

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

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

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

672 5.1.2.2 Binding Endpoint References

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

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

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

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

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

```
691 <wsa:EndpointReference xmlns:wsa="..." xmlns:fabrikam="...">
692   <wsa:Address>http://www.fabrikam123.example/acct</wsa:Address>
693   <wsa:ReferenceParameters>
694     <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
695     <fabrikam:ShoppingCart>ABCDEFGFG</fabrikam:ShoppingCart>
696   </wsa:ReferenceParameters>
697 </wsa:EndpointReference>
```

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

```
701 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
702   xmlns:wsa="..." xmlns:fabrikam="...">
703   <S:Header>
```

```

704     ...
705     <wsa:To>http://www.fabrikam123.example/acct</wsa:To>
706     <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
707     <fabrikam:ShoppingCart>ABCDEFG</fabrikam:ShoppingCart>
708     ...
709     </S:Header>
710     <S:Body>
711     ...
712     </S:Body>
713 </S:Envelope>

```

714 5.1.3 Message Information Headers

715 This clause defines the syntax of a message information header.

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

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

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

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

```

727 <wsa:MessageID> xs:anyURI </wsa:MessageID>
728 <wsa:RelatesTo RelationshipType="..."?>xs:anyURI</wsa:RelatesTo>
729 <wsa:To>xs:anyURI</wsa:To>
730 <wsa:Action>xs:anyURI</wsa:Action>
731 <wsa:From>endpoint-reference</wsa:From>
732 <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
733 <wsa:FaultTo>endpoint-reference</wsa:FaultTo>

```

734 **Figure 1 – Message Information Header Blocks**

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

736 **wsa:MessageID**

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

743 wsa:RelatesTo

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

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

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

751 wsa:RelatesTo/@RelationshipType

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

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

755 **Table 1 – Relationship Type**

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

756 wsa:ReplyTo

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

764 wsa:From

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

767 wsa:FaultTo

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

775 wsa:To

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

778 wsa:Action

779 This required element (of type xs:anyURI) uniquely identifies the semantics implied by this
 780 message. It is recommended that the value of this header be a URI identifying an input, output, or
 781 fault message within a WSDL port type. An action may be explicitly or implicitly associated with

782 the corresponding WSDL definition. Finally, if in addition to the wsa:Action header, a SOAP Action
 783 URI is encoded in a request, the URI of the SOAP Action shall either be the same as the one
 784 specified by the wsa:Action header, or set to "".

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

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

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

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

798 5.1.3.1 Formulating a Reply Message

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

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

```
803 <S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
804   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
805   xmlns:f123="http://www.fabrikam123.example/svc53">
806   <S:Header>
807     <wsa:MessageID>uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff
808     </wsa:MessageID>
809     <wsa:ReplyTo>
810       <wsa:Address>http://business456.example/client1</wsa:Address>
811     </wsa:ReplyTo>
812     <wsa:To S:mustUnderstand="1">mailto:joe@fabrikam123.example</wsa:To>
813     <wsa:Action>http://fabrikam123.example/mail/Delete</wsa:Action>
814   </S:Header>
815   <S:Body>
816     <f123:Delete>
817       <maxCount>42</maxCount>
818     </f123:Delete>
819   </S:Body>
820 </S:Envelope>
```

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

```
823 <S:Envelope
824   xmlns:S="http://www.w3.org/2003/05/soap-envelope"
825   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
826   xmlns:f123="http://www.fabrikam123.example/svc53">
827   <S:Header>
828     <wsa:MessageID>
829       uuid:aaaabbbb-cccc-dddd-eeee-wwwwwwwwwww
830     </wsa:MessageID>
831     <wsa:RelatesTo>
```

```

832     uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff
833   </wsa:RelatesTo>
834   <wsa:To>
835     http://business456.example/client1
836   </wsa:To>
837   <wsa:Action>http://fabrikam123.example/mail/DeleteAck</wsa:Action>
838 </S:Header>
839 <S:Body>
840   <f123>DeleteAck/>
841 </S:Body>
842 </S:Envelope>

```

843 5.1.3.2 Associating Action with WSDL Operations

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

846 5.1.3.2.1 Explicit Association

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

848 EXAMPLE: Consider the following WSDL excerpt:

```

849 <definitions targetNamespace="http://example.com/stockquote" ...>
850   ...
851   <portType name="StockQuotePortType">
852     <operation name="GetLastTradePrice">
853       <input message="tns:GetTradePricesInput"
854         wsa:Action="http://example.com/GetQuote"/>
855       <output message="tns:GetTradePricesOutput"
856         wsa:Action="http://example.com/Quote"/>
857     </operation>
858   </portType>
859   ...
860 </definitions>

```

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

863 5.1.3.2.2 Default Action Pattern

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

```
866 targetNamespace/portTypeName/(inputName|outputName)
```

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

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

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

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

876 EXAMPLE: Consider the following WSDL excerpt:

```
877 <definitions targetNamespace="http://example.com/stockquote" ...>
878   ...
879   <portType name="StockQuotePortType">
880     <operation name="GetLastTradePrice">
881       <input message="tns:GetTradePricesInput" name="GetQuote"/>
882       <output message="tns:GetTradePricesOutput" name="Quote"/>
883     </operation>
884   </portType>
885   ...
886 </definitions>
```

887 *targetNamespace* = http://example.com/stockquote

888 *portTypeName* = StockQuotePortType

889 *inputName* = GetQuote

890 *outputName* = Quote

891 Applying the preceding pattern with these values produces the following:

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

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

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

896 EXAMPLE: The following is a WSDL excerpt:

```
897 <definitions targetNamespace="http://example.com/stockquote" ...>
898   ...
899   <portType name="StockQuotePortType">
900     <operation name="GetLastTradePrice">
901       <input message="tns:GetTradePricesInput"/>
902       <output message="tns:GetTradePricesOutput"/>
903     </operation>
904   </portType>
905   ...
906 </definitions>
```

907 *targetNamespace* = http://example.com/stockquote

908 *portTypeName* = StockQuotePortType

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

911 *inputName* = GetLastTradePriceRequest

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

913 *outputName* = GetLastTradePriceResponse

914 Applying the previous pattern with these values produces the following:

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

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

5.2 Versions of Addressing

To maintain compatibility with implementations of previous versions of WS-Management, this protocol accommodates messages formatted by those previous versions. However, WS-Management 1.1 also allows for the optional use of the [WS-Addressing W3C Recommendation](#).

The following abbreviations are used for clarity and brevity.

- "WSMA" refers to the version of Management Addressing as specified in 5.1.
- "WSA-Rec" refers to the WS-Addressing W3C Recommendation.
- "WS-Man 1.0" refers to the *WS-Management Specification* 1.0 and implementations compatible with that specification.
- "WS-Man 1.1" refers to this specification and implementations compatible with this specification.
- "Addressing Anonymous URI" refers to the anonymous URI that is defined by the version of Addressing currently in use. The anonymous URI defined by WSA-Rec is <http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous>. The anonymous URI defined by WSMA is <http://www.w3.org/2005/08/addressing/anonymous>.

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

5.2.1 Technical Differences

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

5.3 Requirements for Compatibility

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

Table 2 – Interoperability Requirements

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

Homogeneous pairings of compliant clients and services (that is, a version 1.0 client with a version 1.0 service, or a version 1.1 client with a version 1.1 service) can exchange messages in accordance with their respective specifications. To ensure reliable communications, heterogeneous pairings need to meet certain requirements and implement certain sequencing strategies.

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

953 **Table 3 – WSA Versions in Exchanges**

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

954 **5.3.1 Discovery or Negotiation**

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

960 Identify would be used as follows:

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

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

972 **5.3.2 Client Negotiation Strategies**

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

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

984 **5.3.3 Initiating Message Exchanges**

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

5.3.4 Normative Rules

R5.3.4-1: If a WS-Man 1.1 service supports WSA-Rec, then it shall also support the Identify operation.

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

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

R5.3.4-4: Within a single SOAP message, a WS-Man 1.1 implementation shall use the same version of Addressing for all Addressing SOAP headers.

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

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

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

5.4 Use of Addressing in WS-Management

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

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

5.4.1 Use of Endpoint References

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

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

1028 Rule R5.4.1-1 clarifies that messages such as Pull or Renew still require a full EPR. For Pull, for
 1029 example, this EPR would be the same as the original Enumerate, even though EnumerateResponse
 1030 returns a context object that would seem to obviate the need for the EPR. The EPR is still required to
 1031 route the message properly. Similarly, the Renew request uses the SubscriptionManager EPR
 1032 received in the SubscribeResponse.

1033 When a service includes an EPR in a response message, it must be willing to accept subsequent
 1034 request messages targeted to that EPR for the same individual managed resource. Clients are not
 1035 required to process or enhance EPRs given to them by the service before using them to address a
 1036 managed resource.

1037 **R5.4.1-2:** An EPR returned by a service shall be acceptable to that service to refer to the
 1038 same managed resource.

1039 **R5.4.1-3:** All EPRs returned by a service, whether expressed using the WS-Management
 1040 default addressing model (see 5.4.2) or any other addressing model, shall be valid as long as the
 1041 managed resource exists.

1042 5.4.2 WS-Management Default Addressing Model

1043 WS-Management defines a default addressing model for resources. A service is not required to use
 1044 this addressing model, but it is suitable for many new implementations and can increase the chances
 1045 of successful interoperation between clients and services.

1046 This document uses examples of this addressing model that contain its component parts, the
 1047 ResourceURI and SelectorSet SOAP headers. This specification is independent of the actual data
 1048 model and does not define the structure of the ResourceURI or the set of values for selectors for a
 1049 given resource. These may be vendor specific or defined by other specifications.

1050 Description and use of this addressing model in this specification do not indicate that support for this
 1051 addressing model is a requirement for a conformant service.

1052 All of the normative text, examples, and conformance rules in 5.4.2 and 5.4.2.2 presume that the
 1053 service is based on the default addressing model. In cases where this addressing model is not in use,
 1054 these rules do not apply.

1055 The default addressing model uses a representation of an EPR that is a tuple of the following SOAP
 1056 headers:

- 1057 • **wsa:To** (required): the transport address of the service
- 1058 • **wsman:ResourceURI** (required if the default addressing model is used): the URI of the
 1059 resource class representation or instance representation
- 1060 • **wsman:SelectorSet** (optional): a header that identifies or "selects" the resource instance to
 1061 be accessed if more than one instance of a resource class exists

1062 The wsman:ResourceURI value needs to be marked with an s:mustUnderstand attribute set to "true"
 1063 in all messages that use the default addressing model. Otherwise, a service that does not understand
 1064 this addressing model might inadvertently return a resource that was not requested by the client.

1065 The WS-Management default addressing model is defined in the following XML outline for an EPR:

```

1066 (1) <wsa:EndpointReference>
1067 (2)   <wsa:Address>
1068 (3)     Network address
1069 (4)   </wsa:Address>
1070 (5)   <wsa:ReferenceParameters>
1071 (6)     <wsman:ResourceURI> resource URI </wsman:ResourceURI>
1072 (7)     <wsman:SelectorSet>
```

```

1073 (8)      <wsman:Selector Name="selector-name"> *
1074 (9)      Selector-value
1075 (10)     </wsman:Selector>
1076 (11)    </wsman:SelectorSet> ?
1077 (12)    </wsa:ReferenceParameters>
1078 (13)    </wsa:EndpointReference>

```

1079 The following definitions provide additional, normative constraints on the preceding outline:

1080 **wsa:Address**

1081 the URI of the transport address

1082 **wsa:ReferenceParameters/wsman:ResourceURI**

1083 the URI of the resource class or instance to be accessed

1084 Typically, this URI represents the resource class, but it may represent the instance. The
 1085 combination of this URI and the wsa:To URI form the full address of the resource class or
 1086 instance.

1087 **wsa:ReferenceParameters/wsman:SelectorSet:**

1088 the optional set of selectors as described in 5.4.2.2

1089 These values are used to select an instance if the ResourceURI identifies a multi-instanced
 1090 target.

1091 When the default addressing model is used in a SOAP message, Addressing specifies that
 1092 translations take place and the headers are flattened out.

1093 **EXAMPLE:** The following is an example EPR definition:

```

1094 (1) <wsa:EndpointReference>
1095 (2) <wsa:Address> Address </wsa:Address>
1096 (3) <wsa:ReferenceParameters xmlns:wsman="...">
1097 (4) <wsman:ResourceURI>resURI</wsman:ResourceURI>
1098 (5) <wsman:SelectorSet>
1099 (6) <wsman:Selector Name="Selector-name">
1100 (7) Selector-value
1101 (8) </wsman:Selector>
1102 (9) </wsman:SelectorSet>
1103 (10) </wsa:ReferenceParameters>
1104 (11) </wsa:EndpointReference>

```

1105 This address definition is translated as follows when used in a SOAP message. wsa:Address becomes wsa:To,
 1106 and the reference parameters are unwrapped and juxtaposed. The following example shows a sample SOAP
 1107 message using WSMA:

```

1108 (1) <s:Envelope xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
1109 (2) <s:Header>
1110 (3) <wsa:To> Address </wsa:To>
1111 (4) <wsa:Action> Action URI </wsa:Action>
1112 (5) <wsman:ResourceURI s:mustUnderstand="true">resURI</wsman:ResourceURI>
1113 (6) <wsman:SelectorSet>
1114 (7) <wsman:Selector Name="Selector-name">
1115 (8) Selector-value
1116 (9) </wsman:Selector>
1117 (10) </wsman:SelectorSet>
1118 (11) ...
1119 (12) </s:Header>
1120 (13) <s:Body> ... </s:Body>
1121 (14) </s:Envelope>

```

1122 The following message shows a sample SOAP message using WS-Rec:

```

1123 (1) <s:Envelope xmlns:wsa="http://www.w3.org/2005/08/addressing ">
1124 (2)   <s:Header>
1125 (3)     <wsa:To s:mustUnderstand="true"> Address </wsa:To>
1126 (4)     <wsa:Action s:mustUnderstand="true"> Action URI </wsa:Action>
1127 (5)     <wsman:ResourceURI s:mustUnderstand="true"
1128 (6)       wsa:isReferenceParameter="true">resURI</wsman:ResourceURI>
1129 (7)     <wsman:SelectorSet wsa:isReferenceParameter="true">
1130 (8)       <wsman:Selector Name="Selector-name">
1131 (9)         Selector-value
1132 (10)       </wsman:Selector>
1133 (11)     </wsman:SelectorSet>
1134 (12)   ...
1135 (13) </s:Header>
1136 (14) <s:Body> ... </s:Body>
1137 (15) </s:Envelope>

```

1138 In both cases, the wsa:To, wsman:ResourceURI, and wsman:SelectorSet elements work together to
 1139 *reference* the resource instance to be managed, but the actual *method* or *operation* to be executed
 1140 against this resource is indicated by the wsa:Action header.

1141 **EXAMPLE:** The following is an example of Addressing headers based on the default addressing model in an
 1142 actual message:

```

1143 (1) <s:Envelope
1144 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
1145 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
1146 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
1147 (5)   <s:Header>
1148 (6)     ...
1149 (7)     <wsa:To>http://123.99.222.36/wsman</wsa:To>
1150 (8)     <wsman:ResourceURI s:mustUnderstand="true">
1151 (9)       http://example.org/hardware/2005/02/storage/physDisk
1152 (10)    </wsman:ResourceURI>
1153 (11)    <wsman:SelectorSet>
1154 (12)      <wsman:Selector Name="LUN"> 2 </wsman:Selector>
1155 (13)    </wsman:SelectorSet>
1156 (14)    <wsa:Action> http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
1157 (15)    </wsa:Action>
1158 (16)    <wsa:MessageID> urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91
1159 (17)    </wsa:MessageID>
1160 (18)  ...
1161 (19) </s:Header>
1162 (20) <s:Body> ... </s:Body>
1163 (21) </s:Envelope>

```

1164 The following definitions apply to the preceding message example:

- 1165 wsa:To
- 1166 the network (or transport-level) address of the service
- 1167 wsman:ResourceURI
- 1168 the ResourceURI of the resource class or resource instance to be accessed
- 1169 wsman:SelectorSet
- 1170 a wrapper for the selectors

1171 wsman:SelectorSet/wsman:Selector
 1172 identifies or selects the resource instance to be accessed, if more than one instance of the
 1173 resource exists
 1174 In this case, the selector is "LUN" (logical unit number), and the selected device is unit number
 1175 "2".

1176 wsa:Action
 1177 identifies which operation is to be carried out against the resource (in this case, a "Get")

1178 wsa:MessageID
 1179 identifies this specific message uniquely for tracking and correlation purposes
 1180 The format defined in [RFC 4122](#) is often used in the examples in this specification, but it is not
 1181 required.

1182 5.4.2.1 ResourceURI

1183 The ResourceURI is used to indicate the class resource or instance.

1184 **R5.4.2.1-1:** The format of the wsman:ResourceURI is unconstrained provided that it meets
 1185 [RFC 3986](#) requirements.

1186 The format and syntax of the ResourceURI is any valid URI according to [RFC 3986](#). Although there is
 1187 no default scheme, http: and urn: are common defaults. If http: is used, users may expect to find
 1188 Web-based documentation of the resource at that address. The wsa:To and the wsman:ResourceURI
 1189 elements work together to define the actual resource being targeted.

1190 **R5.4.2.1-2:** Vendor-specific or organization-specific URIs should contain the Internet domain
 1191 name in the first token sequence after the scheme, such as "example.org" in ResourceURI in the
 1192 following example.

1193 EXAMPLE:

```

1194 (20) <s:Header>
1195 (21)   <wsa:To> http://128.15.166.67/wsman </wsa:To>
1196 (22)   <wsman:ResourceURI>
1197 (23)     http://schemas.example.org/2005/02/hardware/physDisk
1198 (24)   </wsman:ResourceURI>
1199 (25)   ...
1200 (26) </s:Header>
  
```

1201 **R5.4.2.1-3:** When the default addressing model is used, the wsman:ResourceURI reference
 1202 parameter is required in messages with the following wsa:Action URIs:

1203 http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
 1204 http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
 1205 http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
 1206 http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
 1207 http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
 1208 http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
 1209 http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew
 1210 http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus
 1211 http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release
 1212 http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe

- 1213 The following messages require the EPR to be returned in the SubscriptionManager element of the
1214 SubscribeResponse message. The format of the EPR is determined by the service and might or
1215 might not include the ResourceURI:
- 1216 `http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew`
1217 `http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus`
- 1218 While the ResourceURI SOAP header is required when the WS-Management default addressing
1219 mode is used, it may be short and of a very simple form, such as `http://example.com/*` or
1220 `http://example.com/resource`.
- 1221 **R5.4.2.1-4:** For the request message of custom actions (methods), the ResourceURI header may
1222 be present in the message to help route the message to the correct handler.
- 1223 **R5.4.2.1-5:** The ResourceURI element should not appear in other messages, such as responses
1224 or events, unless the associated EPR includes it in its ReferenceParameters.
- 1225 In practice, the wsman:ResourceURI element is required only in requests to reference the targeted
1226 resource class. Responses are not addressed to a management resource, so the
1227 wsman:ResourceURI has no meaning in that context.
- 1228 **R5.4.2.1-6:** When the default addressing model is used and the wsman:ResourceURI element is
1229 missing or in an incorrect form, the service shall issue a wsa:DestinationUnreachable fault with a
1230 detail code of
- 1231 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI`
- 1232 **R5.4.2.1-7:** The wsman:ResourceURI element shall be used to indicate only the identity of a
1233 resource, and it may not be used to indicate the action being applied to that resource, which is
1234 properly expressed using the wsa:Action URI.
- 1235 Custom WSDL-based methods have both a ResourceURI identity from the perspective of addressing
1236 and a wsa:Action URI from the perspective of execution. In many cases, the ResourceURI is simply a
1237 pseudonym for the WSDL identity and Port, and the wsa:Action URI is the specific method within that
1238 port (or interface) definition.
- 1239 Although a single URI could theoretically be used alone to define an instance of a multi-instance
1240 resource, it is recommended that the wsa:To element be used to locate the WS-Management service,
1241 that the wsman:ResourceURI element be used to identify the resource class, and that the
1242 wsman:SelectorSet element be used to reference the resource instance. If the resource consists of
1243 only a single instance, then the wsman:ResourceURI element alone refers to the single instance.
- 1244 This usage is not a strict requirement, just a guideline. The service can use distinct selectors for any
1245 given operation, even against the same resource class, and may allow or require selectors for the
1246 Enumerate operation.
- 1247 See the recommendations in 7.2 regarding addressing uniformity.
- 1248 Custom actions have two distinct identities: the ResourceURI, which can identify the WSDL and port
1249 (or interface), and the wsa:Action URI, which identifies the specific method. If only one method exists
1250 in the interface, in a sense the ResourceURI and wsa:Action URI are identical.
- 1251 It is not an error to use the wsa:Action URI for the ResourceURI of a custom method, but both are still
1252 required in the message for uniform processing on both clients and servers.

1253 EXAMPLE 1: The following action to reset a network card might have the following EPR usage:

```

1254 (1) <s:Header>
1255 (2)   <wsa:To>
1256 (3)     http://1.2.3.4/wsman/
1257 (4)   </wsa:To>
1258 (5)   <wsman:ResourceURI>http://example.org/2005/02/networkcards/reset
1259 (6)   </wsman:ResourceURI>
1260 (7)   <wsa:Action>
1261 (8)     http://example.org/2005/02/networkcards/reset
1262 (9)   </wsa:Action>
1263 (10)  ...
1264 (11) </s:Header>

```

1265 In many cases, the ResourceURI is equivalent to a WSDL name and port, and the wsa:Action URI
 1266 contains an additional token as a suffix, as in the following example.

1267 EXAMPLE 2:

```

1268 (1) <s:Header>
1269 (2)   <wsa:To>
1270 (3)     http://1.2.3.4/wsman
1271 (4)   </wsa:To>
1272 (5)   <wsman:ResourceURI>http://example.org/2005/02/networkcards
1273 (6)   </wsman:ResourceURI>
1274 (7)   <wsa:Action>
1275 (8)     http://example.org/2005/02/networkcards/reset
1276 (9)   </wsa:Action>
1277 (10)  ...
1278 (11) </s:Header>

```

1279 Finally, the ResourceURI may be completely unrelated to the wsa:Action URI, as in the following
 1280 example.

1281 EXAMPLE 3:

```

1282 (1) <s:Header>
1283 (2)   <wsa:To>http://1.2.3.4/wsman</wsa:To>
1284 (3)   <wsman:ResourceURI>
1285 (4)     http://example.org/products/management/networkcards
1286 (5)   </wsman:ResourceURI>
1287 (6)   <wsa:Action>
1288 (7)     http://example.org/2005/02/netcards/reset
1289 (8)   </wsa:Action>
1290 (9)   ...
1291 (10) </s:Header>

```

1292 All of these uses are legal.

1293 When used with subscriptions, the EPR described by wsa:Address and wsman:ResourceURI (and
 1294 optionally the wsman:SelectorSet values) identifies the event source to which the subscription is
 1295 directed. In many cases, the ResourceURI identifies a real or virtual event log, and the subscription is
 1296 intended to provide real-time notifications of any new entries added to the log. In many cases, the
 1297 wsman:SelectorSet element might not be used as part of the EPR.

1298 5.4.2.2 Selectors

1299 In the WS-Management default addressing model, selectors are optional elements used to identify
 1300 instances within a resource class. For operations such as Get or Put, the selectors are used to
 1301 identify a single instance of the resource class referenced by the ResourceURI.

1302 In practice, because the ResourceURI often acts as a table or a "class," the SelectorSet element is a
 1303 discriminant used to identify a specific "row" or "instance." If only one instance of a resource class is
 1304 implied by the ResourceURI, the SelectorSet can be omitted because the ResourceURI is acting as
 1305 the full identity of the resource. If more than one selector value is required, the entire set of selectors
 1306 is interpreted by the service in order to reference the specific instance. The selectors are interpreted
 1307 as being separated by implied logical AND operators.

1308 In some information domains, the values referenced by the selectors are "keys" that are part of the
 1309 resource content itself, whereas in other domains the selectors are part of a logical or physical
 1310 directory system or search space. In these cases, the selectors are used to identify the resource, but
 1311 are not part of the representation.

1312 **R5.4.2.2-1:** If a resource has more than one instance, a wsman:SelectorSet element may be
 1313 used to distinguish which instance is targeted if the WS-Management default addressing model is
 1314 in use. Any number of wsman:Selector values may appear with the wsman:SelectorSet element,
 1315 as required to identify the precise instance of the resource class. The service may consider the
 1316 case of selector names and values (see 13.6), as required by the underlying execution
 1317 environment.

1318 If the client needs to discover the policy on how the case of selector values is interpreted, the service
 1319 can provide metadata documents that describe this policy. The format of such metadata is beyond
 1320 the scope of this specification.

1321 **R5.4.2.2-2:** All content within the SelectorSet element is to be treated as a single reference
 1322 parameter with a scope relative to the ResourceURI.

1323 **R5.4.2.2-3:** A service using the WS-Management default addressing model shall examine all
 1324 selectors in the message and process them as if they were logically joined by AND. If the set of
 1325 selectors is incorrect for the targeted resource instance, a wsman:InvalidSelectors fault should be
 1326 returned to the client with the following detail codes:

- 1327 • if selectors are missing:
 1328 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelectors>
- 1329 • if selector values are the wrong types:
 1330 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch>
- 1331 • if the selector value is of the correct type from the standpoint of XML types, but out of range
 1332 or otherwise illegal in the specific information domain:
 1333 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue>
- 1334 • if the name is not a recognized selector name
 1335 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelectors>

1336 **R5.4.2.2-4:** The Selector Name attribute shall not be duplicated at the same level of nesting. If
 1337 this occurs, the service should return a wsman:InvalidSelectors fault with the following detail
 1338 code:

1339 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DuplicateSelectors>

1340 This specification does not mandate the use of selectors. Some implementations may decide to use
 1341 complex URI schemes in which the ResourceURI itself implicitly identifies the instance.

1342 The format of the SelectorSet element is as follows:

```

1343 (1) <s:Envelope
1344 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
1345 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
1346 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
1347 (5)   <s:Header>
1348 (6)     ...
1349 (7)     <wsa:To> service transport address </wsa:To>
1350 (8)     <wsman:ResourceURI> ResourceURI </wsman:ResourceURI>
1351 (9)     <wsman:SelectorSet>
1352 (10)      <wsman:Selector Name="name"> value </wsman:Selector> +
1353 (11)      </wsman:SelectorSet> ?
1354 (12)     ...
1355 (13)   </s:Header>
1356 (14)   <s:Body> ... </s:Body>
1357 (15) </s:Envelope>

```

1358 The following definitions provide additional, normative constraints on the preceding outline:

1359 wsman:SelectorSet

1360 the wrapper for one or more Selector elements required to reference the instance

1361 wsman:SelectorSet/wsman:Selector

1362 used to describe the selector and its value

1363 If more than one selector is required, one Selector element exists for each part of the overall
 1364 selector. The value of this element is the Selector value.

1365 wsman:SelectorSet/wsman:Selector/@Name

1366 the name of the selector (to be treated in a case-insensitive manner)

1367 The value of a selector may be a nested EPR.

1368 EXAMPLE: In the following example, the selector on line 9 is a part of a SelectorSet that contains a nested
 1369 EPR (lines 10–18) with its own Address, ResourceURI, and SelectorSet elements:

```

1370 (1) <s:Envelope
1371 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
1372 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
1373 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
1374 (5)   <s:Header>
1375 (6)     ...
1376 (7)     <wsman:SelectorSet>
1377 (8)       <wsman:Selector Name="Primary"> 123 </wsman:Selector>
1378 (9)       <wsman:Selector Name="EPR">
1379 (10)        <wsa:EndpointReference>
1380 (11)          <wsa:Address> address </wsa:Address>
1381 (12)          <wsa:ReferenceParameters>
1382 (13)            <wsman:ResourceURI> resource URI </wsman:ResourceURI>
1383 (14)            <wsman:SelectorSet>
1384 (15)              <wsman:Selector Name="name"> value </wsman:Selector>
1385 (16)            </wsman:SelectorSet>
1386 (17)          </wsa:ReferenceParameters>
1387 (18)        </wsa:EndpointReference>
1388 (19)      </wsman:Selector>
1389 (20)    </wsman:SelectorSet>
1390 (21)    ...
1391 (22)  </s:Header>
1392 (23)  <s:Body> ... </s:Body>
1393 (24) </s:Envelope>

```

1394 **R5.4.2.2-5:** For those services using the WS-Management default addressing model, the value of
 1395 a wsman:Selector shall be one of the following values:

- 1396 • a simple type as defined in the XML schema namespace
 1397 http://www.w3.org/2001/XMLSchema
- 1398 • a nested wsa:EndpointReference using the WS-Management default addressing model

1399 A service may fault selector usage with wsman:InvalidSelectors if the selector is not a simple type or
 1400 an EPR.

1401 **R5.4.2.2-6:** A conformant service may reject any selector or nested selector with a nested EPR
 1402 whose wsa:Address value is not the same as the primary wsa:To value or is not the Addressing
 1403 Anonymous URI.

1404 The primary purpose for this nesting mechanism is to allow resources that can answer questions
 1405 about other resources.

1406 **R5.4.2.2-7:** A service may fail to process a selector name of more than 2048 characters.

1407 **R5.4.2.2-8:** A service may fail to process a selector value of more than 4096 characters,
 1408 including any embedded selectors, and may fail to process a message that contains more than
 1409 8096 characters of content in the root SelectorSet element.

1410 **5.4.2.3 Faults for Default Addressing Model**

1411 When faults related to the information in the addressing model based on the default format are
 1412 generated, they may contain specific fault detail codes. These detail codes are called out separately
 1413 in 14.6 and do not apply when service-specific addressing is used.

1414 **5.4.3 Service-Specific Endpoint References**

1415 Although WS-Management specifies a default addressing model, in some cases this model is not
 1416 available or appropriate.

1417 **R5.4.3-1:** A conformant service may not understand the header values used by the
 1418 WS-Management default addressing model. If this is the case, and if the client marks the
 1419 wsman:ResourceURI with mustUnderstand="true", the service shall return an s:NotUnderstood
 1420 fault.

1421 **R5.4.3-2:** A conformant service may require additional header values to be present that are
 1422 beyond the scope of this specification.

1423 Services can thus use alternative addressing models for referencing resources with
 1424 WS-Management. These addressing models might or might not use ResourceURI or SelectorSet
 1425 elements and still be valid addressing models if they conform to the rules of Addressing.

1426 In addition to a defined alternative addressing model, a service might not explicitly define any
 1427 addressing model at all and instead use an opaque EPR generated at run-time, which is handled
 1428 according to the standard rules of Addressing.

1429 When such addressing models are used, the client application has to understand and interoperate
 1430 with discovery methods for acquiring EPRs that are beyond the scope of this specification.

1431 **5.4.4 mustUnderstand**

1432 This clause describes the use of the mustUnderstand attribute, regardless of whether an
 1433 implementation uses WS-Management Addressing (see 5.1) or the W3C Recommendation type of
 1434 WS-Addressing.

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

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

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

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

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

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

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

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

5.4.5 wsa:To

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

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

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

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

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

1478 **R5.4.5-2:** In the absence of other requirements, it is recommended that the network address
 1479 for resources that do not require authentication be suffixed by the token sequence */wsman-anon*.
 1480 If */wsman-anon* is used, authenticated access shall not be required.

1481 (1) `<wsa:To> http://123.15.166.67/wsman-anon </wsa:To>`

1482 Including the network transport address in the SOAP message may seem redundant because the
 1483 network connection would already be established by the client. However, in cases where the
 1484 message is routed through intermediaries, the network transport address is required so that the
 1485 intermediaries can examine the message and make the connection to the actual endpoint.

1486 The *wsa:To* header may encompass any number of tokens required to locate the service and a group
 1487 of resources within that service.

1488 **R5.4.5-3:** The service should generate a fault when the *wsa:To* address cannot be processed
 1489 due to the following situations::

- 1490 • If the resource is offline, a *wsa:EndpointUnavailable* fault is returned with the following
 1491 detail code:
 1492 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ResourceOffline`
- 1493 • If the resource cannot be located ("not found"), a *wsa:DestinationUnreachable* fault is
 1494 returned.
- 1495 • If the resource is valid, but internal errors occur, a *wsman:InternalError* fault is returned.
- 1496 • If the resource cannot be accessed for security reasons, a *wsman:AccessDenied* fault is
 1497 returned.

1498 **5.4.6 Other Addressing Headers**

1499 This clause describes the use of other Addressing headers, regardless of whether an implementation
 1500 uses WS-Management Addressing (see 5.1) or the W3C Recommendation version of WS-
 1501 Addressing.

1502 WS-Management depends on Addressing to describe the rules for use of other Addressing headers.

1503 **5.4.6.1 Processing Addressing Headers**

1504 The following additional addressing-related header blocks occur in WS-Management messages.

1505 **R5.4.6.1-1:** A conformant service shall recognize and process the following Addressing header
 1506 blocks.

- 1507 • ***wsa:To***
- 1508 • ***wsa:ReplyTo*** (required when a response is expected)
- 1509 • ***wsa:FaultTo*** (optional)
- 1510 • ***wsa:MessageID*** (required)
- 1511 • ***wsa:Action*** (required)
- 1512 • ***wsa:RelatesTo*** (required in responses)

1513 The use of these header blocks is discussed in subsequent clauses.

1514 **5.4.6.2 wsa:ReplyTo**

1515 WS-Management requires the following usage of wsa:ReplyTo in addressing:

1516 **R5.4.6.2-1:** A wsa:ReplyTo header shall be present in all request messages when a reply is
 1517 required. This address shall be either a valid address for a new connection using any transport
 1518 supported by the service or the Addressing Anonymous URI, which indicates that the reply is to
 1519 be delivered over the same connection on which the request arrived. If the wsa:ReplyTo header
 1520 is missing, a wsa:MessageInformationHeaderRequired fault is returned.

1521 Some messages, such as event deliveries, SubscriptionEnd, and so on, do not require a response
 1522 and may omit a wsa:ReplyTo element.

1523 **R5.4.6.2-2:** A conformant service may require that all responses be delivered over the same
 1524 connection on which the request arrives. In this case, the URI discussed in R5.4.6.2-1 shall
 1525 indicate this. Otherwise, the service shall return a wsman:UnsupportedFeature fault with the
 1526 following detail code:

1527 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode`

1528 **R5.4.6.2-3:** When delivering events for which acknowledgement of delivery is required, the
 1529 sender of the event shall include a wsa:ReplyTo element and observe the usage in 10.8 of this
 1530 specification.

1531 **R5.4.6.2-4:** This rule intentionally left blank.

1532 **R5.4.6.2-5:** This rule intentionally left blank.

1533 Addressing allows clients to include client-defined reference parameters in wsa:ReplyTo headers.
 1534 Addressing requires that these reference parameters be extracted from requests and placed in the
 1535 responses by removing the ReferenceParameters wrapper and placing all of the values as top-level
 1536 SOAP headers in the response, as discussed in 5.1. This allows clients to better correlate responses
 1537 with the original requests. This step cannot be omitted.

1538 **EXAMPLE:** In the following example, the header x:someHeader is included in the reply message:

```

1539 (1) <s:Envelope
1540 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
1541 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
1542 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
1543 (5)   <s:Header>
1544 (6)     ...
1545 (7)     <wsa:To> http://1.2.3.4/wsman </wsa:To>
1546 (8)     <wsa:ReplyTo>
1547 (9)       <wsa:Address>
1548 (10)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
1549 (11)      </wsa:Address>
1550 (12)      <wsa:ReferenceParameters>
1551 (13)        <x:someHeader xmlns:x="..."> user-defined content </x:someHeader>
1552 (14)      </wsa:ReferenceParameters>
1553 (15)    </wsa:ReplyTo>
1554 (16)    ...
1555 (17)  </s:Header>
1556 (18)  <s:Body> ... </s:Body>
1557 (19) </s:Envelope>

```

1558 **R5.4.6.2-6:** If the wsa:ReplyTo address is not usable or is missing, the service should not reply to
 1559 the request and it should close or terminate the connection according to the rules of the current
 1560 network transport. In these cases, the service should locally log some type of entry to help locate
 1561 the client defect later.

1562 **5.4.6.3 wsa:FaultTo**

1563 WS-Management qualifies the use of wsa:FaultTo as indicated in this clause.

1564 **R5.4.6.3-1:** A conformant service may support a wsa:FaultTo address that is distinct from the
 1565 wsa:ReplyTo address. If such a request is made and is not supported by the service, a
 1566 wsman:UnsupportedFeature fault shall be returned with the following detail code:

1567 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode`

1568 If both the wsa:FaultTo and wsa:ReplyTo headers are omitted from a request, transport-level
 1569 mechanisms are typically used to fail the request because the address to which the fault is to be sent
 1570 is uncertain. In such a case, it is not an error for the service to simply shut down the connection.

1571 **R5.4.6.3-2:** If wsa:FaultTo is omitted, the service shall return the fault to the wsa:ReplyTo
 1572 address if a fault occurs.

1573 **R5.4.6.3-3:** A conformant service may require that all faults be delivered to the client over the
 1574 same transport or connection on which the request arrives. In this case, the URI shall be the
 1575 Addressing Anonymous URI. If services do not support separately addressed fault delivery and
 1576 the wsa:FaultTo is any other address, a wsman:UnsupportedFeature fault shall be returned with
 1577 the following detail code:

1578 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode`

1579 NOTE: This specification does not restrict richer implementations from fully supporting wsa:FaultTo.

1580 **R5.4.6.3-4:** This rule intentionally left blank.

1581 EXAMPLE: In the following example, the header x:someHeader is included in fault messages if they occur:

```

1582 (1) <s:Envelope
1583 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
1584 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
1585 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
1586 (5)   <s:Header>
1587 (6)     ...
1588 (7)     <wsa:To> http://12.3.4/wsman </wsa:To>
1589 (8)     <wsa:FaultTo>
1590 (9)       <wsa:Address>
1591 (10)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
1592 (11)      </wsa:Address>
1593 (12)      <wsa:ReferenceParameters>
1594 (13)        <x:someHeader xmlns:x="..."> user-defined content </x:someHeader>
1595 (14)      </wsa:ReferenceParameters>
1596 (15)    </wsa:FaultTo>
1597 (16)    ...
1598 (17)  </s:Header>
1599 (18)  <s:Body> ... </s:Body>
1600 (19) </s:Envelope>

```

1601 **R5.4.6.3-5:** If the wsa:FaultTo address is not usable, the service should not reply to the request.
 1602 Similarly, if according to WS-Addressing processing rules there is no suitable address to send a
 1603 fault to, it should not reply and should close the network connection. In these cases, the service
 1604 should locally log some type of entry to help locate the client defect later.

1605 **R5.4.6.3-6:** The service shall properly duplicate the wsa:Address of the wsa:FaultTo element in
 1606 the wsa:To of the reply, even if some of the information is not understood by the service.

1607 This rule applies in cases where the client includes private content suffixes on the HTTP or HTTPS
 1608 address that the service does not understand. If the service removes this information when
 1609 constructing the address, the subsequent message might not be correctly processed.

1610 5.4.6.4 wsa:MessageID and wsa:RelatesTo

1611 WS-Management qualifies the use of wsa:MessageID and wsa:RelatesTo as follows:

1612 **R5.4.6.4-1:** The MessageID and RelatesTo URIs may be of any format, as long as they are valid
 1613 URIs according to [RFC 3986](#). Two URIs are considered different even if the characters in the
 1614 URIs differ only by case.

1615 The following two formats are endorsed by this specification. The first is considered a best
 1616 practice because it is backed by [RFC 4122](#):

1617 urn:uuid:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
 1618 or
 1619 uuid:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

1620 In these formats, each x is an uppercase or lowercase hexadecimal digit (lowercase is required
 1621 by [RFC 4122](#)); there are no spaces or other tokens. The value may be a DCE-style universally
 1622 unique identifier (UUID) with provable uniqueness properties in this format, however, it is not
 1623 necessary to have provable uniqueness properties in the URIs used in the wsa:MessageID and
 1624 wsa:RelatesTo headers.

1625 Regardless of format, the URI should not exceed the maximum defined in R13.1-6.

1626 UUIDs have a numeric meaning as well as a string meaning, and this can lead to confusion. A UUID
 1627 in lowercase is a different URI from the same UUID in uppercase. This is because URIs are case-
 1628 sensitive. If a UUID is converted to its decimal equivalent the case of the original characters is lost.
 1629 WS-Management works with the URI value itself, not the underlying decimal equivalent
 1630 representation. Services are free to *interpret* the URI in any way, but are not allowed to alter the case
 1631 usage when repeating the message or any of the MessageID values in subsequent messages.

1632 The [RFC 4122](#) requires the digits to be lowercase, which is the responsibility of the client. The service
 1633 simply processes the values as URI values and is not required to analyze the URI for correctness or
 1634 compliance. The service replicates the client usage in the wsa:RelatesTo reply header and is not
 1635 allowed to alter the case usage.

1636 **R5.4.6.4-2:** The MessageID should be generated according to any algorithm that ensures that no
 1637 two MessageIDs are repeated. Because the value is treated as case-sensitive (R5.4.6.4-1),
 1638 confusion can arise if the same value is reused differing only in case. As a result, the service shall
 1639 not create or employ MessageID values that differ only in case. For any message transmitted by
 1640 the service, the MessageID shall not be reused.

1641 The client ensures that MessageID values are not reused in requests. Although services and clients
 1642 can issue different MessageIDs that differ only in case, the service is not required to detect this
 1643 difference, nor is it required to analyze the URI for syntactic correctness or repeated use.

1644 **R5.4.6.4-3:** The RelatesTo element shall be present in all response messages and faults, shall
 1645 contain the MessageID of the associated request message, and shall match the original in case,
 1646 being treated as a URI value and not as a binary UUID value.

1647 **R5.4.6.4-4:** If the MessageID is not parsable or is missing, a
 1648 wsa:InvalidMessageInformationHeader fault should be returned.

1649 EXAMPLE: The following examples show wsa:MessageID usage:

```

1650 (20) <wsa:MessageID>
1651 (21)     uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91
1652 (22) </wsa:MessageID>
1653 (23)
1654 (24) <wsa:MessageID>
1655 (25)     anotherScheme:ID/12310/1231/16607/25
1656 (26) </wsa:MessageID>

```

1657 5.4.6.5 wsa:Action

1658 The wsa:Action URI indicates the "operation" being invoked against the resource.

1659 **R5.4.6.5-1:** The wsa:Action URI shall not be used to identify the specific resource class or
 1660 instance, but only to identify the operation to use against that resource.

1661 **R5.4.6.5-2:** For all resource endpoints, a service shall return a wsa:ActionNotSupported fault if a
 1662 requested action is not supported by the service for the specified resource.

1663 In other words, to model the "Get" of item "Disk", the wsa:Action URI contains the "Get". The wsa:To,
 1664 and potentially other SOAP headers, indicate *what* is being accessed. When the default addressing
 1665 model is used, for example, the ResourceURI typically contains the reference to the "Disk" and the
 1666 SelectorSet identifies which disk. Other service-specific addressing models can factor the identity of
 1667 the resource in different ways.

1668 Implementations are free to support additional custom methods that combine the notion of "Get" and
 1669 "Disk" into a single "GetDisk" action if they strive to support the separated form to maximize
 1670 interoperation. One of the main points behind WS-Management is to unify common methods
 1671 wherever possible.

1672 **R5.4.6.5-3:** If a service exposes any of the following types of capabilities, a conformant service
 1673 shall at least expose that capability using the definitions in Table 4 according to the rules of this
 1674 specification. The service may optionally expose additional similar functionality using a distinct
 1675 wsa:Action URI.

1676 **Table 4 – wsa:Action URI Descriptions**

Action URI	Description
http://schemas.xmlsoap.org/ws/2004/09/transfer/Get	Models any simple single item retrieval
http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse	Response to "Get"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Put	Models an update of an entire item
http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse	Response to "Put"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Create	Models creation of a new item
http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse	Response to "Create"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete	Models the deletion of an item
http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse	Response to "Delete"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate	Begins an enumeration or query
http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse	Response to "Enumerate"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull	Retrieves the next batch of results from enumeration
http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse	Response to "Pull"

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

1677 **R5.4.6.5-4:** A custom action may be supported if the operation is a custom method whose
 1678 semantic meaning is not present in the table.

1679 **R5.4.6.5-5:** All notifications shall contain a unique action URI that identifies the type of the event
 1680 delivery. For singleton notifications with only one event per message (the delivery mode
 1681 <http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push>), the wsa:Action URI

1682 defines the event type. For other delivery modes, the Action varies, as described in clause 10.2.7
1683 of this specification.

1684 5.4.6.6 wsa:From

1685 The wsa:From header can be used in any messages, responses, or events to indicate the source.
1686 When the same connection is used for both request and reply, this header provides no useful
1687 information, but can be useful in cases where the response arrives on a different connection.

1688 **R5.4.6.6-1:** A conformant service may include a wsa:From address in the message. A
1689 conformant service should process any incoming message that has a wsa:From element.

1690 **R5.4.6.6-2:** A conformant service should not fault any message with a wsa:From element,
1691 regardless of whether the mustUnderstand attribute is included.

1692 NOTE: Processing the wsa:From header is trivial because it has no effect on the meaning of the
1693 message. The *From* address is primarily for auditing and logging purposes.

1694 6 WS-Management Control Headers

1695 WS-Management defines several SOAP headers that can be used with any operation.

1696 6.1 wsman:OperationTimeout

1697 Most management operations are time-critical due to quality-of-service constraints and obligations. If
1698 operations cannot be completed in a specified time, the service returns a fault so that a client can
1699 comply with its obligations. The following header value can be supplied with any WS-Management
1700 message to indicate that the client expects a response or a fault within the specified time:

1701 (1) `<wsman:OperationTimeout> xs:duration </wsman:OperationTimeout>`

1702 **R6.1-1:** All request messages may contain a wsman:OperationTimeout header element that
1703 indicates the maximum amount of time the client is willing to wait for the service to issue a
1704 response. The service should interpret the timeout countdown as beginning from the point the
1705 message is processed until a response is generated.

1706 **R6.1-2:** The service should *immediately* issue a wsman:TimedOut fault if the countdown time is
1707 exceeded and the operation is not yet complete. If the OperationTimeout value is not valid, a
1708 wsa:InvalidMessageInformationHeader fault should be returned.

1709 **R6.1-3:** If the service does not support user-defined timeouts, a wsman:UnsupportedFeature
1710 fault should be returned with the following detail code:

1711 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OperationTimeout`

1712 **R6.1-4:** If the wsman:OperationTimeout element is omitted, the service may interpret this
1713 omission as an instruction to block indefinitely until a response is available, or it may impose a
1714 default timeout.

1715 These rules do not preclude services from supporting infinite or very long timeouts. Because network
1716 connections seldom block indefinitely with no traffic occurring, some type of transport timeout is likely.
1717 Also the countdown is initiated from the time the message is received, so network latency is not
1718 included. If a client needs to discover the range of valid timeouts or defaults, metadata can be
1719 retrieved, but the format of such metadata is beyond the scope of this specification.

1720 If the timeout occurs in such a manner that the service has already performed some of the work
1721 associated with the request, the service state reaches an anomalous condition. This specification
1722 does not attempt to address behavior in this situation. Clearly, services can attempt to undo the

1723 effects of any partially complete operations, but this is not always practical. In such cases, the service
1724 can keep a local log of requests and operations, which the client can query later.

1725 For example, if a Delete operation is in progress and a timeout occurs, the service decides whether to
1726 attempt a rollback or roll-forward of the deletion, even though it issues a wsman:TimedOut fault. The
1727 service can elect to include additional information in the fault (see 14.5) regarding its internal policy in
1728 this regard. The service can attempt to return to the state that existed before the operation was
1729 attempted, but this is not always possible.

1730 **R6.1-5:** If the mustUnderstand attribute is applied to the wsman:OperationTimeout element and
1731 the service understands wsman:OperationTimeout, the service shall observe the requested value
1732 or return the fault specified in R6.1-2. The service should attempt to complete the request within
1733 the specified time or issue a fault without any further delay.

1734 Clients can always omit the mustUnderstand header for uniform behavior against all implementations.
1735 It is not an error for a compliant service to ignore the timeout value or treat it as a hint if
1736 mustUnderstand is omitted.

1737 EXAMPLE: The following is an example of a correctly formatted 30-second timeout in the SOAP header:

1738 (1) `<wsman:OperationTimeout>PT30S</wsman:OperationTimeout>`

1739 If the transport timeout occurs before the actual wsman:OperationTimeout, the operation can be
1740 treated as specified in 13.3, the same as a failed connection. In practice, the network transport
1741 timeout can be configured to be longer than any expected wsman:OperationTimeout.

1742 6.2 wsman:MaxEnvelopeSize

1743 To prevent a response beyond the capability of the client, the request message can contain a
1744 restriction on the response size.

1745 The following header value may be supplied with any WS-Management message to indicate that the
1746 client expects a response whose total SOAP envelope does not exceed the specified number of
1747 octets:

1748 (1) `<wsman:MaxEnvelopeSize>xs:positiveInteger </wsman:MaxEnvelopeSize>`

1749 The limitation is on the entire envelope. Resource-constrained implementations need a reliable figure
1750 for the required amount of memory for all SOAP processing, not just the SOAP Body.

1751 **R6.2-1:** All request messages may contain a wsman:MaxEnvelopeSize header element that
1752 indicates the maximum number of octets (not characters) in the entire SOAP envelope in the
1753 response. If the service cannot compose a reply within the requested size, it should return a
1754 wsman:EncodingLimit fault with the following detail code:

1755 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize`

1756 **R6.2-2:** If the mustUnderstand attribute is set to "true", the service shall comply with the
1757 request. If the response would exceed the maximum size, the service should return a
1758 wsman:EncodingLimit fault. Because a service might execute the operation prior to knowing the
1759 response size, the service should undo any effects of the operation before issuing the fault. If the
1760 operation cannot be reversed (such as a destructive Put or Delete, or a Create), the service shall
1761 indicate that the operation succeeded in the wsman:EncodingLimit fault with the following detail
1762 code:

1763 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess`

1764 **R6.2-3:** If the mustUnderstand attribute is set to "false", the service may ignore the header.

1765 **R6.2-4:** Services should reject any MaxEnvelopeSize value less than 8192 octets. This number
 1766 is the safe minimum in which faults can be reliably encoded for all character sets. If the requested
 1767 size is less than this, the service should return a wsman:EncodingLimit fault with the following
 1768 detail code:

1769 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MinimumEnvelopeLimit`

1770 A service might have its own encoding limit independent of what the client specifies, and the same
 1771 fault applies.

1772 **R6.2-5:** If the service cannot compose a reply within its own internal limits, the service should
 1773 return a wsman:EncodingLimit fault with the following detail code:

1774 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit`

1775 The definition of the wsman:MaxEnvelopeSize element in the schema contains a Policy attribute
 1776 because this element is used for other purposes. This specification does not define a meaning for the
 1777 Policy attribute when the wsman:MaxEnvelopeSize element is used as a SOAP header.

1778 **R6.2-6:** Clients should not add the Policy attribute to the wsman:MaxEnvelopeSize element
 1779 when it is used as a SOAP header. Services should ignore the Policy attribute if it appears in the
 1780 wsman:MaxEnvelopeSize element when used as a SOAP header.

1781 **6.3 wsman:Locale**

1782 Management operations often span locales, and many items in responses can require translation.
 1783 Typically, translation is required for descriptive information, intended for human readers, that is sent
 1784 back in the response. If the client requires such output to be translated into a specific language, it can
 1785 employ the optional wsman:Locale header, which makes use of the standard XML attribute xml:lang,
 1786 as follows:

1787 `(1) <wsman:Locale xml:lang="xs:language" s:mustUnderstand="false"/>`

1788 **R6.3-1:** If the mustUnderstand attribute is omitted or set to "false", the service should use this
 1789 value when composing the response message and adjust any localizable values accordingly.
 1790 This use is recommended for most cases. The locale is treated as a hint in this case.

1791 **R6.3-2:** If the mustUnderstand attribute is set to "true", the service shall ensure that the replies
 1792 contain localized information where appropriate, or else the service shall issue a
 1793 wsman:UnsupportedFeature fault with the following detail code:

1794 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale`

1795 A service may always fault if wsman:Locale contains s:mustUnderstand set to "true", because it
 1796 may not be able to ensure that the reply is localized.

1797 Some implementations delegate the request to another subsystem for processing, so the service
 1798 cannot be certain that the localization actually occurred.

1799 **R6.3-3:** The value of the xml:lang attribute in the wsman:Locale header shall be a valid
 1800 [RFC 5646](#) language code.

1801 **R6.3-4:** In any response, event, or singleton message, the service should include the xml:lang
 1802 attribute in the s:Envelope (or other elements) to signal to the receiver that localized content
 1803 appears in the body of the message. This attribute may be omitted if no descriptive content
 1804 appears in the body. Including the xml:lang attribute is not an error, even if no descriptive content
 1805 occurs.

EXAMPLE:

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

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

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

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

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

6.4 wsman:OptionSet

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

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

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

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

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

R6.4-4: Each resource class may observe its own set of options, and an individual instance of that resource class may further observe its own set of options. Consistent option usage is not

1852 required across resource class and instance boundaries. The metadata formats and definitions of
1853 options are beyond the scope of this specification and may be service-specific.

1854 **R6.4-5:** Any number of individual option elements may appear under the `wsman:OptionSet`
1855 wrapper. Option names may be repeated if appropriate. The content shall be a simple string
1856 (`xs:string`). This specification places no restrictions on whether the names or values are to be
1857 treated in a case-sensitive or case-insensitive manner. However, case usage shall be retained as
1858 the message containing the `OptionSet` element and its contents are propagated through SOAP
1859 intermediaries.

1860 Interpretation of the option with regard to case sensitivity is up to the service and the definition of the
1861 specific option because the value might be passed through to real-world subsystems that
1862 inconsistently expose case usage. Where interoperability is a concern, the client can omit both
1863 `mustUnderstand` and `MustComply` attributes.

1864 **R6.4-6:** Individual option values may be advisory or may be required by the client. The service
1865 shall observe and execute any option marked with the `MustComply` attribute set to "true", or
1866 return a `wsman:InvalidOptions` fault with the following detail code:

1867 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported`

1868 Any option not marked with this attribute (or if the attribute is set to "false") is advisory to the
1869 service, and the service may ignore it. If any option is marked with `MustComply` set to "true", then
1870 the `mustUnderstand` attribute shall be used on the entire `wsman:OptionSet` block.

1871 This capability is required when the service delegates interpretation and execution of the options
1872 to another component. In many cases, the SOAP processor cannot know if the option was
1873 observed and can only pass it along to the next subsystem.

1874 **R6.4-7:** Options may optionally contain a `Type` attribute, which indicates the data type of the
1875 content of the `Option` element. A service may require that this attribute be present on any given
1876 option and that it be set to the `QName` of a valid XML schema data type. Only the standard
1877 simple types declared in the `http://www.w3.org/2001/XMLSchema` namespace are supported in
1878 this version of WS-Management.

1879 This rule can help some services distinguish numeric or date/time types from other string values.

1880 **R6.4-8:** Options should not be used as a replacement for the documented parameterization
1881 technique for the message; they should be used only as a modifier for it.

1882 Options are primarily used to establish context or otherwise instruct the service to perform side-band
1883 operations while performing the operation, such as turning on logging or tracing.

1884 **R6.4-9:** The following faults should be returned by the service:

- 1885 • when options are not supported, **`wsman:InvalidOptions`** with the following detail code:

1886 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported`

- 1887 • when one or more option names are not valid or supported by the specific
1888 resource, **`wsman:InvalidOptions`** with the following detail code:

1889 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName`

- 1890 • when the value is not correct for the option name, **`wsman:InvalidOptions`** with the
1891 following detail code:

1892 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue`

1893 **R6.4-10:** For operations that span multiple message sequences, the `wsman:OptionSet` element
1894 is processed in the initial message only. It should be ignored in subsequent messages because

1895 the first message establishes the required set of options. The service may issue a fault if the
 1896 wsman:OptionSet is present in subsequent messages and the value is different from that used in
 1897 the initiating request, or the service may ignore the values of wsman:OptionSet in such
 1898 messages.

1899 This rule applies primarily to Enumerate and Pull messages. The set of options is established once
 1900 during the initial Enumerate request, so changing the options during the enumeration would constitute
 1901 an error.

1902 Options are intended to make operations more efficient or to preprocess output on behalf of the client.
 1903 For example, the options could indicate to the service that the returned values are to be recomputed
 1904 and that cached values are not to be used, or that any optional values in the reply may be omitted.
 1905 Alternately, the options could be used to indicate verbose output within the limits of the XML schema
 1906 associated with the reply.

1907 Option values are not intended to contain XML. If XML-based input is required, a custom operation
 1908 with its own wsa:Action URI is the correct model for the operation. This ensures that no backdoor
 1909 parameters are introduced over well-known message types. For example, when issuing a Subscribe
 1910 request, the message already defines a technique for passing an event filter to the service, so the
 1911 option is not used to circumvent this and pass a filter using an alternate method.

1912 EXAMPLE: The following is an example of wsman:OptionSet:

```

1913 (1) <s:Envelope
1914 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
1915 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
1916 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
1917 (5)   xmlns:xs="http://www.w3.org/2001/XMLSchema">
1918 (6)   <s:Header>
1919 (7)     ...
1920 (8)     <wsman:OptionSet s:mustUnderstand="true">
1921 (9)       <wsman:Option Name="VerbosityLevel" Type="xs:int">
1922 (10)        3
1923 (11)      </wsman:Option>
1924 (12)      <wsman:Option Name="LogAllRequests" MustComply="true"/>
1925 (13)    </wsman:OptionSet>
1926 (14)    ...
1927 (15)  </s:Header>
1928 (16)  <s:Body> .. </s:Body>
1929 (17) </s:Envelope>
  
```

1930 The following definitions provide additional, normative constraints on the preceding outline:

1931 wsman:OptionSet

1932 used to wrap individual option blocks

1933 In this example, s:mustUnderstand is set to "true", indicating that the client is requiring the
 1934 service to process the option block using the given rules.

1935 wsman:OptionSet/wsman:Option/@Name

1936 identifies the option (an xs:string), which may be a simple name or a URI

1937 This name is scoped to the resource to which it applies. The name may be repeated in
 1938 subsequent elements. The name cannot be blank and can be a short non-colliding URI that is
 1939 vendor-specific.

1940 wsman:OptionSet/wsman:Option/@MustComply

1941 if set to "true", indicates that the option shall be observed; otherwise, indicates an advisory or a
 1942 hint

1943 wsman:OptionSet/wsman:Option/@Type
 1944 (optional) if present, indicates the data type of the element content, which helps the service to
 1945 interpret the content
 1946 A service may require this attribute to be present on any given option element.

1947 wsman:OptionSet/wsman:Option
 1948 the content of the option
 1949 The value may be any simple string value. If the option value is empty, the option should be
 1950 interpreted as logically "true", and the option should be "enabled". The following example
 1951 enables the "Verbose" option:

1952 (1) `<wsman:Option Name="Verbose"/>`

1953 Options are logically false if they are not present in the message. All other cases require an explicit
 1954 string to indicate the option value. The reasoning for allowing the same option to repeat is to allow
 1955 specification of a list of options of the same name.

1956 6.5 wsman:RequestEPR

1957 Some service operations, including "Put", are able to modify the resource representation in such a
 1958 way that the update results in a logical identity change for the resource, such as the "rename" of a
 1959 document. In many cases, this modification in turn alters the EPR of that resource after the operation
 1960 is completed, as EPRs are often dynamically derived from naming values within the resource
 1961 representation itself. This behavior is common in SOAP implementations that delegate operations to
 1962 underlying systems.

1963 To provide the client a way to determine when such a change has happened, two SOAP headers are
 1964 defined to request and return the EPR of a resource instance.

1965 In any WS-Management request message, the following header may appear:

1966 (1) `<wsman:RequestEPR .../>`

1967 **R6.5-1:** A service receiving a message that contains the wsman:RequestEPR header block
 1968 should return a response that contains a wsman:RequestedEPR header block. This block
 1969 contains the most recent EPR of the resource being accessed or a status code if the service
 1970 cannot determine or return the EPR. This EPR reflects any identity changes that may have
 1971 occurred as a result of the current operation, as set forth in the following behavior. The header
 1972 block in the corresponding response message has the following format:

1973 (1) `<wsman:RequestedEPR ...>`
 1974 (2) `[ <wsa:EndpointReference>`
 1975 (3) `wsa:EndpointReferenceType`
 1976 (4) `</wsa:EndpointReference> |`
 1977 (5) `<wsman:EPRInvalid/> |`
 1978 (6) `<wsman:EPRUnknown/> ]`
 1979 (7) `</wsman:RequestedEPR>`

1980 The following definitions describe additional, normative constraints on the preceding format:

1981 wsman:RequestedEPR/wsa:EndpointReference

1982 one of three elements that can be returned as a child element of the wsman:RequestedEPR
 1983 element

1984 The use of this element indicates that the service understood the request to return the EPR of
 1985 the resource and is including the EPR of the resource. The returned EPR is calculated after all
 1986 intentional effects or side effects of the associated request message have occurred. The EPR
 1987 may not have changed as a result of the operation, but the service is still obligated to return it.

- 1988 wsman:RequestedEPR/wsman:EPRInvalid
 1989 one of three elements that can be returned as a child element of the wsman:RequestedEPR
 1990 element
 1991 The use of this element (no value is required) indicates that the service understands the request
 1992 to return the EPR of the resource but is unable to calculate a full EPR. However, the service is
 1993 able to determine that this message exchange has modified the resource representation in such
 1994 a way that any previous references to the resource are no longer valid. When EPRInvalid is
 1995 returned, the client shall not use the old wsa:EndpointReference in subsequent operations.
- 1996 wsman:RequestedEPR/wsman:EPRUnknown
 1997 one of three elements that can be returned as a child element of the wsman:RequestedEPR
 1998 element
 1999 The use of this element (no value is required) indicates that the service understands the request
 2000 to return the EPR of the resource but is unable to determine whether existing references to the
 2001 resource are still valid. When EPRUnknown is returned, the client may attempt to use the old
 2002 wsa:EndpointReference in subsequent operations. The result of using an old
 2003 wsa:EndpointReference, however, is unpredictable; a result may be a fault or a successful
 2004 response.

2005 7 Resource Access

2006 7.1 General

- 2007 Resource access applies to all synchronous operations regarding getting, setting, and enumerating
 2008 values. The subclauses in clause 7 define a mechanism for acquiring management-specific XML-
 2009 based representations of entities using the Web service infrastructure, such as managed resources.
- 2010 Specifically, two operations are defined for sending and receiving the management representation of
 2011 a given resource and two operations are defined for creating and deleting a management resource
 2012 and its corresponding representation. Multi-instance retrieval is achieved using the enumeration
 2013 messages. This specification does not define any messages or techniques for batched operations,
 2014 such as batched Get or Delete. All such operations can be sent as a series of single messages.
- 2015 It should be noted that the state maintenance of a resource is at most subject to the "best efforts" of
 2016 the hosting server. When a client receives the server's acceptance of a request to create or update a
 2017 resource, it can reasonably expect that the resource now exists at the confirmed location and with the
 2018 confirmed representation, but this is not a guarantee, even in the absence of any third parties. The
 2019 server may change the representation of a resource, may remove a resource entirely, or may bring
 2020 back a resource that was deleted.
- 2021 For instance, the server may store resource state information on a disk drive. If that drive crashes and
 2022 the server recovers state information from a backup tape, changes that occurred after the backup
 2023 was made would be lost.
- 2024 A server may have other operational processes that change resource state information. A server may
 2025 run a background process that examines resources for objectionable content and deletes any such
 2026 resources it finds. A server may purge resources that have not been accessed for some period of
 2027 time. A server may apply storage quotas that cause it to occasionally purge resources.
- 2028 In essence, the confirmation by a service of having processed a request to create, modify, or delete a
 2029 resource implies a commitment only at the instant that the confirmation was generated. While the
 2030 usual case should be that resources are long-lived and stable, there are no guarantees, and clients
 2031 should code defensively.
- 2032 There is no requirement for uniformity in resource representations between the messages defined in
 2033 this specification. For example, the representations required by Create or Put may differ from the

2034 representation returned by Get, depending on the semantic requirements of the service. Additionally,
 2035 there is no requirement that the resource content is fixed for any given endpoint reference. The
 2036 resource content may vary based on environmental factors, such as the security context, time of day,
 2037 configuration, or the dynamic state of the service.

2038 As per the SOAP processing model, other specifications may define SOAP headers that may be
 2039 optionally added to request messages to require the transfer of subsets or the application of
 2040 transformations of the resource associated with the endpoint reference. When the Action URIs
 2041 defined by this specification are used, such extension specifications must also allow the basic
 2042 processing models defined herein.

2043 NOTE: The WSDL for the resource access operations (see ANNEX G), as well as the pseudo schema and
 2044 example message fragments throughout clause 7, is not usable as represented without first replacing the
 2045 "resource-specific-GED" text with the application-defined GED.

2046 EXAMPLE 1: Following is a full example of a hypothetical Get request:

```

2047 (1) <s:Envelope
2048 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
2049 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
2050 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
2051 (5)   <s:Header>
2052 (6)     <wsa:To>http://1.2.3.4/wsman/</wsa:To>
2053 (7)     <wsman:ResourceURI>http://example.org/2005/02/physicalDisk
2054 (8)       </wsman:ResourceURI>
2055 (9)     <wsa:ReplyTo>
2056 (10)      <wsa:Address>
2057 (11)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
2058 (12)      </wsa:Address>
2059 (13)    </wsa:ReplyTo>
2060 (14)    <wsa:Action>
2061 (15)      http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
2062 (16)    </wsa:Action>
2063 (17)    <wsa:MessageID>
2064 (18)      urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
2065 (19)    </wsa:MessageID>
2066 (20)    <wsman:SelectorSet>
2067 (21)      <wsman:Selector Name="LUN"> 2 </wsman:Selector>
2068 (22)    </wsman:SelectorSet>
2069 (23)    <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
2070 (24)  </s:Header>
2071 (25)  <s:Body/>
2072 (26) </s:Envelope>
  
```

2073 Notice that the `wsa:ReplyTo` indicates the response is to be sent on the same connection as the
 2074 request (line 10), the action is a Get (line 14), and the ResourceURI (line 7) and `wsman:SelectorSet`
 2075 (line 20) are used to address the requested management information. This example assumes that the
 2076 WS-Management default addressing model is in use. The service is expected to complete the
 2077 operation in 30 seconds or return a fault to the client (line 22).

2078 Also, the `s:Body` in a Get request has no content.

2079 EXAMPLE 1 (continued): The following shows a hypothetical response to the preceding hypothetical Get request:

```

2080 (26) <s:Envelope
2081 (27)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
2082 (28)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
2083 (29)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
2084 (30)   <s:Header>
2085 (31)     <wsa:To>
2086 (32)       http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
  
```

```

2087 (33) </wsa:To>
2088 (34) <wsa:Action s:mustUnderstand="true">
2089 (35) http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
2090 (36) </wsa:Action>
2091 (37) <wsa:MessageID s:mustUnderstand="true">
2092 (38) urn:uuid:217a431c-b071-3301-9bb8-5f538bec89b8
2093 (39) </wsa:MessageID>
2094 (40) <wsa:RelatesTo>
2095 (41) urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
2096 (42) </wsa:RelatesTo>
2097 (43) </s:Header>
2098 (44) <s:Body>
2099 (45) <PhysicalDisk
2100 (46) xmlns="http://schemas.example.org/2005/02/samples/physDisk">
2101 (46) <Manufacturer> Acme, Inc. </Manufacturer>
2102 (47) <Model> 123-SCSI 42 GB Drive </Model>
2103 (48) <LUN> 2 </LUN>
2104 (49) <Cylinders> 16384 </Cylinders>
2105 (50) <Heads> 80 </Heads>
2106 (51) <Sectors> 63 </Sectors>
2107 (52) <OctetsPerSector> 512 </OctetsPerSector>
2108 (53) <BootPartition> 0 </BootPartition>
2109 (54) </PhysicalDisk>
2110 (55) </s:Body>
2111 (56) </s:Envelope>

```

2112 Notice that the response uses the wsa:To address (line 32) that the original request had specified in
 2113 wsa:ReplyTo. Also, the wsa:MessageID for this response is unique (line 38). The wsa:RelatesTo
 2114 (line 41) contains the UUID of the wsa:MessageID of the original request to allow the client to
 2115 correlate the response.

2116 The s:Body (lines 44-55) contains the requested resource representation.

2117 The same general approach exists for Delete, except that no content exists in the response s:Body.
 2118 The Create and Put operations are similar, except that they contain content in the request s:Body to
 2119 specify the values being created or updated.

2120 7.2 Addressing Uniformity

2121 Where practical, the EPR of the resource can be the same whether a Get, Delete, or Put operation is
 2122 being used. This is not a strict requirement, but it reduces the education and training required to
 2123 construct and use WS-Management-aware tools.

2124 Create is a special case, in that the EPR of the newly created resource is often not known until the
 2125 resource is actually created. For example, although it might be possible to return running process
 2126 information using a hypothetical *ProcessID* in an addressing header, it is typically not possible to
 2127 assert the *ProcessID* during the creation phase because the underlying system does not support the
 2128 concept. Thus, the Create operation would not have the same addressing headers as the
 2129 corresponding Get or Delete operations.

2130 If the WS-Management default addressing model is in use, it would be typical to use the
 2131 ResourceURI as a "type" and selector values for "instance" identification. Thus, the same address
 2132 would be used for Get, Put, and Delete when working with the same instance. When enumerating all
 2133 instances, the selectors would be omitted and the ResourceURI would be used alone to indicate the
 2134 "type" of the object being enumerated. The Create operation might also share this usage, or have its
 2135 own ResourceURI and selector usage (or not even use selectors). This pattern is not a requirement.

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

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

2148 7.3 Get

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

The Get request message shall be of the following form:

```

2153 (1) <s:Envelope ...>
2154 (2)   <s:Header ...>
2155 (3)     <wsa:Action>
2156 (4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
2157 (5)     </wsa:Action>
2158 (6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
2159 (7)     <wsa:To>xs:anyURI</wsa:To>
2160 (8)     ...
2161 (9)   </s:Header>
2162 (10)  <s:Body .../>
2163 (11) </s:Envelope>

```

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

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

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

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

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

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

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

```

2182 (1) <s:Envelope ...>

```

```

2183 (2) <s:Header ...>
2184 (3) <wsa:Action>
2185 (4) http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
2186 (5) </wsa:Action>
2187 (6) <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
2188 (7) <wsa:To>xs:anyURI</wsa:To>
2189 (8) ...
2190 (9) </s:Header>
2191 (10) <s:Body ...>
2192 (11) resource-specific-element
2193 (12) </s:Body>
2194 (13) </s:Envelope>

```

2195 The following describes additional, normative constraints on the preceding outline:

2196 /s:Envelope/s:Header/wsa:Action

2197 This required element shall contain the value
 2198 http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse. If a SOAP Action URI is also
 2199 present in the underlying transport, its value shall convey the same value.

2200 /s:Envelope/s:Body/child

2201 The representation itself shall be the child element of the SOAP:Body element of the response
 2202 message.

2203 Other components of the preceding outline are not further constrained by this specification.

2204 The Get operation retrieves resource representations. The message can be targeted to return a
 2205 complex XML document or to return a single, simple value. The nature and complexity of the
 2206 representation is not constrained by this specification.

2207 **R7.3-1:** A conformant service should support Get operations to service metadata requests
 2208 about the service itself or to verify the result of a previous action or operation.

2209 This statement does not constrain implementations from supplying additional similar methods for
 2210 resource and metadata retrieval.

2211 **R7.3-2:** Execution of Get should not in itself have side effects on the value of the resource.

2212 **R7.3-3:** If an object cannot be retrieved due to locking conditions, simultaneous access, or
 2213 similar conflicts, a wsman:Concurrency fault should be returned.

2214 In practice, Get is designed to return XML that corresponds to real-world objects. To retrieve
 2215 individual property values, either the client can postprocess the XML content for the desired value, or
 2216 the service can support fragment-level access (7.7).

2217 Fault usage is generally as described in clause 14. An inability to locate or access the resource is
 2218 equivalent to problems with the SOAP message when the EPR is defective. There are no "Get-
 2219 specific" faults.

2220 7.4 Put

2221 A Web service operation (Put) is defined for updating a resource by providing a replacement
 2222 representation. A resource may accept updates that provide different XML representations than that
 2223 returned by the resource; in such a case, the semantics of the update operation is defined by the
 2224 resource.

2225 The Put request message shall be of the following form:

```

2226 (1) <s:Envelope ...>
2227 (2) <s:Header ...>

```

```

2228 (3)      <wsa:Action>
2229 (4)      http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
2230 (5)      </wsa:Action>
2231 (6)      <wsa:MessageID>xs:anyURI</wsa:MessageID>
2232 (7)      <wsa:To>xs:anyURI</wsa:To>
2233 (8)      ...
2234 (9)      </s:Header>
2235 (10)     <s:Body...>
2236 (11)     resource-specific-element
2237 (12)     </s:Body>
2238 (13)    </s:Envelope>

```

2239 The following describes additional, normative constraints on the preceding outline:

2240 /s:Envelope/s:Header/wsa:Action

2241 This required element shall contain the value
 2242 http://schemas.xmlsoap.org/ws/2004/09/transfer/Put. If a SOAP Action URI is also present in the
 2243 underlying transport, its value shall convey the same value.

2244 /s:Envelope/s:Body/child

2245 The representation to be used for the update shall be the child element of the s:Body element of
 2246 the request message.

2247 A Put request shall be targeted at the resource whose representation is desired to be replaced. As
 2248 per the SOAP processing model, other specifications may introduce various types of extensions to
 2249 this message, which are enabled through headers tagged with s:mustUnderstand="true". Such
 2250 extensions may require that a full or partial update should be accomplished using symbolic,
 2251 instruction-based, or other methodologies.

2252 Extension specifications may also define extensions to the original Put request, enabled by optional
 2253 SOAP headers, which control the nature of the response (see the information about PutResponse
 2254 later in this clause).

2255 Specifications that define any of these extensions shall allow processing of the Put message without
 2256 such extensions.

2257 In addition to the standard fault codes defined in Addressing, implementations may use the fault code
 2258 wsmt:InvalidRepresentation if the presented representation is invalid for the target resource. Other
 2259 components of the preceding outline are not further constrained by this specification.

2260 A successful Put operation updates the current representation associated with the targeted resource.

2261 If the resource accepts a Put request and performs the requested update, it shall reply with a
 2262 response of the following form:

```

2263 (1) <s:Envelope ...>
2264 (2) <s:Header ...>
2265 (3) <wsa:Action>
2266 (4) http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse
2267 (5) </wsa:Action>
2268 (6) <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
2269 (7) <wsa:To>xs:anyURI</wsa:To>
2270 (8) ...
2271 (9) </s:Header>
2272 (10) <s:Body ...>
2273 (11) resource-specific-element ?
2274 (12) </s:Body>
2275 (13) </s:Envelope>

```

- 2276 /s:Envelope/s:Header/wsa:Action
 2277 This required element shall contain the value
 2278 http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse. If a SOAP Action URI is also
 2279 present in the underlying transport, its value shall convey the same value.
- 2280 /s:Envelope/s:Body/child
 2281 An implementation of a service shall choose, in advance, whether to return an empty Body or the
 2282 resulting representation of the resource. This choice shall be explicitly stated in the WSDL, if
 2283 WSDL is provided.
 2284 By default, a service shall return the current representation of the resource as the child of the
 2285 s:Body element if the updated representation differs from the representation sent in the Put
 2286 request message.
 2287 As an optimization and as a service to the requester, the s:Body element of the response
 2288 message should be empty if the updated representation does not differ from the representation
 2289 sent in the Put request message; that is, if the service accepted the new representation
 2290 verbatim.
 2291 Such a response (an empty s:Body) implies that the update request was successful in its entirety
 2292 (assuming no intervening mutating operations are performed). A service may return the current
 2293 representation of the resource as the initial child of the s:Body element even in this case,
 2294 however.
- 2295 Extension specifications may define extensions to the original Put request, enabled by optional
 2296 header values, in order to optimize the response. In the absence of such headers, the behavior shall
 2297 be as previously described. Specifications that define any of these extensions shall allow processing
 2298 the Put message without such extensions. Because the response may not be sent to the original
 2299 sender, extension specifications should consider adding a corresponding SOAP header value in the
 2300 response to signal to the receiver that the extension is being used.
- 2301 Other components of the preceding outline are not further constrained by this specification.
- 2302 If a resource can be updated in its entirety within the constraints of the corresponding XML schema
 2303 for the resource, the service can support the Put operation.
- 2304 **R7.4-1:** A conformant service may support Put.
- 2305 **R7.4-2:** If a single resource instance can be updated (within the constraints of its schema) by
 2306 using a SOAP message, and that resource subsequently can be retrieved using Get, a service
 2307 should support updating the resource by using Put. The service may additionally export a custom
 2308 method for updates.
- 2309 **R7.4-3:** If a single resource instance contains a mix of modifiable and non-modifiable
 2310 properties, the Put message may contain values for both the modifiable and non-modifiable
 2311 properties if the XML content is legal with regard to its XML schema namespace. If the Put
 2312 message contains values for modifiable properties, the service shall set these properties to these
 2313 values during the Put operation. If the Put message contains values for non-modifiable properties,
 2314 the service should ignore those values during the Put operation. If none of the properties are
 2315 modifiable, the service should return a wsa:ActionNotSupported fault.
- 2316 This situation typically happens if a Get operation is performed, a value is altered, and the entire
 2317 updated representation is sent using Put. In this case, any read-only values would still be present.
- 2318 A complication arises because Put contains the complete new representation for the instance. If the
 2319 resource schema requires the presence of any given value (minOccurs is not zero), it will be supplied
 2320 as part of the Put message, even if it is not being altered from its original value.
- 2321 **R7.4-4:** If a Put operation specifies a modifiable value as NULL using the xsi:nil attribute, then
 2322 the service shall set the value to NULL.

2323 If the schema definition includes elements that are optional (minOccurs=0), the Put message can omit
 2324 these values. Existing implementations provide two different responses when these elements are
 2325 modifiable (writable). They either set the omitted element's value to NULL or leave the value
 2326 unchanged. Given this reality, the following rules apply:

2327 **R7.4-5:** Any modifiable properties that are optional in the XML schema (that is, minOccurs="0")
 2328 and that are omitted from the Put message shall either be set to a resource-specific default
 2329 value or be left unchanged. Setting to a resource specific default value is recommended.

2330 NOTE 1: Elements not set may have their value changed as a result of other constraints.

2331 NOTE 2: The resource-specific default value is outside the scope of this specification.

2332 To update isolated values without having to supply all values, use the fragment-level resource access
 2333 mechanism described in 7.7.

2334 In short, the s:Body of the Put message complies with the constraints of the associated XML schema.

2335 EXAMPLE 1: For example, assume that Get returns the following information:

```
2336 (1) <s:Body>
2337 (2)   <MyObject xmlns="examples.org/2005/02/MySchema">
2338 (3)     <A> 100 </A>
2339 (4)     <B> 200 </B>
2340 (5)     <C> 100 </C>
2341 (6)   </MyObject>
2342 (7) </s:Body>
```

2343 EXAMPLE 2: The corresponding XML schema has defined A, B, and C as minOccurs=1:

```
2344 (8) <xs:element name="MyObject">
2345 (9)   <xs:complexType>
2346 (10)    <xs:sequence>
2347 (11)      <xs:element name="A" type="xs:int" minOccurs="1" maxOccurs="1"/>
2348 (12)      <xs:element name="B" type="xs:int" minOccurs="1" maxOccurs="1"/>
2349 (13)      <xs:element name="C" type="xs:int" minOccurs="1" maxOccurs="1"/>
2350 (14)      ...
2351 (15)    </xs:sequence>
2352 (16)  </xs:complexType>
2353 (17) </xs:element>
```

2354 In this case, the corresponding Put needs to contain all three elements because the schema mandates that all
 2355 three be present. Even if the only value being updated is , the client has to supply all three values. This
 2356 usually means that the client first has to issue a Get to preserve the current values of <A> and <C>, change
 2357 to the desired value, and then write the object using Put. As noted in R7.4-3, the service can ignore attempts to
 2358 update values that are read-only with regard to the underlying real-world object.

2359 **R7.4-6:** A conformant service should support Put using the same EPR as a corresponding Get
 2360 or other messages, unless the Put mechanism for a resource is semantically distinct.

2361 **R7.4-7:** If the supplied Body does not have the correct content to update the resource, the
 2362 service should return a wsmt:InvalidRepresentation fault and detail codes as follows:

- 2363 • if any values in the s:Body are not correct:
 2364 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues>
- 2365 • if any values in the s:Body are missing:
 2366 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues>

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

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

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

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

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

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

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

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

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

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

2392 7.5 Delete

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

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

The Delete request message shall be of the following form:

```

2400 (1) <s:Envelope ...>
2401 (2)   <s:Header ...>
2402 (3)     <wsa:Action>
2403 (4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
2404 (5)     </wsa:Action>
2405 (6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
2406 (7)     <wsa:To>xs:anyURI</wsa:To>
2407 (8)     ...
2408 (9)   </s:Header>
2409 (10)  <s:Body ... />
2410 (11) </s:Envelope>

```

2411 The following describes additional, normative constraints on the preceding outline:

2412 /s:Envelope/s:Header/wsa:Action

2413 This required element shall contain the value

2414 http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete. If a SOAP Action URI is also present in
2415 the underlying transport, its value shall convey the same value.

2416 A Delete request shall be targeted at the resource to be deleted.

2417 There are no body blocks defined for a Delete Request.

2418 Implementations may respond with a fault message using the standard fault codes defined in
2419 Addressing (for example, wsa:ActionNotSupported). Other components of the preceding outline are
2420 not further constrained by this specification.

2421 A successful Delete operation invalidates the current representation associated with the targeted
2422 resource.

2423 If the resource accepts a Delete request, it shall reply with a response of the following form:

```
2424 (1) <s:Envelope ...>
2425 (2)   <s:Header ...>
2426 (3)     <wsa:Action>
2427 (4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse
2428 (5)     </wsa:Action>
2429 (6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
2430 (7)     <wsa:To>xs:anyURI</wsa:To>
2431 (8)     ...
2432 (9)   </s:Header>
2433 (10)  <s:Body .../>
2434 (11) </s:Envelope>
```

2435 /s:Envelope/s:Header/wsa:Action

2436 This required element shall contain the value

2437 http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse. If a SOAP Action URI is also
2438 present in the underlying transport, its value shall convey the same value.

2439 By default, there are no s:Body blocks defined for a Delete response. Specifications that define
2440 extensions for use in the original Delete request that control the format of the response shall allow
2441 processing the Delete message without such extensions.

2442 Other components of the preceding outline are not further constrained by this specification.

2443 In general, the addressing can be the same as for a corresponding Get operation for uniformity, but
2444 this is not absolutely required.

2445 **R7.5-1:** A conformant service may support Delete.

2446 **R7.5-2:** A conformant service should support Delete using the same EPR as a corresponding
2447 Get or other messages, unless the deletion mechanism for a resource is semantically distinct.

2448 **R7.5-3:** If deletion is supported and the corresponding resource can be retrieved using Get, a
2449 conformant service should support deletion using Delete. The service may additionally export a
2450 custom action for deletion.

2451 **R7.5-4:** If an object cannot be deleted due to locking conditions, simultaneous access, or
2452 similar conflicts, a wsman:Concurrency fault should be returned.

2453 In practice, Delete removes the resource instance from the visibility of the client and is a *logical*
2454 deletion.

2455 The operation might result in an actual deletion, such as removal of a row from a database table, or it
 2456 might simulate deletion by unbinding the representation from the real-world object. Deletion of a
 2457 "printer," for example, does not result in literal annihilation of the printer, but simply removes it from
 2458 the access scope of the service, or "unbinds" it from naming tables. WS-Management makes no
 2459 distinction between literal deletions and logical deletions.

2460 To delete individual property values within an object that, itself, is not to be deleted, either the client
 2461 can perform a Put, according to section 7.4 or the service can support fragment-level delete (7.7).

2462 Fault usage is generally as described in clause 14. Inability to locate or access the resource is
 2463 equivalent to problems with the SOAP message when the EPR is defective. There are no "Delete-
 2464 specific" faults.

2465 7.6 Create

2466 A Web service operation (Create) is defined for creating a resource and providing its initial
 2467 representation. In some cases, the initial representation may constitute the representation of a logical
 2468 constructor for the resource and may thus differ structurally from the representation returned by Get
 2469 or the one required by Put. This difference is because the parameterization requirement for creating a
 2470 resource is often distinct from the steady-state representation of the resource. Implementations
 2471 should provide metadata that describes the use of the representation and how it relates to the
 2472 resource which is created, but such mechanisms are beyond the scope of this specification. The
 2473 resource factory that receives a Create request allocates a new resource that is initialized from the
 2474 presented representation. The new resource is assigned a service-determined endpoint reference
 2475 that is returned in the response message.

2476 The Create request message shall be of the following form:

```

2477 (1) <s:Envelope ...>
2478 (2)   <s:Header ...>
2479 (3)     <wsa:Action>
2480 (4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
2481 (5)     </wsa:Action>
2482 (6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
2483 (7)     <wsa:To>xs:anyURI</wsa:To>
2484 (8)     ...
2485 (9)   </s:Header>
2486 (10)  <s:Body ...>
2487 (11)    resource-specific-element
2488 (12)  </s:Body>
2489 (13) </s:Envelope>
  
```

2490 The following describes additional, normative constraints on the preceding outline:

2491 /s:Envelope/s:Header/wsa:Action

2492 This required element shall contain the value
 2493 http://schemas.xmlsoap.org/ws/2004/09/transfer/Create. If a SOAP Action URI is also present in
 2494 the underlying transport, its value shall convey the same value.

2495 /s:Envelope/s:Body/child

2496 The child element of the s:Body element shall not be omitted. The contents of this element are
 2497 service-specific, and may contain the literal initial resource representation, a representation of
 2498 the constructor for the resource, or other instructions for creating the resource.

2499 Extension specifications may also define extensions to the original Create request, enabled by
 2500 optional SOAP headers, which constrain the nature of the response (see information about the
 2501 CreateResponse later in this clause). Similarly, they may require headers that control the
 2502 interpretation of the s:Body as part of the resource creation process.

2503 Such specifications shall also allow processing the Create message without such extensions.

2504 A Create request shall be targeted at a resource factory capable of creating the desired new
2505 resource. This factory is distinct from the resource being created (which by definition does not exist
2506 prior to the successful processing of the Create request message).

2507 In addition to the standard fault codes defined in Addressing, implementations may use the fault code
2508 wsmt:InvalidRepresentation if the presented representation is invalid for the target resource.

2509 Other components of the preceding outline are not further constrained by this specification.

2510 If the resource factory accepts a Create request, it shall reply with a response of the following form:

```

2511 (1) <s:Envelope ...>
2512 (2)   <s:Header ...>
2513 (3)     <wsa:Action>
2514 (4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
2515 (5)     </wsa:Action>
2516 (6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
2517 (7)     <wsa:To>xs:anyURI</wsa:To>
2518 (8)   ...
2519 (9)   </s:Header>
2520 (10)  <s:Body ...>
2521 (11)    <wsmt:ResourceCreated>endpoint-reference</wsmt:ResourceCreated>
2522 (12)  </s:Body>
2523 (13) </s:Envelope>

```

2524 /s:Envelope/s:Header/wsa:Action

2525 This required element shall contain the value
2526 http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse. If a SOAP Action URI is also
2527 present in the underlying transport, its value shall convey the same value.

2528 /s:Envelope/s:Body/wsmt:ResourceCreated

2529 This required element shall contain a resource reference for the newly created resource. This
2530 resource reference, represented as an endpoint reference as defined in Addressing, shall
2531 identify the resource for future Get, Put, and Delete operations.

2532 Extension specifications may define extensions to the original Create request, enabled by optional
2533 header values. These headers may override the default behavior if they are marked with
2534 s:mustUnderstand="true". In the absence of such optional headers, the behavior shall be as
2535 described in the previous paragraphs. Because the response may not be sent to the original sender,
2536 extension specifications should consider adding a corresponding SOAP header value in the response
2537 to signal to the receiver that the extension is being used.

2538 Other components of the preceding outline are not further constrained by this specification.

2539 In general, the addressing is not the same as that used for Get or Delete in that the EPR assigned to
2540 a newly created instance for subsequent access is not necessarily part of the XML content used for
2541 creating the resource. Because the EPR is usually assigned by the service or one of its underlying
2542 systems, the CreateResponse contains the applicable EPR of the newly created instance.

2543 **R7.6-1:** A conformant service may support Create.

2544 **R7.6-2:** If a single resource can be created using a SOAP message and that resource can be
2545 subsequently retrieved using Get, then a service should support creation of the resource using
2546 Create. The service may additionally export a custom method for instance creation.

2547 **R7.6-3:** If the supplied SOAP Body does not have the correct content for the resource to be
2548 created, the service should return a wsmt:InvalidRepresentation fault and detail codes as follows:

- 2549 • if one or more values in the <s:Body> were not correct:
2550 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues`
- 2551 • if one or more values in the <s:Body> were missing:
2552 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues`
- 2553 • if the wrong XML schema namespace was used and is not recognized by the service:
2554 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace`

2555 **R7.6-4:** A service shall not use Create to modify the value of an existing representation (except
2556 as specified in 7.11). If the targeted object already exists, the service should return a
2557 `wsman:AlreadyExists` fault.

2558 The message body for Create is not required to use the same schema as that returned with a Get
2559 operation for the resource. Often, the values required to create a resource are different from those
2560 retrieved using a Get operation or those used for updates with a Put operation.

2561 If a service needs to support creation of individual values within a representation (fragment-level
2562 creation, array insertion, and so on), it can support fragment-level access (7.7).

2563 **R7.6-5:** The response to a Create message shall contain the new EPR of the created resource
2564 in the `ResourceCreated` element.

2565 **R7.6-6:** This rule intentionally left blank.

2566 **EXAMPLE:** The following is a hypothetical example of a response for a newly created virtual drive:

```

2567 (1) <s:Envelope
2568 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
2569 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
2570 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
2571 (5)   xmlns:wsmst="http://schemas.xmlsoap.org/ws/2004/09/transfer">
2572 (6)   <s:Header>
2573 (7)     ...
2574 (8)     <wsa:Action>
2575 (9)       http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
2576 (10)    </wsa:Action>
2577 (11)    ...
2578 (12)   </s:Header>
2579 (13)   <s:Body>
2580 (14)     <wsmst:ResourceCreated>
2581 (15)       <wsa:Address>
2582 (16)         http://1.2.3.4/wsman/
2583 (17)       </wsa:Address>
2584 (18)       <wsa:ReferenceParameters>
2585 (19)         <wsman:ResourceURI>
2586 (20)           http://example.org/2005/02/virtualDrive
2587 (21)         </wsman:ResourceURI>
2588 (22)         <wsman:SelectorSet>
2589 (23)           <wsman:Selector Name="ID"> F: </wsman:Selector>
2590 (24)         </wsman:SelectorSet>
2591 (25)       </wsa:ReferenceParameters>
2592 (26)     </wsmst:ResourceCreated>
2593 (27)   </s:Body>
2594 (28) </s:Envelope>

```

2595 This example assumes that the default addressing model is in use. The response contains a ResourceCreated
 2596 block (lines 14-26), which contains the new endpoint reference of the created resource, including its
 2597 ResourceURI and the SelectorSet. This address would be used to retrieve the resource in a subsequent Get
 2598 operation.

2599 The service might use a network address that is the same as the <wsa:To> address in the Create request.

2600 **R7.6-7:** The service may ignore any values in the initial representation that are considered
 2601 read-only from the point of view of the underlying real-world object.

2602 This rule allows Get, Put, and Create to share the same schema. Put also allows the service to ignore
 2603 read-only properties during an update.

2604 **R7.6-8:** If the success of an operation cannot be reported as described in this clause and
 2605 cannot be reversed, the service should return a wsman:EncodingLimit fault with the following
 2606 detail code:

2607 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess`

2608 7.7 Fragment-Level Access

2609 Because the resource access mechanism defined in this specification works with entire instances and
 2610 it can be inconvenient to specify hundreds or thousands of EPRs just to model fragment-level access
 2611 with full EPRs, WS-Management supports the concept of fragment-level (property) access of
 2612 resources that are normally accessed through the resource access operations. This access is done
 2613 through special use of these operations.

2614 Because of the XML schema limitations discussed in 7.6, simply returning a subset of the XML
 2615 defined for the object being accessed is often incorrect because a subset may violate the XML
 2616 schema for that fragment. To support resource access of fragments or individual elements of a
 2617 representation object, several modifications to the basic resource access operations are made.

2618 **R7.7-1:** A conformant service may support fragment-level access. If the service supports
 2619 fragment-level access, the service shall not behave as if the normal access operations were in
 2620 place but shall operate exclusively on the fragments specified. If the service does not support
 2621 fragment-level access, it shall return a wsman:UnsupportedFeature fault with the following detail
 2622 code:

2623 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAccess`

2624 **R7.7-2:** A conformant service that supports fragment-level access shall accept the following
 2625 SOAP header in all requests and include it in all responses that transport the fragments:

```
2626 (1) <wsman:FragmentTransfer s:mustUnderstand="true">
2627 (2)   xpath to fragment
2628 (3) </wsman:FragmentTransfer>
```

2629 The value of this header is the [XPath 1.0](#) expression that identifies the fragment being transferred
 2630 with relation to the full representation of the object. If an expression other than [XPath 1.0](#) is used,
 2631 a Dialect attribute can be added to indicate this, as follows:

```
2632 (4) <wsman:FragmentTransfer s:mustUnderstand="true"
2633 (5)   Dialect="URIToNewFragmentDialect">
2634 (6)   dialect expression
2635 (7) </wsman:FragmentTransfer>
```

2636 The client needs to understand that unless the header is marked mustUnderstand="true", the service
 2637 might process the request while ignoring the header, resulting in unexpected and potentially serious
 2638 side effects.

2639 XPath is explicitly defined as a dialect due to its importance, but it is not required that
 2640 implementations support XPath as a fragment dialect. Any other type of language to describe
 2641 fragment-level access is permitted as long as the Dialect value is set to indicate to the service what
 2642 dialect is being used.

2643 **R7.7-3:** For resource access fragment operations that use [XPath 1.0](#) (Dialect URI of
 2644 <http://www.w3.org/TR/1999/REC-xpath-19991116>), the value of the
 2645 `/s:Envelope/s:Header/wsman:FragmentTransfer` element is an XPath expression. This XPath
 2646 expression is evaluated using the following context:

- 2647 • **Context Node:** the root element of the XML representation of the resource addressed in
 2648 the request that would be returned as the initial child element of the SOAP Body response if
 2649 a Get operation was applied against the addressed resource without using fragment access
- 2650 • **Context Position:** 1
- 2651 • **Context Size:** 1
- 2652 • **Variable Bindings:** none
- 2653 • **Function Libraries:** Core Function Library [XPath 1.0](#)
- 2654 • **Namespace Declarations:** the [in-scope namespaces] property [XML Infoset](#) of the
 2655 request `/s:Envelope/s:Header/wsman:FragmentTransfer` element

2656 This rule means that the XPath is to be interpreted relative to the XML representation of the resource
 2657 and not relative to any of the SOAP content.

2658 For the Enumeration operations, the XPath is interpreted as defined in clause 8, although the output
 2659 is subsequently wrapped in `wsman:XmlFragment` wrappers after the XPath is evaluated.

2660 An XPath value can refer to the entire node, so the concept of a fragment includes the entire object,
 2661 making fragment-level access a proper superset of normal resource access operations.

2662 If the full XPath expression syntax cannot be supported, a common subset for this purpose is
 2663 described in ANNEX C of this specification. However, in such cases, the Dialect URI is still that of
 2664 XPath.

2665 **R7.7-4:** If a service understands fragment access but does not understand the specified
 2666 fragment Dialect URI or the default dialect, the service shall issue a
 2667 `wsman:FragmentDialectNotSupported` fault.

2668 **R7.7-5:** All resource access messages in either direction of the XML fragments shall be
 2669 wrapped with a `<wsman:XmlFragment>` wrapper that contains a definition that suppresses
 2670 validation and allows any content to pass. A service shall reject any attempt to use
 2671 `wsman:FragmentTransfer` unless the `s:Body` wraps the content using a `wsman:XmlFragment`
 2672 wrapper. If any other usage is encountered, the service shall fault the request by using a
 2673 `wsmt:InvalidRepresentation` fault with the following detail code:

2674 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFragment>

2675 Fragment access can occur at any level, including single element, complex elements, simple values,
 2676 and attributes. In practice, services typically support only value-level access to elements.

2677 **R7.7-6:** If fragment-level access is supported, a conformant service should support at least
 2678 leaf-node, value-level access using an XPath expression that uses the `/text()` NodeTest. In this
 2679 case, the value is not wrapped with XML but is transferred directly as text within the
 2680 `wsman:XmlFragment` wrapper.

2681 In essence, the transferred content is whatever an XPath operation over the full XML would produce.

2682 **R7.7-7:** If fragment-level access is supported but the filter expression exceeds the capability of
 2683 the service, the service should return a wsman:CannotProcessFilter fault with text explaining why
 2684 the filter was problematic.

2685 **R7.7-8:** For all fragment-level operations, partial successes are not permitted. The entire
 2686 meaning of the XPath expression or other dialect shall be fully observed by the service in all
 2687 operations, and the entire fragment that is specified shall be successfully transferred in either
 2688 direction. Otherwise, faults occur as if none of the operation had succeeded.

2689 All faults are the same as for normal, "full" resource access operations.

2690 The following clauses show how the underlying resource access operations change when transferring
 2691 XML fragments.

2692 7.8 Fragment-Level Get

2693 Fragment-level Get is similar to full Get, except for the wsman:FragmentTransfer header (lines 25-
 2694 27).

2695 EXAMPLE 1: The following example is drawn from the example in 7.1:

```

2696 (1) <s:Envelope
2697 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
2698 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
2699 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
2700 (5)   <s:Header>
2701 (6)     <wsa:To>
2702 (7)       http://1.2.3.4/wsman
2703 (8)     </wsa:To>
2704 (9)     <wsman:ResourceURI>http://example.org/2005/02/physicalDisk
2705 (10)    </wsman:ResourceURI>
2706 (11)    <wsa:ReplyTo>
2707 (12)      <wsa:Address>
2708 (13)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
2709 (14)      </wsa:Address>
2710 (15)    </wsa:ReplyTo>
2711 (16)    <wsa:Action>
2712 (17)      http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
2713 (18)    </wsa:Action>
2714 (19)    <wsa:MessageID>
2715 (20)      urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
2716 (21)    </wsa:MessageID>
2717 (22)    <wsman:SelectorSet>
2718 (23)      <wsman:Selector Name="LUN"> 2 </wsman:Selector>
2719 (24)    </wsman:SelectorSet>
2720 (25)    <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
2721 (26)    <wsman:FragmentTransfer s:mustUnderstand="true">
2722 (27)      Manufacturer
2723 (28)    </wsman:FragmentTransfer>
2724 (29)  </s:Header>
2725 (30)  <s:Body/>
2726 (31) </s:Envelope>
  
```

2727 In this case, the service executes the specified XPath expression against the representation that
 2728 would normally have been retrieved, and then return a fragment instead.

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

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

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

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

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

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

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

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

7.9 Fragment-Level Put

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

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

2777 EXAMPLE 1: Consider the following XML for illustrative purposes:

```

2778 (1) <a>
2779 (2)   <b>
2780 (3)     <c> </c>
2781 (4)     <d> </d>
2782 (5)   </b>
2783 (6)   <e>
2784 (7)     <f> </f>
2785 (8)     <g> </g>
2786 (9)   </e>
2787 (10) </a>

```

2788 Although <a> is the entire representation of the resource instance, if the operation references the a/b
 2789 node during the Put operation, using an XPath expression of "b", then the content of is updated
 2790 without touching other parts of <a>, such as <e>. If the client wants to update only <d>, then the
 2791 XPath expression used is "b/d".

2792 EXAMPLE 2: Continuing from the example in SECTION 7.1, if the client wanted to update the <BootPartition>
 2793 value from 0 to 1, the following Put fragment could be sent to the service:

```

2794 (1) <s:Envelope
2795 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
2796 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
2797 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
2798 (5) <s:Header>
2799 (6)   <wsa:To>
2800 (7)     http://1.2.3.4/wsman
2801 (8)   </wsa:To>
2802 (9)   <wsman:ResourceURI>http://example.org/2005/02/physicalDisk
2803 (10)  </wsman:ResourceURI>
2804 (11)  <wsa:ReplyTo>
2805 (12)    <wsa:Address>
2806 (13)      http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
2807 (14)    </wsa:Address>
2808 (15)  </wsa:ReplyTo>
2809 (16)  <wsa:Action>
2810 (17)    http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
2811 (18)  </wsa:Action>
2812 (19)  <wsa:MessageID>
2813 (20)    urn:uuid:d9726315-bc91-2222-9ed8-c044c9658a87
2814 (21)  </wsa:MessageID>
2815 (22)  <wsman:SelectorSet>
2816 (23)    <wsman:Selector Name="LUN"> 2 </wsman:Selector>
2817 (24)  </wsman:SelectorSet>
2818 (25)  <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
2819 (26)  <wsman:FragmentTransfer s:mustUnderstand="true">
2820 (27)    BootPartition
2821 (28)  </wsman:FragmentTransfer>
2822 (29) </s:Header>
2823 (30) <s:Body>
2824 (31)   <wsman:XmlFragment>
2825 (32)     <BootPartition> 1 </BootPartition>
2826 (33)   </wsman:XmlFragment>
2827 (34) </s:Body>
2828 </s:Envelope>

```

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

```
(35) <s:Header>
(36)   ...
(37)   <wsman:FragmentTransfer s:mustUnderstand="true">
(38)     BootPartition/text()
(39)   </wsman:FragmentTransfer>
(40) </s:Header>
(41) <s:Body>
(42)   <wsman:XmlFragment>
(43)     1
(44)   </wsman:XmlFragment>
(45) </s:Body>
```

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

EXAMPLE 4: The following is a sample reply:

```
(46) <s:Envelope
(47)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(48)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(49)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
(50)   <s:Header>
(51)     <wsa:To>
(52)       http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(53)     </wsa:To>
(54)     <wsa:Action s:mustUnderstand="true">
(55)       http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse
(56)     </wsa:Action>
(57)     <wsa:MessageID s:mustUnderstand="true">
(58)       urn:uuid:ee7f13b5-0091-430b-9ed8-2e12fbaa8a7e
(59)     </wsa:MessageID>
(60)     <wsa:RelatesTo>
(61)       urn:uuid:d9726315-bc91-2222-9ed8-c044c9658a87
(62)     </wsa:RelatesTo>
(63)     <wsman:FragmentTransfer s:mustUnderstand="true">
(64)       BootPartition/text()
(65)     </wsman:FragmentTransfer>
(66)   </s:Header>
(67)   <s:Body>
(68)     <wsman:XmlFragment>
(69)       1
(70)     </wsman:XmlFragment>
(71)   </s:Body>
(72) </s:Envelope>
```

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

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

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

2880 NOTE: The fragment-level Put operation implies replacement or update and does not insert new values into the
 2881 representation object. Thus, it is not appropriate to use Put to insert a new value at the end of an array, for
 2882 example. The entire array can be returned and then updated and replaced (because it is therefore an update of
 2883 the entire array), but a single operation to insert a new element in the middle or at the end of an array is actually
 2884 a Create operation.

2885 As stated in 7.4, if the new representation differs from the input, the new representation is to be
 2886 returned in the response. With fragment-level Put, this rule applies only to the portion of the
 2887 representation object being written, not the entire object. If a single value is written and accepted, but
 2888 has side effects on other values in the representation, the entire object is *not* returned.

2889 To set a value to NULL without removing it as an element, use an attribute value of xsi:nil on the
 2890 element being set to NULL to ensure that the fragment path is adjusted appropriately.

2891 EXAMPLE 5:

```

2892 (73) <s:Header> ...
2893 (74) <wsman:FragmentTransfer s:mustUnderstand="true">
2894 (75) AssetLabel
2895 (76) </wsman:FragmentTransfer>
2896 (77) ...
2897 (78) </Header>
2898 (79) <s:Body>
2899 (80) <wsman:XmlFragment xmlns:xsi="www.w3.org/2001/XMLSchema-instance">
2900 (81) <AssetLabel xsi:nil="true"/>
2901 (82) </wsman:XmlFragment>
2902 (83) </s:Body>
  
```

2903 7.10 Fragment-Level Delete

2904 Fragment-level Delete applies only if the XML schema for the targeted object supports optional
 2905 elements that can be removed from the representation object, or supports arrays (repeated elements)
 2906 with varying numbers of elements and the client wants to remove an element in an array. If
 2907 replacement of an entire array is needed, fragment-level Put can be used. For array access, the
 2908 XPath array access notation can conveniently be used. To delete a value that is legal to remove
 2909 (according to the rules of the schema for the object), the wsman:FragmentTransfer expression
 2910 identifies the item to be removed.

2911 EXAMPLE 1:

```

2912 (1) <wsman:FragmentTransfer s:mustUnderstand="true">
2913 (2) VolumeLabel
2914 (3) </wsman:FragmentTransfer>
  
```

2915 To set a value to NULL without removing it as an element, use fragment-level Put with a value of
 2916 xsi:nil.

2917 To delete an array element, use the XPath [] operators.

2918 EXAMPLE 2: The following example deletes the second <BlockedIPAddress> element in the representation.
 2919 (XPath arrays are 1 based.)

```

2920 (1) <wsman:FragmentTransfer s:mustUnderstand="true">
2921 (2) BlockedIPAddress[2]
2922 (3) </wsman:FragmentTransfer>
  
```

2923 The <s:Body> is empty for all Delete operations, even with fragment-level access, and all normal
 2924 faults for Delete apply.

2925 **R7.10-1:** If a value cannot be deleted because of locking conditions or similar phenomena, the
 2926 service should return a wsman:AccessDenied fault.

7.11 Fragment-Level Create

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

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

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

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

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

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

```
(4) <s:Body>
(5)   <wsman:XmlFragment>
(6)     <VolumeLabel> MyDisk </VolumeLabel>
(7)   </wsman:XmlFragment>
(8) </s:Body>
```

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

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

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

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

```
(12) <s:Body>
(13)   <wsman:XmlFragment>
(14)     MyDisk
(15)   </wsman:XmlFragment>
(16) </s:Body>
```

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

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

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

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

2967 The body of Create contains the value to be inserted and is the same as for fragment-level Put.

2968 EXAMPLE 6:

```

2969 (20) <s:Body>
2970 (21)   <wsman:XmlFragment>
2971 (22)     <BlockedIPAddress> 123.12.188.44 </BlockedIPAddress>
2972 (23)   </wsman:XmlFragment>
2973 (24) </s:Body>

```

2974 This operation adds a third IP address to the <BlockedIPAddress> array (a repeated element),
 2975 assuming that at least two elements are at that level already.

2976 **R7.11-1:** A service shall not use fragment-level Create to modify the value of an existing
 2977 property. If the targeted object and the targeted property already exists, the service should return
 2978 a wsman:AlreadyExists fault.

2979 **R7.11-2:** If the Create fails because the result would not conform to the schema in some way,
 2980 the service should return a wsmt:InvalidRepresentation fault.

2981 As defined in 7.6, the CreateResponse contains the EPR of the created resource. In the case of
 2982 fragment-level Create, the response additionally contains the wsman:FragmentTransfer block,
 2983 including the path (line 12), in a SOAP header.

2984 EXAMPLE 7: In the following example, the ResourceCreated EPR continues to refer to the entire object, not just
 2985 to the fragment.

```

2986 (25) <s:Envelope
2987 (26)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
2988 (27)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
2989 (28)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
2990 (29)   xmlns:wsmt="http://schemas.xmlsoap.org/ws/2004/09/transfer">
2991 (30) <s:Header>
2992 (31)   ...
2993 (32)   <wsa:Action>
2994 (33)     http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
2995 (34)   </wsa:Action>
2996 (35)   <wsman:FragmentTransfer s:mustUnderstand="true">
2997 (36)     Path To Fragment
2998 (37)   </wsman:FragmentTransfer>
2999 (38)   ...
3000 (39) </s:Header>
3001 (40) <s:Body>
3002 (41)   <wsmt:ResourceCreated>
3003 (42)     <wsa:Address> ... </wsa:Address>
3004 (43)     <wsa:ReferenceParameters>
3005 (44)       <wsman:SelectorSet>
3006 (45)         <wsman:Selector ...> ... </wsman:Selector>
3007 (46)       </wsman:SelectorSet>
3008 (47)     </wsa:ReferenceParameters>
3009 (48)   </wsmt:ResourceCreated>
3010 (49) </s:Body>
3011 (50) </s:Envelope>

```

3012 As discussed in 7.6, to remain compatible with WSDL, only the EPR of the item is returned in the
 3013 SOAP Body, in spite of other options discussed in 7.6.

8 Enumeration of Datasets

8.1 General

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

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

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

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

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

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

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

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

Enumeration contexts are represented as XML data that is opaque to the consumer. Initially, the consumer gets an enumeration context from the data source by means of an Enumerate operation. The consumer then passes that XML data back to the data source in the Pull request. Optionally, the data source may return an updated enumeration context in the Pull response; when present, this new

- 3061 enumeration context should replace the old one on the consumer, and it should be passed to the data
 3062 source in all future responses until and unless the data source again returns an updated enumeration
 3063 context.
- 3064 Consumers should not reuse old enumeration contexts that have been replaced by the data source.
 3065 Using a replaced enumeration context in a Pull response may yield undefined results, including being
 3066 ignored or generating a fault.
- 3067 After the last element in a sequence has been returned, or the enumeration context has expired, the
 3068 enumeration context is considered invalid and the result of subsequent operations referencing that
 3069 context is undefined.
- 3070 Callers may issue a Release operation against a valid enumeration context at any time, which causes
 3071 the enumeration context to become invalid and allows the data source to free up any resources it may
 3072 have allocated to the enumeration. Issuing a Release operation prior to reaching the end of the
 3073 sequence of elements is explicitly allowed; however, no further operations should be issued after a
 3074 Release.
- 3075 In addition, the data source may invalidate an enumeration context at any time, as necessary.
- 3076 If a resource with multiple instances provides a mechanism for enumerating or querying the set of
 3077 instances, the operations defined in this clause can be used to perform the iteration.
- 3078 **R8.1-1:** A service may support the Enumeration operations if enumeration of any kind is
 3079 supported.
- 3080 **R8.1-2:** If simple, unfiltered enumeration of resource instances is exposed through Web
 3081 services, a conformant service shall support the Enumeration operations to expose this. The
 3082 service may also support other techniques for enumerating the instances.
- 3083 **R8.1-3:** If filtered enumeration (queries) of resource instances is exposed through Web
 3084 services, a conformant service should support the Enumeration operations to expose this. The
 3085 service may also support other techniques for enumerating the instances.
- 3086 This clause indicates that enumeration is a three-part operation:
- 3087 1) An initial Enumerate message is issued to establish the enumeration context.
 3088 2) Pull operations are used to iterate over the result set.
 3089 3) When the enumeration iterator is no longer required and not yet exhausted, a Release
 3090 message is issued to release the enumerator and associated resources.
- 3091 As with other WS-Management methods, the enumeration can make use of wsman:OptionSet.
- 3092 **R8.1-4:** A service may implement wsmen:Renew, wsmen:GetStatus and
 3093 wsmen:EnumerationEnd messages; however, in constrained environments these are candidates
 3094 for exclusion. If these messages are not supported, then a wsa:ActionNotSupported fault shall be
 3095 returned in response to these requests.
- 3096 **R8.1-5:** If a service is exposing enumeration, it shall at least support the following messages:
 3097 Enumerate, Pull, and Release, and their associated responses.
- 3098 If the service does not support stateful enumerators, the Release is a simple no-op, so it is trivial to
 3099 implement. (It always succeeds when the operation is valid.) However, it is supported to allow for the
 3100 uniform construction of clients.
- 3101 **R8.1-6:** The Pull and Release operations are a continuation of the original Enumerate
 3102 operation. The service should enforce the same authentication and authorization throughout the

3103 entire sequence of operations and should fault any attempt to change credentials during the
3104 sequence.

3105 Some transports such as HTTP might drop or reestablish connections between Enumerate and
3106 subsequent Pull operations, or between Pull operations. It is expected that services will allow the
3107 enumeration to continue uninterrupted, but for practical reasons some services might require that the
3108 same connection be used. This specification establishes no requirements in this regard. However,
3109 R8.1-6 establishes that the user credentials do not change during the entire enumeration sequence.

3110 8.2 Enumerate

3111 All data sources shall support some operation that allows an enumeration to be started. A data
3112 source may support the Enumerate operation, or it may provide some other mechanism for starting
3113 an enumeration and receiving an enumeration context.

3114 The Enumerate operation is initiated by sending an Enumerate request message to the data source.
3115 The Enumerate request message shall be of the following form:

```
3116 (1) <s:Envelope ...>
3117 (2)   <s:Header ...>
3118 (3)     <wsa:Action>
3119 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
3120 (5)     </wsa:Action>
3121 (6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
3122 (7)     <wsa:To>xs:anyURI</wsa:To>
3123 (8)   ...
3124 (9) </s:Header>
3125 (10) <s:Body ...>
3126 (11)   <wsmen:Enumerate ...>
3127 (12)     <wsmen:EndTo>endpoint-reference</wsmen:EndTo> ?
3128 (13)     <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
3129 (14)     <wsmen:Filter Dialect="xs:anyURI"?> xs:any </wsmen:Filter> ?
3130 (15)   ...
3131 (16)   </wsmen:Enumerate>
3132 (17) </s:Body>
3133 (18) </s:Envelope>
```

3134 The following describes additional, normative constraints on the preceding outline:

3135 /s:Envelope/s:Header/wsa:Action

3136 This required element shall contain the value:

3137 http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate.

3138 If a SOAP Action URI is also present in the underlying transport, its value shall convey the same
3139 value.

3140 /s:Envelope/s:Body/*wsmen:EndTo

3141 This optional element denotes where to send an EnumerationEnd message if the enumeration is
3142 terminated unexpectedly. If present, this element shall be of type wsa:EndpointReferenceType.
3143 The default is to not send this message. The endpoint referenced by this EPR shall implement a
3144 binding of the "EnumEndEndpoint" portType described in ANNEX H.

3145 /s:Envelope/s:Body/*wsmen:Expires

3146 Requested expiration time for the enumeration. (No implied value.) The data source defines the
3147 actual expiration and is not constrained to use a time less or greater than the requested
3148 expiration. The expiration time may be a specific time or a duration from the enumeration's
3149 creation time. Both specific times and durations are interpreted based on the data source's clock.

3150 If this element does not appear, then the request is for an enumeration that will not expire. That
 3151 is, the consumer is requesting the data source to create an enumeration with an indefinite
 3152 lifetime. If the data source grants such an enumeration, it will terminate when the end of the
 3153 enumeration is reached, or if the consumer sends a Release request, or by the data source at
 3154 any time for reasons such as connection termination, resource constraints, or system shut-down.
 3155 If the expiration time is either a zero duration or a specific time that occurs in the past according
 3156 to the data source, then the request shall fail, and the data source may generate a
 3157 `wsmen:InvalidExpirationTime` fault indicating that an invalid expiration time was requested.
 3158 Some data sources may not have a "wall time" clock available, and so are able only to accept
 3159 durations as expirations. If such a source receives an Enumerate request containing a specific
 3160 time expiration, then the request shall fail; if so, the data source should generate a
 3161 `wsmen:UnsupportedExpirationType` fault indicating that an unsupported expiration type was
 3162 requested.

3163 `/s:Envelope/s:Body/wsmen:Enumerate/wsmen:Filter`

3164 This optional element contains a Boolean predicate in some dialect (see
 3165 `/s:Envelope/s:Body/*/wsmen:Filter/@Dialect`) that all elements of interest must satisfy. The
 3166 resultant enumeration context shall not return elements for which this predicate expression
 3167 evaluates to the value false. If this element is absent, then the implied value is the expression
 3168 `true()`, indicating that no filtering is desired.

3169 If the data source does not support filtering, the request shall fail, and the data source may
 3170 generate a `wsmen:FilteringNotSupported` SOAP fault as follows:

3171 If the data source supports filtering but cannot honor the requested filter dialect, the request shall
 3172 fail, and the data source may generate a `wsmen:FilterDialectRequestedUnavailable` SOAP fault
 3173 as follows:

3174 If the data source supports filtering and the requested dialect but cannot process the requested
 3175 filter content, the request shall fail, and the data source may generate a
 3176 `wsmen:CannotProcessFilter` SOAP fault as follows:

3177 `/s:Envelope/s:Body/*/wsmen:Filter/@Dialect`

3178 Implied value is "`http://www.w3.org/TR/1999/REC-xpath-19991116`".

3179 `/s:Envelope/s:Body/*/wsmen:Filter/@Dialect= "http://www.w3.org/TR/1999/REC-xpath-19991116"`

3180 Value of `/s:Envelope/s:Body/*/wsmen:Filter` is an XPath [XPath 1.0] predicate expression
 3181 (PredicateExpr); the context of the expression is:

- 3182 • **Context Node:** any XML element that could be returned as a direct child of the Items
 3183 element
- 3184 • **Context Position:** 1
- 3185 • **Context Size:** 1
- 3186 • **Variable Bindings:** None
- 3187 • **Function Libraries:** Core Function Library [XPath 1.0]
- 3188 • **Namespace Declarations:** The [in-scope namespaces] property [XML Infoset] of
 3189 `/s:Envelope/s:Body/*/wsmen:Filter`

3190 Other components of the preceding outline are not further constrained by this specification.

3191 Upon successful processing of an Enumerate request message, a data source is expected to create
 3192 an enumeration context and return that context in an Enumerate response message, which shall
 3193 adhere to the following form:

```

3194 (1) <s:Envelope ...>
3195 (2)   <s:Header ...>
3196 (3)     <wsa:Action>
```

```

3197 (4) http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse
3198 (5) </wsa:Action>
3199 (6) <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
3200 (7) <wsa:To>xs:anyURI</wsa:To>
3201 (8) ...
3202 (9) </s:Header>
3203 (10) <s:Body ...>
3204 (11) <wsmen:EnumerateResponse ...>
3205 (12) <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
3206 (13) <wsmen:EnumerationContext>...</wsmen:EnumerationContext>
3207 (14) ...
3208 (15) </wsmen:EnumerateResponse>
3209 (16) </s:Body>
3210 (17) </s:Envelope>

```

3211 The following describes additional, normative constraints on the preceding outline:

3212 /s:Envelope/s:Header/wsa:Action

3213 This required element shall contain the value:

3214 http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse

3215 If a SOAP Action URI is also present in the underlying transport, its value shall convey the same
3216 value.

3217 /s:Envelope/s:Body/*wsmen:Expires

3218 The expiration time assigned by the data source. The expiration time may be either an absolute
3219 time or a duration but should be of the same type as the requested expiration (if any).

3220 If this element does not appear, then the enumeration will not expire. That is, the enumeration
3221 has an indefinite lifetime. It will terminate when the end of the enumeration is reached, if the
3222 consumer sends a Release request, or by the data source at any time for reasons such as
3223 connection termination, resource constraints, or system shut-down.

3224 /s:Envelope/s:Body/wsmen:EnumerateResponse/wsmen:EnumerationContext

3225 The required EnumerationContext element contains the XML representation of the new
3226 enumeration context. The consumer is required to pass this XML data in Pull requests for this
3227 enumeration context, until and unless a PullResponse message updates the enumeration
3228 context.

3229 8.2.1 General

3230 WS-Management qualifies the Enumerate operation as described in this clause.

3231 **R8.2.1-1:** A conformant service may accept a wsmen:Enumerate message with an EndTo
3232 address; however, if EnumerationEnd is not supported, a service may instead issue a
3233 wsman:UnsupportedFeature fault with the following detail code:

3234 http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode

3235 **R8.2.1-2:** A conformant service shall accept an Enumerate message with an Expires timeout
3236 or fault with wsman:UnsupportedFeature and the following detail code:

3237 http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime

3238 **R8.2.1-3:** The wsman:Filter element (see 8.3) in the Enumerate body shall be either simple
3239 text or a single complex XML element. A conformant service shall not accept mixed content of
3240 both text and elements, or multiple peer XML elements under the wsman:Filter element.

3241 Although this use of mixed content is allowed in the general case of Enumerate, it is unnecessarily
3242 complex for WS-Management implementations.

3243 A common filter dialect is [XPath 1.0](#) (identified by the Dialect URI [http://www.w3.org/TR/1999/REC-](http://www.w3.org/TR/1999/REC-xpath-19991116)
3244 [xpath-19991116](#)). Resource-constrained implementations might have difficulty exporting full XPath
3245 processing and yet still want to use a subset of XPath syntax. As long as the filter expression is a
3246 proper subset of the specified dialect, it is legal and can be described using that Dialect value.

3247 No rule mandates the use of XPath or any subset as a filtering dialect. If no Dialect is specified, the
3248 default interpretation is that the Filter value is XPath (as specified previously in this clause).

3249 **R8.2.1-4:** A conformant service may not support the entire syntax and processing power of
3250 the specified Filter Dialect. The only requirement is that the specified Filter is syntactically correct
3251 within the definition of the Dialect. Subsets are therefore legal. If the specified Filter exceeds the
3252 capability of the service, the service should return a wsman:CannotProcessFilter fault with some
3253 text indicating what went wrong.

3254 Some services require filters to function because their search space is so large that simple
3255 enumeration is meaningless or impossible.

3256 **R8.2.1-5:** If a wsman:Filter is required, a conformant service shall fault any request without a
3257 wsman:Filter, by using a wsman:UnsupportedFeature fault with the following detail code:

3258 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired>

3259 **R8.2.1-6:** A conformant service may block, fault (using wsman:Concurrency faults), or allow
3260 other concurrent operations on the resource for the duration of the enumeration, and may include
3261 or exclude the results of such operations as part of any enumeration still in progress.

3262 If clients execute other operations, such as Create or Delete, while an enumeration is occurring, this
3263 specification makes no restrictions on the behavior of the enumeration. The service can include or
3264 exclude the results of these operations in real time, can produce an initial snapshot of the
3265 enumeration and execute the Pull requests from this snapshot, or can deny access to other
3266 operations while enumerations are in progress.

3267 8.2.2 Enumeration "Count" Option

3268 To give clients an estimate of the number of items in an enumeration, two optional SOAP headers are
3269 defined: one for use in the request message to return an approximate count of items in an
3270 enumeration sequence, and a corresponding header for use in the response to return this value to the
3271 client.

3272 These SOAP headers are defined for use with the Enumerate and Pull messages and their
3273 responses. The header used in Enumerate and Pull is as follows:

```
3274 (1) <s:Header>
3275 (2)   ...
3276 (3)   <wsman:RequestTotalItemsCountEstimate .../>
3277 (4) </s:Header>
```

3278 The header used by the service to return the value is as follows:

```
3279 (5) <s:Header>
3280 (6)   ...
3281 (7)   <wsman:TotalItemsCountEstimate>
3282 (8)     xs:nonNegativeInteger
3283 (9)   </wsman:TotalItemsCountEstimate>
3284 (10) </s:Header>
```

3285 The following definitions provide additional, normative constraints on the preceding headers:

3286 wsman:RequestTotalItemsCountEstimate

3287 when present as a SOAP header on an Enumerate or Pull message, indicates that the client is
3288 requesting that the associated response message includes an estimate of the total number of
3289 items in the enumeration sequence

3290 This SOAP header does not have any meaning defined by this specification when included with
3291 any other messages.

3292 wsman:TotalItemsCountEstimate

3293 when present as a SOAP header on an EnumerateResponse or PullResponse message,
3294 indicates the approximate number of items in the enumeration sequence

3295 This is the total number of items and not the remaining number of items in the sequence. This
3296 SOAP header does not have any meaning defined by this specification when included with any
3297 other messages.

3298 When a service understands the TotalItemsCountEstimate feature but cannot determine the
3299 number of items, the service responds with the wsman:TotalItemsCountEstimate element having
3300 an xsi:nil attribute with value 'true', and having no value, as follows:

3301 (1) `<wsman:TotalItemsCountEstimate xsi:nil="true"/>`

3302 **R8.2.2-1:** A conformant service may support the ability to return an estimate of the number of
3303 items in an enumeration sequence. If a service receives an Enumerate or Pull message without
3304 the wsman:RequestTotalItemsCountEstimate SOAP header, the service shall not return the
3305 wsman:TotalItemsCountEstimate SOAP header on the associated response message.

3306 **R8.2.2-2:** The value returned in the wsman:TotalItemsCountEstimate SOAP header is only an
3307 estimate of the number of items in the sequence. The client should not use the
3308 wsman:TotalItemsCountEstimate value for determining an end of enumeration instead of using
3309 EndOfSequence.

3310 This mechanism is intended to assist clients in determining the percentage of completion of an
3311 enumeration as it progresses. When a service sends a result count estimate after a previous estimate
3312 for the same enumeration sequence, the most recent total results count estimate is considered to be
3313 the more precise estimate.

3314 8.2.3 Optimization for Enumerations with Small Result Sets

3315 To optimize the number of round-trip messages required to enumerate the items in an enumerable
3316 resource, a client can request optimized enumeration behavior. This behavior is useful in cases
3317 where the enumeration has such a small number of items that the initial EnumerateResponse could
3318 reasonably include the entire result, without the need for a subsequent Pull to retrieve the items. This
3319 mechanism can be used even for large enumerations to get the first few results in the initial response.

3320 A client initiates an optimized enumeration by placing the wsman:OptimizeEnumeration element as a
3321 child element of the Enumerate element, and can optionally include the wsman:MaxElements
3322 element, as follows:

3323 EXAMPLE:

3324 (1) `<s:Body>`
3325 (2) `<wsman:Enumerate>`
3326 (3) `...`
3327 (4) `<wsman:OptimizeEnumeration/>`
3328 (5) `<wsman:MaxElements>xsi:positiveInteger</wsman:MaxElements> ?`

```

3329 (6)    </wsmen:Enumerate>
3330 (7)    </s:Body>

```

3331 The following definitions provide additional, normative constraints on the preceding outline:

3332 wsmen:Enumerate/wsman:OptimizeEnumeration

3333 when present as a child of the Enumerate element, indicates that the client is requesting an
 3334 optimized enumeration

3335 wsmen:Enumerate/wsman:MaxElements

3336 (optional) indicates the maximum number of items the consumer is willing to accept in the
 3337 EnumerateResponse

3338 It plays the same role as wsman:Pull/wsman:MaxElements. When this element is absent, its
 3339 implied value is 1.

3340 **R8.2.3-1:** A conformant service may support enumeration optimization. If a service receives
 3341 the wsman:OptimizeEnumeration element in an Enumerate message and it does not support
 3342 enumeration optimization, it should ignore the element and complete the enumeration request as
 3343 if the element were not present.

3344 If the service ignores the element, the client continues with a subsequent Pull as if the option was not
 3345 in force. The client requires no special mechanisms over what was needed for normal enumeration if
 3346 the optimization request is ignored.

3347 **R8.2.3-2:** A conformant service that receives an Enumerate message without the
 3348 wsman:OptimizeEnumeration element shall not return any enumeration items in the
 3349 EnumerateResponse message and shall return a EnumerationContext initialized to return the first
 3350 items when the first Pull message is received.

3351 If the service implements the optimization even if it was not requested, clients unaware of the
 3352 optimization will incorrectly process the enumeration result.

3353 **R8.2.3-3:** A conformant service that receives an Enumerate message with the
 3354 wsman:OptimizeEnumeration element shall not return more elements in the Enumerate response
 3355 message than requested in the wsman:MaxElements element (or no more than 1 item if the
 3356 wsman:MaxElements element is not present). Implementations may return fewer items based on
 3357 either the wsman:OperationTimeout SOAP header, wsman:MaxEnvelopeSize SOAP header, or
 3358 implementation-specific constraints.

3359 When requested by the client, a service implementing the optimized enumeration will respond with
 3360 the following additional content in an EnumerateResponse message:

```

3361 (1)    <s:Body>
3362 (2)    <wsmen:EnumerateResponse>
3363 (3)    <wsmen:EnumerationContext> ... </wsmen:EnumerationContext>
3364 (4)    <wsman:Items>
3365 (5)    ...same as for wsman:Items in wsman:PullResponse
3366 (6)    </wsman:Items> ?
3367 (7)    <wsman:EndOfSequence/> ?
3368 (8)    ...
3369 (9)    </wsmen:EnumerateResponse>
3370 (10)   </s:Body>

```

3371 The following definitions provide additional, normative constraints on the preceding outline:

3372 `wsman:Items`

3373 (optional) contains one or more enumeration-specific elements as would have been encoded for
3374 `Items` in a `PullResponse`

3375 The service will return no more than `wsman:MaxElements` elements in this list if
3376 `wsman:MaxElements` is specified in the request message, or one element if
3377 `wsman:MaxElements` was omitted.

3378 `wsman:EndOfSequence`

3379 (optional) indicates that no more elements are available from this enumeration and that the
3380 entire result (even if there are zero elements) is contained within the `wsman:Items` element

3381 `wsmen:EnumerationContext`

3382 required context for requesting additional items, if any, in subsequent Pull messages

3383 If the `wsman:EndOfSequence` is also present, the `EnumerationContext` cannot be used in a
3384 subsequent Pull request. The service should observe the same fault usage that would occur if
3385 the `EnumerationContext` were used in a Pull request after the `EndOfSequence` element occurred
3386 in a `PullResponse`. Although the `EnumerationContext` element must be present, no value is
3387 required; therefore, in cases where the `wsman:EndOfSequence` element is present, the value for
3388 `EnumerationContext` can be empty.

3389 EXAMPLE:

```
3390 (1) <s:Body>
3391 (2)   <wsmen:EnumerateResponse>
3392 (3)     <wsmen:EnumerationContext/>
3393 (4)     <wsman:Items>
3394 (5)       Items
3395 (6)     </wsman:Items>
3396 (7)     <wsman:EndOfSequence/>
3397 (8)     ...
3398 (9)   </wsmen:EnumerateResponse>
3399 (10) </s:Body>
```

3400 **R8.2.3-4:** A conformant service that supports optimized enumeration and is responding with
3401 an `EnumerateResponse` message shall include the `wsman:Items` element, the
3402 `wsman:EndOfSequence` element, or both in the response as an indication to the client that the
3403 optimized enumeration request was understood and honored.

3404 If neither `wsman:Items` nor `wsman:EndOfSequence` is in the `EnumerateResponse` message, the
3405 client can continue to use the enumeration message exchanges as defined in 8.2.1.

3406 **R8.2.3-5:** A conformant service that supports optimized enumeration and has not returned all
3407 items of the enumeration sequence in the `EnumerateResponse` message shall return an
3408 `EnumerationContext` element that is initialized such that a subsequent Pull message will return
3409 the set of items after those returned in the `EnumerateResponse`. If all items of the enumeration
3410 sequence have been returned in the `EnumerateResponse` message, the service should return an
3411 empty `EnumerationContext` element and shall return the `wsman:EndOfSequence` element in the
3412 response.

3413 A client that has requested optimized enumeration can determine if this request was understood and
3414 honored by the service by examining the response message.

3415 Clients concerned about the size of the initial response, irrespective of the number of items, can use
 3416 the wsman:MaxEnvelopeSize mechanism described in 6.2.

3417 **8.3 Filter Interpretation**

3418 The Filter expression is constrained to be a Boolean predicate. To support ad hoc queries including
 3419 projections, WS-Management defines a wsman:Filter element of exactly the same form as in the
 3420 Enumeration filter except that the filter expression is not constrained to be a Boolean predicate. This
 3421 allows the use of enumeration using existing query languages such as SQL and CQL, which combine
 3422 predicate and projection information in the same syntax. The use of projections is defined by the filter
 3423 dialect, not by WS-Management.

3424 (1) `<wsman:Filter Dialect="xs:anyURI"?> xs:any </wsman:Filter>`

3425 The Dialect attribute is optional. When not specified, it has the following implied value:

3426 `http://www.w3.org/TR/1999/REC-xpath-19991116`

3427 This dialect allows any full XPath expression or subset to be used.

3428 The wsman:Filter element is a child of the Enumerate element.

3429 If the filter dialect used for the Enumerate message is XPath 1.0, the context node is the same as that
 3430 specified in 8.1.

3431 **R8.3-1:** If a service supports filtered enumeration using Filter, it shall also support filtering using
 3432 wsman:Filter. This rule allows client stacks to always pick the wsman XML namespace for the
 3433 Filter element. Even though a service supports wsman:Filter, it is not required to support
 3434 projections.

3435 **R8.3-2:** If a service supports filtered enumeration using wsman:Filter, it should also support
 3436 filtering using Filter.

3437 **R8.3-3:** If an Enumerate request contains both Filter and wsman:Filter, the service shall return
 3438 a wsman:CannotProcessFilter fault.

3439 Filters are generally intended to select entire XML document representations. However, most query
 3440 languages have both filtering and compositional capabilities in that they can return subsets of the
 3441 original representation, or perform complex operations on the original representation and return
 3442 something entirely new.

3443 This specification places no restriction on the capabilities of the service, but services may elect to
 3444 provide only simple filtering capability and no compositional capabilities. In general, filtering dialects
 3445 fall into the following simple hierarchy:

- 3446 1) simple enumeration with no filtering
- 3447 2) filtered enumeration with no representation change (within the capabilities of XPath, for
 3448 example)
- 3449 3) filtered enumeration in which a subset of each item is selected (within the capabilities of
 3450 XPath, for example)
- 3451 4) composition of new output (XQuery), including simple projection

3452 Most services fall into the first or second category. However, if a service wants to support fragment-
 3453 level enumeration to complement fragment-level access (7.7), the service can implement category 3
 3454 as well. Only rarely do services implement category 4.

3455 [XPath 1.0](#) can be used simply for filtering, or it can be used to send back subsets of the
 3456 representation (or even the values without XML wrappers). In cases where the result is not just
 3457 filtered but also "altered," the technique in 8.6 applies.

3458 If full XPath cannot be supported, a common subset for this purpose is described in D.3 of this
 3459 specification.

3460 EXAMPLE 1: Following is a typical example of the use of XPath in a filter. Assume that each item in the
 3461 enumeration to be delivered has the following XML content:

```

3462 (1) <s:Body>
3463 (2)   ...
3464 (3)   <wsmen:Items>
3465 (4)     <DiskInfo xmlns="...">
3466 (5)       <LogicalDisk>C:</LogicalDisk>
3467 (6)       <CurrentMegabytes>12</CurrentMegabytes>
3468 (7)       <BackupDrive> true </BackupDrive>
3469 (8)     </DiskInfo>
3470 (9)   ...
3471 (10) </wsmen:Items>
3472 (11) </s:Body>
  
```

3473 The anchor point for the XPath evaluation is at the first element of each item within the Items
 3474 wrapper, and it does not reference the s:Body or Items elements. The XPath expression is evaluated
 3475 as if each item in the Items block were a separate document.

3476 EXAMPLE 2: When used for simple document processing, the following four XPath expressions "select" the
 3477 entire DiskInfo node:

```

3478 (12) /
3479 (13) /DiskInfo
3480 (14) ../DiskInfo
3481 (15) .
  
```

3482 If used as a "filter," this XPath expression does not filter out any instances and is the same as
 3483 selecting all instances, or omitting the filter entirely. However, using the following syntax, the XPath
 3484 expression selects the XML node only if the test expression in brackets evaluates to logical "true":

```

3485 (1) ../DiskInfo[LogicalDisk="C:"]
  
```

3486 In this case, the item is selected only if it refers to disk drive "C:"; otherwise the XML node is not
 3487 selected. This XPath expression filters out all DiskInfo instances for other drives.

3488 EXAMPLE 3: Full XPath implementations may support more complex test expressions, as follows:

```

3489 (1) ../DiskInfo[CurrentMegabytes>"10" and CurrentMegabytes <"200"]
  
```

3490 This action selects only drives with free space within the range of values specified.

3491 In essence, the XML form of the event passes logically through the XPath processor to see if it would
 3492 be selected. If so, it is delivered in the enumeration. If not, the item is discarded and not delivered as
 3493 part of the enumeration.

3494 See the related clause (10.2.2) on filtering over subscriptions.

3495 **8.4 Pull**

3496 The Pull operation is initiated by sending a Pull request message to the data source. The Pull request
3497 message shall be of the following form:

```

3498 (1) <s:Envelope ...>
3499 (2)   <s:Header ...>
3500 (3)     <wsa:Action>
3501 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull
3502 (5)     </wsa:Action>
3503 (6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
3504 (7)     <wsa:ReplyTo>wsa:EndpointReference</wsa:ReplyTo>
3505 (8)     <wsa:To>xs:anyURI</wsa:To>
3506 (9)   ...
3507 (10)  </s:Header>
3508 (11)  <s:Body ...>
3509 (12)    <wsmen:Pull ...>
3510 (13)      <wsmen:EnumerationContext>...</wsmen:EnumerationContext>
3511 (14)      <wsmen:MaxTime>xs:duration</wsmen:MaxTime> ?
3512 (15)      <wsmen:MaxElements>xs:long</wsmen:MaxElements> ?
3513 (16)      <wsmen:MaxCharacters>xs:long</wsmen:MaxCharacters> ?
3514 (17)    ...
3515 (18)    </wsmen:Pull>
3516 (19)  </s:Body>
3517 (20) </s:Envelope>

```

3518 The following describes additional, normative constraints on the preceding outline:

3519 /s:Envelope/s:Header/wsa:Action

3520 This required element shall contain the value:

3521 http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull

3522 If a SOAP Action URI is also present in the underlying transport, its value shall convey the same
3523 value.

3524 /s:Envelope/s:Body/wsmen:Pull/wsmen:EnumerationContext

3525 This required element contains the XML data that represents the current enumeration context.

3526 If the enumeration context is not valid, because it has been replaced in the response to another
3527 Pull request, it has completed (EndOfSequence has been returned in a Pull response), it has
3528 been Released, it has expired, or the data source has had to invalidate the context, then the
3529 data source should fail the request, and may generate a wsmen:InvalidEnumerationContext
3530 fault.

3531 The data source may not be able to determine that an enumeration context is not valid,
3532 especially if all of the state associated with the enumeration is kept in the enumeration context
3533 and refreshed on every PullResponse.

3534 /s:Envelope/s:Body/wsmen:Pull/wsmen:MaxTime

3535 This optional element (of type xs:duration) indicates the maximum amount of time the initiator is
3536 willing to allow the data source to assemble the Pull response. When this element is absent, the
3537 data source is not required to limit the amount of time it takes to assemble the Pull response.

3538 This is useful with data sources that accumulate elements over time and package them into a
3539 single Pull response.

3540 /s:Envelope/s:Body/wsmen:Pull/wsmen:MaxElements

3541 This optional element (of type xs:long) indicates the number of items (child elements of Items in
3542 the Pull response) the consumer is willing to accept. When this element is absent, its implied
3543 value is 1. Implementations shall not return more than this number of elements in the Pull

3544 response message. Implementations may return fewer than this number based on either the
3545 MaxTime timeout, the MaxCharacters size limit, or implementation-specific constraints.

3546 /s:Envelope/s:Body/wsmen:Pull/wsmen:MaxCharacters

3547 This optional element (of type xs:long) indicates the maximum size of the returned elements, in
3548 Unicode characters, that the initiator is willing to accept. When this element is absent, the data
3549 source is not required to limit the number of characters in the Pull response. Implementations
3550 shall not return a Pull response message whose Items element is larger than MaxCharacters.
3551 Implementations may return a smaller message based on the MaxTime timeout, the
3552 MaxElements limit, or implementation-specific constraints.

3553 Even if a Pull request contains a MaxCharacters element, the consumer shall be prepared to
3554 receive a Pull response that contains more data characters than specified, as XML
3555 canonicalization or alternate XML serialization algorithms may change the size of the
3556 representation.

3557 It may happen that the next item the data source would return to the consumer is larger than
3558 MaxCharacters. In this case, the data source may skip the item, or may return an abbreviated
3559 representation of the item that fits inside MaxCharacters. If the data source skips the item, it may
3560 return it as part of the response to a future Pull request with a larger value of MaxCharacters, or
3561 it may omit it entirely from the enumeration. If the oversize item is the last item to be returned for
3562 this enumeration context and the data source skips it, it shall include the EndOfSequence item in
3563 the Pull response and invalidate the enumeration context; that is, it may not return zero items but
3564 not consider the enumeration completed. See the discussion of EndOfSequence later in this
3565 clause.

3566 Other components of the preceding outline are not further constrained by this specification.

3567 Upon receipt of a Pull request message, the data source may wait as long as it deems necessary (but
3568 not longer than the value of the MaxTime element, if present) to produce a message for delivery to
3569 the consumer. The data source shall recognize the MaxTime element and return the
3570 wsmen:TimedOut fault if no elements are available prior to the request message's deadline.

3571 However, this fault should not cause the enumeration context to become invalid (of course, the data
3572 source may invalidate the enumeration context for other reasons). That is, the requestor should be
3573 able to issue additional Pull requests using this enumeration context after receiving this fault.

3574 Upon successful processing of a Pull request message, a data source is expected to return a Pull
3575 response message, which shall adhere to the following form:

```

3576 (1) <s:Envelope ...>
3577 (2)   <s:Header ...>
3578 (3)     <wsa:Action>
3579 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse
3580 (5)     </wsa:Action>
3581 (6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
3582 (7)     <wsa:To>xs:anyURI</wsa:To>
3583 (8)     ...
3584 (9)   </s:Header>
3585 (10)  <s:Body ...>
3586 (11)    <wsmen:PullResponse ...>
3587 (12)      <wsmen:EnumerationContext>...</wsmen:EnumerationContext> ?
3588 (13)      <wsmen:Items> ?
3589 (14)        <xs:any> enumeration-specific element </xs:any> +
3590 (15)      </wsmen:Items>
3591 (16)      <wsmen:EndOfSequence/> ?
3592 (17)      ...
3593 (18)    </wsmen:PullResponse>
3594 (19)  </s:Body>
3595 (20) </s:Envelope>

```

3596 The following describes additional, normative constraints on the preceding outline:

3597 /s:Envelope/s:Header/wsa:Action

3598 This required element shall contain the value:

3599 `http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse`

3600 If a SOAP Action URI is also present in the underlying transport, its value shall convey the same
3601 value.

3602 /s:Envelope/s:Body/wsmen:PullResponse/wsmen:EnumerationContext

3603 The optional EnumerationContext element, if present, contains a new XML representation of the
3604 current enumeration context. The consumer is required to replace the prior representation with
3605 the contents of this element.

3606 /s:Envelope/s:Body/wsmen:PullResponse/wsmen:Items/any

3607 The optional Items element contains one or more enumeration-specific elements, one for each
3608 element being returned.

3609 /s:Envelope/s:Body/wsmen:PullResponse/wsmen:EndOfSequence

3610 This optional element indicates that no more elements are available from this enumeration.
3611 Additionally, once this element is returned in a Pull response message, subsequent Pull
3612 requests using that enumeration context should generate an InvalidEnumerationContext fault
3613 message; in any case, they shall not return a valid PullResponse.

3614 At least one of Items or EndOfSequence shall appear. It is possible for both to appear if items are
3615 returned and the sequence is exhausted. Similarly, EnumerationContext and EndOfSequence shall
3616 not both appear; neither may appear, or one without the other, but not both in the same
3617 PullResponse.

3618 The consumer should not issue additional Pull request messages after a Pull response containing an
3619 EndOfSequence element has been returned. Similarly, upon receipt of a Pull response containing an
3620 EndOfSequence element, the consumer should not issue a Release operation to signal that the
3621 enumeration context is no longer needed.

3622 If the consumer does issue a Pull or Release on an invalid enumeration context, the result is
3623 undefined: the data source may ignore the request or may return an InvalidEnumerationContext fault,
3624 as described previously in this clause, or may take some other action.

3625 Because Pull allows the client to specify a wide range of batching and timing parameters, it is often
3626 advisable for the client to know the valid ranges ahead of time. This information can be exported from
3627 the service in the form of metadata, which is beyond the scope of this specification. No message-
3628 based negotiation is available for discovering the valid ranges of the parameters.

3629 Because wsman:MaxEnvelopeSize can be requested for any response in WS-Management, it is used
3630 in the Pull message instead of MaxCharacters, which is generally redundant and preferably is
3631 omitted. However, if wsman:MaxEnvelopeSize is present, it has the following characteristics:

3632 **R8.4-1:** If a service is exposing enumeration operations and supports Pull with the
3633 MaxCharacters element, the service should implement MaxCharacters as a general guideline or
3634 hint, but may ignore it if wsman:MaxEnvelopeSize is present, because it takes precedence. The
3635 service should not fault in the case of a conflict but should observe the wsman:MaxEnvelopeSize
3636 value.

3637 **R8.4-2:** If a service is exposing enumeration operations and supports Pull with the
3638 MaxCharacters element, and a single response element would cause the limit to be exceeded,
3639 the service may return the single element in violation of the hint. However, the service shall not
3640 violate wsman:MaxEnvelopeSize in any case.

3641 A service can send a PullResponse with fewer elements to ensure that the wsman:MaxEnvelopeSize
 3642 is not exceeded. However, if a single item would cause this to be exceeded, then the rules from 6.2
 3643 apply.

3644 In general, MaxCharacters is a hint, and wsman:MaxEnvelopeSize is a strict rule.

3645 **R8.4-3:** If any fault occurs during a Pull, a compliant service should allow the client to retry Pull
 3646 with other parameters, such as a larger limit or with no limit, and attempt to retrieve the items.
 3647 The service should not cancel the enumeration as a whole, but retain enough context to be able
 3648 to retry if the client so wishes. However, the service may cancel the enumeration outright if an
 3649 error occurs with an InvalidEnumerationContext fault.

3650 If a fault occurs with a Pull request, the service generally does not need to cancel the entire
 3651 enumeration, but it can simply freeze the cursor and allow the client to try again.

3652 The EnumerationContext from only the latest response is considered to be valid. Although the service
 3653 can return the same EnumerationContext values with each Pull, it is not required to do so and can in
 3654 fact change the EnumerationContext unpredictably.

3655 **R8.4-4:** A conformant service may ignore MaxTime if wsman:OperationTimeout is also
 3656 specified, as wsman:OperationTimeout takes precedence. These elements have precisely the
 3657 same meaning and may be used interchangeably. If both are used, the service should observe
 3658 only the wsman:OperationTimeout element.

3659 Clients can use wsman:OperationTimeout and wsman:MaxEnvelopeSize rather than MaxTime and
 3660 MaxCharacters to allow for uniform message construction.

3661 Any fault issued for Pull applies to the Pull message itself, not the underlying enumeration that is in
 3662 progress. The most recent EnumerationContext is still considered valid, and if the service allows a
 3663 retry of the most recent Pull message, the client can continue. However, the service can terminate
 3664 early upon encountering any kind of problem (as specified in R8.4-7).

3665 **R8.4-5:** This rule intentionally left blank.

3666 If no content is available, the enumerator is still considered active and the Pull message can be
 3667 retried.

3668 **R8.4-6:** If a service cannot populate the PullResponse with any items before the timeout, it
 3669 should return a wsman:TimedOut fault to indicate that true timeout conditions occurred and that
 3670 the client is not likely to succeed by simply issuing another Pull message. If the service is only
 3671 waiting for results at the point of the timeout, it should return a response with no items and an
 3672 updated EnumerationContext, which may have changed, even though no items were returned, as
 3673 follows:

```

3674 (1) <s:Body>
3675 (2)   <wsman:PullResponse>
3676 (3)   <wsman:EnumerationContext> ...possibly updated...
3677 (4)   </wsman:EnumerationContext>
3678 (5)   <wsman:Items/>
3679 (6)   </wsman:PullResponse>
3680 (7) </s:Body>
```

3681 An empty Items block is essentially a directive from the service to try again. If the service faults with a
 3682 wsman:TimedOut fault, it implies that a retry is not likely to succeed. Typically, the service knows
 3683 which one to return based on its internal state. For example, on the very first Pull message, if the
 3684 service is waiting for another component, a wsman:TimedOut fault could be likely. If the enumeration
 3685 is continuing with no problem and after 50 requests a particular Pull message times out, the service
 3686 can simply send back zero items in the expectation that the client can continue with another Pull
 3687 message.

3688 **R8.4-7:** The service may terminate the entire enumeration early at any time, in which case an
 3689 InvalidEnumerationContext fault is returned. No further operations are possible, including
 3690 Release. In specific cases, such as internal errors or responses that are too large, other faults
 3691 may also be returned. In all such cases, the service should invalidate the enumeration context as
 3692 well.

3693 **R8.4-8:** If the EndOfSequence marker occurs in the PullResponse message, the
 3694 EnumerationContext element shall be omitted, as the enumeration has completed. The client
 3695 cannot subsequently issue a Release message.

3696 Normally, the end of an enumeration in all cases is reported by the EndOfSequence element being
 3697 present in the PullResponse content, not through faults. If the client attempts to enumerate past the
 3698 end of an enumeration, an InvalidEnumerationContext fault is returned. The client need not issue a
 3699 Release message if the EndOfSequence actually occurs because the enumeration is then completed
 3700 and the enumeration context is invalid.

3701 **R8.4-9:** If no MaxElements element is specified, the batch size is 1.

3702 **R8.4-10:** If the value of MaxElements is larger than the service supports, the service may ignore
 3703 the value and use any default maximum of its own.

3704 The service can export its maximum MaxElements value in metadata, but the format and location of
 3705 such metadata is beyond the scope of this specification.

3706 **R8.4-11:** The EnumerationContext element shall be present in all Pull requests, even if the
 3707 service uses a constant value for the lifetime of the enumeration sequence.

3708 8.5 Release

3709 The Release operation is initiated by sending a Release request message to the data source. The
 3710 Release request message shall be of the following form:

```

3711 (1) <s:Envelope ...>
3712 (2)   <s:Header ...>
3713 (3)     <wsa:Action>
3714 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release
3715 (5)     </wsa:Action>
3716 (6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
3717 (7)     <wsa:ReplyTo>wsa:EndpointReference</wsa:ReplyTo>
3718 (8)     <wsa:To>xs:anyURI</wsa:To>
3719 (9)     ...
3720 (10)   </s:Header>
3721 (11)   <s:Body ...>
3722 (12)     <wsmen:Release ...>
3723 (13)       <wsmen:EnumerationContext>...</wsmen:EnumerationContext>
3724 (14)     ...
3725 (15)   </wsmen:Release>
3726 (16)   </s:Body>
3727 (17) </s:Envelope>
  
```

3728 The following describes additional, normative constraints on the preceding outline:

3729 /s:Envelope/s:Header/wsa:Action

3730 This required element shall contain the value:

3731 http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release

3732 If a SOAP Action URI is also present in the underlying transport, its value shall convey the same
 3733 value.

3734 /s:Envelope/s:Body/wsmen:Release/wsmen:EnumerationContext
 3735 This required element contains the XML data that represents the enumeration context being
 3736 abandoned.

3737 Other components of the preceding outline are not further constrained by this specification.

3738 Upon successful processing of a Release request message, a data source is expected to return a
 3739 Release response message, which shall adhere to the following form:

```

3740 (1) <s:Envelope ...>
3741 (2)   <s:Header ...>
3742 (3)     <wsa:Action>
3743 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse
3744 (5)     </wsa:Action>
3745 (6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
3746 (7)     <wsa:To>xs:anyURI</wsa:To>
3747 (8)     ...
3748 (9)   </s:Header>
3749 (10)  <s:Body />
3750 (11) </s:Envelope>
  
```

3751 The following describes additional, normative constraints on the preceding outline:

3752 /s:Envelope/s:Header/wsa:Action

3753 This required element shall contain the value:

3754 http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse

3755 If a SOAP Action URI is also present in the underlying transport, its value shall convey the same
 3756 value.

3757 Release is used only to perform an early cancellation of the enumeration. In cases in which it is not
 3758 actually needed, the implementation can expose a dummy implementation that always succeeds.
 3759 This promotes uniform client-side messaging.

3760 **R8.5-1:** The service shall recognize and process the Release message if the enumeration is
 3761 terminated early. If an EndOfSequence marker occurs in a PullResponse message, the
 3762 enumerator is already completed and a Release message cannot be issued because no up-to-
 3763 date EnumerationContext exists.

3764 **R8.5-2:** The client may fail to deliver the Release message in a timely fashion or may never
 3765 send it. A conformant service may terminate the enumeration after a suitable idle time has
 3766 expired, and any attempt to reuse the enumeration context shall result in an
 3767 InvalidEnumerationContext fault.

3768 **R8.5-3:** This rule intentionally left blank.

3769 **R8.5-4:** The service may accept a Release message asynchronously to any Pull requests
 3770 already in progress and cancel the enumeration. The service may refuse such an asynchronous
 3771 request and fault it with a wsmen:UnsupportedFeature fault with the following detail code:

3772 http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AsynchronousRequest

3773 The service may also queue or block the request and serialize it so that it is processed after the Pull
 3774 message.

3775 In most cases, it is desirable to be able to asynchronously cancel an outstanding Pull message. This
 3776 capability requires the service to be able to receive the Release message asynchronously while still
 3777 processing a pending Pull message. Further, it requires that the EnumerationContext element contain
 3778 information that is constant between Pull operations.

3779 NOTE: If the value of EnumerationContext is a simple increasing integer, Release always uses a previous value,
 3780 so the service may consider it to be invalid. If the EnumerationContext element contains a value that is constant
 3781 across Pull requests (as well as any other information that the service might need), the service can more easily
 3782 implement the cancellation.

3783 8.6 Ad-Hoc Queries and Fragment-Level Enumerations

3784 As discussed in 7.7, it is desirable that clients be able to access subsets of a representation. This is
 3785 especially important in the area of query processing, where users routinely want to execute XPath or
 3786 XQuery operations over the representation to receive ad-hoc results.

3787 Because SOAP messages need to conform to known schemas, and ad-hoc queries return results
 3788 that are dynamically generated and might conform to no schema, the wsman:XmlFragment wrapper
 3789 from 7.7 is used to wrap the responses.

3790 **R8.6-1:** The service may support ad-hoc compositional queries, projections, or enumerations of
 3791 fragments of the representation objects by supplying a suitable dialect in the wsman:Filter. The
 3792 resulting set of Items in the PullResponse element (or EnumerateResponse element if
 3793 OptimizedEnumeration is used) should be wrapped with wsman:XmlFragment wrappers as
 3794 follows:

```

3795 (1) <s:Body>
3796 (2)   <wsmen:PullResponse>
3797 (3)     <wsmen:EnumerationContext> ..possibly updated..
3798 (4)   </wsmen:EnumerationContext>
3799 (5)   <wsmen:Items>
3800 (6)     <wsman:XmlFragment>
3801 (7)       XML content
3802 (8)     </wsman:XmlFragment>
3803 (9)     <wsman:XmlFragment>
3804 (10)      XML content
3805 (11)    </wsman:XmlFragment>
3806 (12)    ...
3807 (13)   </wsmen:Items>
3808 (14)  </wsmen:PullResponse>
3809 (15) </s:Body>
  
```

3810 The schema for wsman:XmlFragment contains a directive to suppress schema validation, allowing a
 3811 validating parser to accept ad-hoc content produced by the query processor acting behind the
 3812 enumeration.

3813 [XPath 1.0](#) and [XQuery 1.0](#) already support returning subsets or compositions of representations, so
 3814 they are suitable for use in this regard.

3815 **R8.6-2:** If the service does not support fragment-level enumeration, it should return a
 3816 wsman:FilterDialectRequestedUnavailable fault, the same as for any other unsupported dialect.

3817 The XPath expression used for filtering is still as described in the Enumeration clauses (see 8.2,
 3818 8.2.2, 8.2.3). The wsman:XmlFragment wrappers are applied after the XPath is evaluated to prevent
 3819 schema violations if the XPath selects node sets that are fragments and not legal according to the
 3820 original schema.

3821 8.7 Enumeration of EPRs

3822 Typically, inferring the EPR of an enumerated object simply by inspection is not possible. In many
 3823 cases, it is desirable to enumerate the EPRs of objects rather than the objects themselves. Such
 3824 EPRs can be usable in subsequent Get or Delete requests, for example. Similarly, it is often desirable
 3825 to enumerate both the objects and the associated EPRs.

3826 The default behavior for Enumerate is as defined in 8.1. However, WS-Management provides an
 3827 additional extension for controlling the output of the enumeration.

3828 **R8.7-1:** A service may optionally support the wsman:EnumerationMode modifier element with a
 3829 value of *EnumerateEPR*, which returns only the EPRs of the objects as the result of the
 3830 enumeration.

3831 EXAMPLE 1:

```

3832 (1) <s:Envelope ...>
3833 (2)   <s:Header>
3834 (3)     ...
3835 (4)     <wsa:Action>
3836 (5)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
3837 (6)     </wsa:Action>
3838 (7)     ...
3839 (8)   </s:Header>
3840 (9)   <s:Body>
3841 (10)    <wsmen:Enumerate>
3842 (11)      <wsman:Filter Dialect="..."> filter </wsman:Filter>
3843 (12)      <wsman:EnumerationMode> EnumerateEPR </wsman:EnumerationMode>
3844 (13)    ...
3845 (14)    </wsmen:Enumerate>
3846 (15)  </s:Body>
3847 (16) </s:Envelope>
  
```

3848 EXAMPLE 2: The hypothetical response would appear as in the following example:

```

3849 (17) <s:Body>
3850 (18)   <wsmen:PullResponse>
3851 (19)     <wsmen:Items>
3852 (20)       <wsa:EndpointReference> ... </wsa:EndpointReference>
3853 (21)       <wsa:EndpointReference> ... </wsa:EndpointReference>
3854 (22)       <wsa:EndpointReference> ... </wsa:EndpointReference>
3855 (23)     ...
3856 (24)   </wsmen:Items>
3857 (25) </wsmen:PullResponse>
3858 (26) </s:Body>
  
```

3859 The filter, if any, is still applied to the enumeration, but the response contains only the EPRs of the
 3860 items that would have been returned. These EPRs are intended for use in subsequent Get
 3861 operations.

3862 **R8.7-2:** A service may optionally support the wsman:EnumerationMode modifier with the value
 3863 of *EnumerateObjectAndEPR*. If present, the enumerated objects are wrapped in a wsman:Item
 3864 element that juxtaposes two XML representations: the payload representation followed by the
 3865 associated wsa:EndpointReference.

3866 EXAMPLE 3: The wsman:EnumerationMode example appears as follows:

```

3867 (1) <s:Header>
3868 (2)   ...
3869 (3)   <wsa:Action>
3870 (4)     http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate
3871 (5)   </wsa:Action>
3872 (6) </s:Header>
3873 (7) <s:Body>
3874 (8)   <wsmen:Enumerate>
3875 (9)     <wsman:Filter Dialect="..."> filter </wsman:Filter>
  
```

```

3876 (10)    <wsman:EnumerationMode> EnumerateObjectAndEPR
3877    </wsman:EnumerationMode>
3878 (11)    ...
3879 (12)    </wsmen:Enumerate>
3880 (13)    </s:Body>

```

3881 **EXAMPLE 4:** The response appears as follows:

```

3882 (1)    <s:Body>
3883 (2)    <wsmen:PullResponse>
3884 (3)    <wsmen:Items>
3885 (4)    <wsman:Item>
3886 (5)    <PayloadObject xmlns="..."> ... </PayloadObject> <!-- Object -->
3887 (6)    <wsa:EndpointReference> ... </wsa:EndpointReference> <!-- EPR -->
3888 (7)    </wsman:Item>
3889 (8)    <wsman:Item>
3890 (9)    <PayloadObject xmlns="..."> ... </PayloadObject> <!-- Object -->
3891 (10)   <wsa:EndpointReference> ... </wsa:EndpointReference> <!-- EPR -->
3892 (11)   </wsman:Item>
3893 (12)   ...
3894 (13)   </wsmen:Items>
3895 (14)   </wsmen:PullResponse>
3896 (15)   </s:Body>

```

3897 In the preceding example, each item is wrapped in a wsman:Item wrapper (line 8), which itself contains the
 3898 representation object (line 9) followed by its EPR (line 10). As many wsman:Item objects may be present as is
 3899 consistent with other encoding limitations.

3900 **R8.7-3:** If a service does not support the wsman:EnumerationMode modifier, it shall return a
 3901 fault of wsman:UnsupportedFeature with the following detail code:

3902 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EnumerationMode`

3903 8.8 Renew

3904 To renew an enumeration, the consumer sends a request of the following form to the data source:

```

3905 (1)    <s:Envelope ...>
3906 (2)    <s:Header ...>
3907 (3)    <wsa:Action>
3908 (4)    http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew
3909 (5)    </wsa:Action>
3910 (6)    <wsa:MessageID>xs:anyURI</wsa:MessageID>
3911 (7)    <wsa:FaultTo>endpoint-reference</wsa:FaultTo> ?
3912 (8)    <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
3913 (9)    <wsa:To>xs:anyURI</wsa:To>
3914 (10)   ...
3915 (11)   </s:Header>
3916 (12)   <s:Body ...>
3917 (13)   <wsmen:Renew ...>
3918 (14)   <wsmen:EnumerationContext>...</wsmen:EnumerationContext>
3919 (15)   <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
3920 (16)   ...
3921 (17)   </wsmen:Renew>
3922 (18)   </s:Body>
3923 (19)   </s:Envelope>

```

3924 Components of the preceding outline are additionally constrained as for a request to create an
 3925 enumeration with the following addition(s):

3926 /s:Envelope/s:Body/*/wsmen:EnumerationContext

3927 This required element contains the XML data that represents the current enumeration context.

3928 If the enumeration context is not valid, either because it has been replaced in the response to
 3929 another Pull request, or because it has completed (EndOfSequence has been returned in a Pull
 3930 response), or because it has been Released, or because it has expired, or because the data
 3931 source has had to invalidate the context, then the data source should fail the request, and may
 3932 generate a wsmen:InvalidEnumerationContext fault.

3933 The data source may not be able to determine that an enumeration context is not valid,
 3934 especially if all of the state associated with the enumeration is kept in the enumeration context
 3935 and refreshed on every PullResponse.

3936 Other components of the preceding outline are not further constrained by this specification.

3937 If the data source accepts a request to renew an enumeration, it shall reply with a response of the
 3938 following form:

```

3939 (1) <s:Envelope ...>
3940 (2)   <s:Header ...>
3941 (3)     <wsa:Action>
3942 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse
3943 (5)     </wsa:Action>
3944 (6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
3945 (7)     <wsa:To>xs:anyURI</wsa:To>
3946 (8)   ...
3947 (9) </s:Header>
3948 (10) <s:Body ...>
3949 (11)   <wsmen:RenewResponse ...>
3950 (12)     <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
3951 (13)     <wsmen:EnumerationContext>...</wsmen:EnumerationContext> ?
3952 (14)   ...
3953 (15)   </wsmen:RenewResponse>
3954 (16) </s:Body>
3955 (17) </s:Envelope>
  
```

3956 Components of the preceding outline listed are constrained as for a response to an Enumerate
 3957 request with the following addition:

3958 /s:Envelope/s:Body/wsmen:RenewResponse/wsmen:Expires

3959 If the requested expiration is a duration, then the implied start of that duration is the time when
 3960 the data source starts processing the Renew request.

3961 /s:Envelope/s:Body/wsmen:RenewResponse/wsmen:EnumerationContext

3962 This element is optional in this response.

3963 If the data source chooses not to renew this enumeration, the request shall fail, and the data
 3964 source should generate a wsmen:UnableToRenew fault indicating that the renewal was not
 3965 accepted.

3966 Other components of the preceding outline are not further constrained by this specification.

3967 8.9 GetStatus

3968 To get the status of an enumeration, the subscriber sends a request of the following form to the data
3969 source:

```

3970 (1) <s:Envelope ...>
3971 (2)   <s:Header ...>
3972 (3)     <wsa:Action>
3973 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus
3974 (5)     </wsa:Action>
3975 (6)     <wsa:MessageID>xs:anyURI</wsa:MessageID>
3976 (7)     <wsa:FaultTo>endpoint-reference</wsa:FaultTo> ?
3977 (8)     <wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
3978 (9)     <wsa:To>xs:anyURI</wsa:To>
3979 (10)    ...
3980 (11)   </s:Header>
3981 (12)   <s:Body ...>
3982 (13)     <wsmen:GetStatus ...>
3983 (14)       <wsmen:EnumerationContext>...</wsmen:EnumerationContext> ?
3984 (15)     ...
3985 (16)   </wsmen:GetStatus>
3986 (17) </s:Body>
3987 (18) </s:Envelope>

```

3988 Components of the preceding outline are additionally constrained as for a request to renew an
3989 enumeration. Other components of the preceding outline are not further constrained by this
3990 specification.

3991 If the enumeration is valid and has not expired, the data source shall reply with a response of the
3992 following form:

```

3993 (1) <s:Envelope ...>
3994 (2)   <s:Header ...>
3995 (3)     <wsa:Action>
3996 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse
3997 (5)     </wsa:Action>
3998 (6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
3999 (7)     <wsa:To>xs:anyURI</wsa:To>
4000 (8)    ...
4001 (9)   </s:Header>
4002 (10)  <s:Body ...>
4003 (11)    <wsmen:GetStatusResponse ...>
4004 (12)      <wsmen:Expires>[xs:dateTime | xs:duration]</wsmen:Expires> ?
4005 (13)    ...
4006 (14)  </wsmen:GetStatusResponse>
4007 (15) </s:Body>
4008 (16) </s:Envelope>

```

4009 Components of the preceding outline are constrained as for a response to a Renew request. Other
4010 components of the preceding outline are not further constrained by this specification.

4011 8.10 EnumerationEnd

4012 If the data source terminates an enumeration unexpectedly, the data source should send an
4013 EnumerationEnd SOAP message to the endpoint reference indicated when the enumeration was
4014 created. The message shall be of the following form:

```

4015 (1) <s:Envelope ...>
4016 (2)   <s:Header ...>
4017 (3)     <wsa:Action>
4018 (4)       http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd

```

```

4019 (5)      </wsa:Action>
4020 (6)      <wsa:To>xs:anyURI</wsa:To>
4021 (7)      ...
4022 (8)      </s:Header>
4023 (9)      <s:Body ...>
4024 (10)     <wsmen:EnumerationEnd ...>
4025 (11)     <wsmen:EnumerationContext>...</wsmen:EnumerationContext>
4026 (12)     <wsmen:Code>
4027 (13)     [
4028 (14)     http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceShuttingDown
4029 (15)     | http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceCancelling
4030 (16)     ]
4031 (17)     </wsmen:Code>
4032 (18)     <wsmen:Reason xml:lang="language identifier" >
4033 (19)     xs:string
4034 (20)     </wsmen:Reason> ?
4035 (21)     ...
4036 (22)     </wsmen:EnumerationEnd>
4037 (23)    </s:Body>
4038 (24)    </s:Envelope>

```

4039 The following describes additional, normative constraints on the preceding outline:

4040 /s:Envelope/s:Body/wsmen:Release/wsmen:EnumerationContext

4041 This required element contains the XML data that represents the enumeration context being
 4042 terminated. It is recommended that consumers DO NOT attempt to compare this element
 4043 against any collection of wsmen:EnumerationContext elements for purposes of correlation,
 4044 because that requires the ability to compare arbitrary XML elements. If consumers wish to
 4045 correlate this message against their outstanding contexts, it is recommend that they use the
 4046 reference parameters of the /wsmen:Enumerate/wsmen:EndTo EPR.

4047 /s:Envelope/s:Body/wsmen:EnumerationEnd/wsmen:Code =

4048 "http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceShuttingDown"

4049 This value shall be used if the data source terminated the enumeration because the source is
 4050 being shut down in a controlled manner; that is, if the data source is being shut down but has the
 4051 opportunity to send an EnumerationEnd message before it exits.

4052 /s:Envelope/s:Body/wsmen:EnumerationEnd/wsmen:Code =

4053 "http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceCancelling"

4054 This value shall be used if the data source terminated the enumeration for some other reason
 4055 before it expired.

4056 /s:Envelope/s:Body/wsmen:EnumerationEnd/wsmen:Reason

4057 This optional element contains text, in the language specified by the @xml:lang attribute,
 4058 describing the reason for the unexpected enumeration termination.

4059 Other components of the preceding outline are not further constrained by this specification.

4060 9 Custom Actions (Methods)

4061 Custom actions, or "methods," are ordinary SOAP messages with unique Actions. An implementation
 4062 can support resource-specific methods in any form, subject to the addressing model and restrictions
 4063 described in clause 5 of this specification.

4064 **R9-1:** A conformant service may expose any custom actions or methods.

- 4065 **R9-2:** If custom methods are exported, Addressing rules, as described elsewhere in this
 4066 specification, shall be observed, and each custom method shall have a unique `wsa:Action`.
- 4067 **R9-3:** If a request does not contain the correct parameters for the custom action, the service
 4068 may return a `wsman:InvalidParameter` fault. Fault details for incorrect type and incorrect name
 4069 may also be included.
- 4070 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch` (incorrect type)
 4071 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName` (incorrect name)
- 4072 As defined by Addressing, the Action URI is used to describe the semantics of the operation and the
 4073 `wsa:To` element describes the destination of the message. A custom method thus has a dedicated
 4074 Addressing Action URI.
- 4075 Because options are a parameterization technique for message types that are not user-extensible,
 4076 such as the resource access operations, they are not appropriate for use as a custom method or
 4077 combined with a custom method. Custom operations defined in a WSDL document define any
 4078 required parameters and thus expose naming and type checking in a stringent way. Mixing
 4079 `wsman:OptionSet` with a strongly typed WSDL operation is likely to lead to confusion.

4080 10 Notifications (Eventing)

4081 10.1 General

- 4082 Management infrastructures often want to receive messages when events occur in remote
 4083 management services and applications. A mechanism for registering interest is needed because the
 4084 set of Web services interested in receiving such messages is often unknown in advance or changes
 4085 over time. This specification defines a set of operations for one management Web service (called a
 4086 "subscriber") to register interest (called a "subscription") with another management Web service
 4087 (called an "event source") in receiving messages about events (called "notifications" or "event
 4088 messages"). The subscriber may manage the subscription by interacting with a Web service (called
 4089 the "subscription manager") designated by the event source.
- 4090 To improve robustness, a subscription may be leased by an event source to a subscriber, and the
 4091 subscription expires over time. The subscription manager provides the ability for the subscriber to
 4092 renew or cancel the subscription before it expires.
- 4093 There are many mechanisms by which event sources may deliver events to event sinks. This
 4094 specification provides an extensible way for subscribers to identify the delivery mechanism they
 4095 prefer. While asynchronous, pushed delivery is defined here; the intent is that there should be no
 4096 limitation or restriction on the delivery mechanisms capable of being supported by this specification.
- 4097 To create, renew, and delete subscriptions, subscribers send request messages to event sources and
 4098 subscription managers.
- 4099 When an event source accepts a request to create a subscription, it typically does so for a given
 4100 amount of time, although an event source may accept an indefinite subscription with no time-based
 4101 expiration. If the subscription manager accepts a renewal request, it updates that amount of time.
 4102 During that time, notifications are delivered by the event source to the requested event sink. An event
 4103 source may support filtering to limit notifications that are delivered to the event sink; if it does, and a
 4104 subscribe request contains a filter, the event source sends only notifications that match the requested
 4105 filter. The event source sends notifications until one of the following happens: the subscription
 4106 manager accepts an unsubscribe request for the subscription, the subscription expires without being
 4107 renewed, or the event source cancels the subscription prematurely. In this last case, the event source
 4108 makes a best effort to indicate why the subscription ended.

4109 In the absence of reliable messaging at the application layer (for example, [WS-ReliableMessaging]),
 4110 messages defined herein are delivered using the quality of service of the underlying transport(s) and
 4111 on a best-effort basis at the application layer.

4112 If a managed entity emits events, it can publish those events using this publish-and-subscribe
 4113 mechanism and paradigms.

4114 **R10.1-1:** If a resource can emit events and allows clients to subscribe to and receive notification
 4115 messages, it shall do so by implementing the operations as specified in this clause.

4116 **R10.1-2:** If the eventing mechanism as described in this clause is supported, the
 4117 wsme:Subscribe, wsme:Renew, and wsme:Unsubscribe messages shall be supported. The
 4118 wsme:SubscriptionEnd message is optional. The wsme:GetStatus message in a constrained
 4119 environment is a candidate for exclusion. If this message is not supported, then a
 4120 wsa:ActionNotSupported fault shall be returned in response to this request.

4121 10.2 Subscribe

4122 In some scenarios the event source itself manages the subscriptions it has created. In other
 4123 scenarios, for example a geographically distributed publish-and-subscribe system, it may be useful to
 4124 delegate the management of a subscription to another Web service. To support this flexibility, the
 4125 response to a subscription request to an event source includes the EPR of a service that the
 4126 subscriber may interact with to manage this subscription. This EPR should be the target for future
 4127 requests to renew or cancel the subscription. It may address the same Web service (Address and
 4128 ReferenceParameters) as the event source itself, or it may address some other Web service to which
 4129 the event source has delegated management of this subscription; however, the full subscription
 4130 manager EPR (Address and ReferenceParameters) must be unique for each subscription.

4131 We use the term "subscription manager" in this specification to refer to the Web service that manages
 4132 the subscription, whether it is the event source itself or some separate Web service.

4133 To create a subscription, a subscriber sends a request message of the following form to an event
 4134 source:

```

4135 (1) <s:Envelope ...>
4136 (2)   <s:Header ...>
4137 (3)     <wsa:Action>
4138 (4)       http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
4139 (5)     </wsa:Action>
4140 (6)   ...
4141 (7) </s:Header>
4142 (8) <s:Body ...>
4143 (9)   <wsme:Subscribe ...>
4144 (10)    <wsme:EndTo>endpoint-reference</wsme:EndTo> ?
4145 (11)    <wsme:Delivery Mode="xs:anyURI"? >xs:any</wsme:Delivery>
4146 (12)    <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
4147 (13)    <wsme:Filter Dialect="xs:anyURI"? > xs:any </wsme:Filter> ?
4148 (14)  ...
4149 (15)  </wsme:Subscribe>
4150 (16) </s:Body>
4151 (17) </s:Envelope>
  
```

4152 The following describes additional, normative constraints on the preceding outline:

4153 /s:Envelope/s:Header/wsa:Action

4154 If a SOAP Action URI is used in the binding for SOAP, the value indicated herein shall be used
 4155 for that URI.

- 4156 /s:Envelope/s:Body/*/wsme:EndTo
- 4157 Where to send a SubscriptionEnd message if the subscription is terminated unexpectedly. If
4158 present, this element shall be of type wsa:EndpointReferenceType. The default is not to send
4159 this message. The endpoint referenced by this EPR shall implement a binding of the
4160 "EndToEndpoint" portType described in ANNEX I.
- 4161 /s:Envelope/s:Body/*/wsme:Delivery
- 4162 A delivery destination for notification messages, using some delivery mode.
- 4163 /s:Envelope/s:Body/*/wsme:Delivery/@Mode
- 4164 The delivery mode to be used for notification messages sent in relation to this subscription.
4165 Implied value is "http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push", which
4166 indicates that Push Mode delivery should be used.
- 4167 If the event source does not support the requested delivery mode, the request shall fail, and the
4168 event source may generate a wsme:DeliveryModeRequestedUnavailable fault indicating that the
4169 requested delivery mode is not supported.
- 4170 /s:Envelope/s:Body/*/wsme:Delivery/@Mode="http://schemas.xmlsoap.org/ws/2004/08/eventing/Deliv
4171 eryModes/Push"
- 4172 The value of /s:Envelope/s:Body/*/wsme:Delivery is a single element, NotifyTo, that contains the
4173 endpoint reference to which notification messages should be sent.
- 4174 /s:Envelope/s:Body/*/wsme:Expires
- 4175 Requested expiration time for the subscription. (No implied value.) The event source defines the
4176 actual expiration and is not constrained to use a time less or greater than the requested
4177 expiration. The expiration time may be a specific time or a duration from the subscription's
4178 creation time. Both specific times and durations are interpreted based on the event source's
4179 clock.
- 4180 If this element does not appear, then the request is for a subscription that will not expire. That is,
4181 the subscriber is requesting the event source to create a subscription with an indefinite lifetime. If
4182 the event source grants such a subscription, it may be terminated by the subscriber using an
4183 Unsubscribe request, or it may be terminated by the event source at any time for reasons such
4184 as connection termination, resource constraints, or system shut-down.
- 4185 If the expiration time is either a zero duration or a specific time that occurs in the past according
4186 to the event source, then the request shall fail, and the event source may generate a
4187 InvalidExpirationTime fault indicating that an invalid expiration time was requested.
- 4188 Some event sources may not have a "wall time" clock available, and so are only able to accept
4189 durations as expirations. If such a source receives a Subscribe request containing a specific time
4190 expiration, then the request may fail; if so, the event source may generate an
4191 UnsupportedExpirationType fault indicating that an unsupported expiration type was requested.
- 4192 /s:Envelope/s:Body/*/wsme:Filter
- 4193 A Boolean expression in some dialect, either as a string or as an XML fragment. If the
4194 expression evaluates to false for a notification, the notification shall not be sent to the event sink.
4195 Implied value is an expression that always returns true. If the event source does not support
4196 filtering, then a request that specifies a filter shall fail, and the event source may generate a
4197 wsme:FilteringNotSupported fault indicating that filtering is not supported.

4198 If the event source supports filtering but cannot honor the requested filtering, the request shall
 4199 fail, and the event source may generate a wsme:FilteringRequestedUnavailable fault indicating
 4200 that the requested filter dialect is not supported.

4201 /s:Envelope/s:Body/*/wsme:Filter/@Dialect

4202 Implied value is "http://www.w3.org/TR/1999/REC-xpath-19991116".

4203 While an XPath predicate expression provides great flexibility and power, alternate filter dialects
 4204 may be defined. For instance, a simpler, less powerful dialect might be defined for resource-
 4205 constrained implementations, or a new dialect might be defined to support filtering based on data
 4206 not included in the notification message itself. If desired, a filter dialect could allow the definition
 4207 of a composite filter that contained multiple filters from other dialects.

4208 /s:Envelope/s:Body/*/wsme:Filter/@Dialect=" http://www.w3.org/TR/1999/REC-xpath-19991116"

4209 Value of /s:Envelope/s:Body/*/wsme:Filter is an XPath [\[XPath 1.0\]](#) predicate expression
 4210 (PredicateExpr); the context of the expression is:

- 4211 • **Context Node:** the SOAP Envelope containing the notification
- 4212 • **Context Position:** 1
- 4213 • **Context Size:** 1
- 4214 • **Variable Bindings:** None
- 4215 • **Function Libraries:** Core Function Library [\[XPath 1.0\]](#)
- 4216 • **Namespace Declarations:** The [in-scope namespaces] property [\[XML Infoset\]](#) of
 4217 /s:Envelope/s:Body/*/wsme:Filter

4218 Other message information headers defined by Addressing may be included in the request and
 4219 response messages, according to the usage and semantics defined in Addressing.

4220 Other components of the preceding outline are not further constrained by this specification.

4221 If the event source accepts a request to create a subscription, it shall reply with a response of the
 4222 following form:

```

4223 (1) <s:Envelope ...>
4224 (2)   <s:Header ...>
4225 (3)     <wsa:Action>
4226 (4)       http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
4227 (5)     </wsa:Action>
4228 (6)   ...
4229 (7) </s:Header>
4230 (8)   <s:Body ...>
4231 (9)     <wsme:SubscribeResponse ...>
4232 (10)      <wsme:SubscriptionManager>
4233 (11)        wsa:EndpointReferenceType
4234 (12)      </wsme:SubscriptionManager>
4235 (13)      <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires>
4236 (14)      ...
4237 (15)    </wsme:SubscribeResponse>
4238 (16)  </s:Body>
4239 (17) </s:Envelope>
  
```

- 4240 The following describes additional, normative constraints on the preceding outline:
- 4241 `/s:Envelope/S:Header/wsa:RelatesTo`
- 4242 Shall be the value of the `wsa:MessageID` of the corresponding request.
- 4243 `/s:Envelope/s:Body/*/wsme:SubscriptionManager`
- 4244 The EPR of the subscription manager for this subscription.
- 4245 In some cases, it is convenient for all EPRs issued by a single event source to address a single
 4246 Web service and use a reference parameter to distinguish among the active subscriptions. For
 4247 convenience in this common situation, this specification defines a global element, Identifier of
 4248 type `xs:anyURI`, that may be used as a distinguishing reference parameter if desired by the
 4249 event source.
- 4250 `/s:Envelope/s:Body/*/wsme:Expires`
- 4251 The expiration time assigned by the event source. The expiration time may be either an absolute
 4252 time or a duration but should be of the same type as the requested expiration (if any).
- 4253 If this element does not appear, then the subscription will not expire. That is, the subscription
 4254 has an indefinite lifetime. It may be terminated by the subscriber using an Unsubscribe request,
 4255 or it may be terminated by the event source at any time for reasons such as connection
 4256 termination, resource constraints, or system shut-down.
- 4257 Other components of the preceding outline are not further constrained by this specification.
- 4258 If the event source chooses not to accept a subscription, the request shall fail, and the event source
 4259 may generate a `wsme:EventSourceUnableToProcess` fault indicating that the request was not
 4260 accepted.
- 4261 This specification does not constrain notifications because any message may be a notification.
- 4262 However, if a subscribing event sink wishes to have notifications specifically marked, it may specify
 4263 literal SOAP header blocks in the Subscribe request, in the
 4264 `/s:Envelope/s:Body/wsme:Subscribe/wsme:NotifyTo/wsa:ReferenceParameters` elements; per
 4265 Addressing, the event source shall include each such literal SOAP header block in every notification
 4266 sent to the endpoint addressed by `/s:Envelope/s:Body/wsme:Subscribe/wsme:NotifyTo`.
- 4267 **10.2.1 General**
- 4268 WS-Management uses Subscribe substantially as documented here, except that the
 4269 WS-Management default addressing model is incorporated as described in 5.1.
- 4270 **R10.2.1-1:** The identity of the event source shall be based on the Addressing EPR.
- 4271 **R10.2.1-2:** If the service cannot support the requested addressing, it should return a
 4272 `wsman:UnsupportedFeature` fault with the following detail code:
- 4273 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode`
- 4274 Verifying that the address is usable allows errors to be detected at the time the subscription is
 4275 created. For example, if the address cannot be reached due to firewall configuration and the service
 4276 can detect this, telling the client allows for it to be corrected immediately.
- 4277 **R10.2.1-3:** Because many delivery modes require a separate connection to deliver the event,
 4278 the service should comply with the security profiles defined in clause 11 of this specification, if
 4279 HTTP or HTTPS is used to deliver events. If no security is specified, the service may attempt to

4280 use default security mechanisms, or return a wsman:UnsupportedFeature fault with the following
 4281 detail code:

4282 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress`

4283 Because clients might need to have client-side context sent back with each event delivery, the
 4284 NotifyTo address in the Delivery block can be used for this purpose. This NotifyTo EPR can contain
 4285 any number of client-defined reference parameters.

4286 **R10.2.1-4:** A service may validate the address by attempting a connection while the Subscribe
 4287 request is being processed to ensure delivery can occur successfully. If the service determines
 4288 that the address is not valid or permissions cannot be acquired, it should emit a
 4289 wsman:EventDeliverToUnusable fault.

4290 This situation can occur when the address is incorrect or when the event source cannot acquire
 4291 permissions to deliver events properly.

4292 **R10.2.1-5:** Any reference parameters supplied in the NotifyTo address shall be included with
 4293 each event delivery as top-level headers as specified 5.4. If EndTo is supported, this behavior
 4294 applies as well.

4295 When the default addressing model is used by the service, the ResourceURI is often used to
 4296 reference the logical event source, and selector values can additionally be used to indicate a real or
 4297 virtual log within the scope of that source, or might even be used to limit the types or groups of events
 4298 available. This action can logically overlap with the Filter mechanism in the subscription body itself, so
 4299 due consideration should be given to the interplay among the address of the event source, the types
 4300 of events it can publish, and the subscription-level filtering.

4301 If a client needs to have events delivered to more than one destination, more than one subscription is
 4302 required.

4303 **R10.2.1-6:** If the events contain localized content, the service should accept a subscription with
 4304 a wsman:Locale block acting as a hint (see 6.3) within the Delivery block of the Subscribe
 4305 message. The language is encoded in an xml:lang attribute using [RFC 5646](#) language codes.

4306 The service attempts to localize any descriptive content to the specified language when delivering
 4307 such events, which is outlined as follows:

```
4308 (1) <wsme:Subscribe>
4309 (2)   <wsme:Delivery>
4310 (3)     <wsme:NotifyTo> ... </wsme:NotifyTo>
4311 (4)     <wsman:Locale xml:lang="language-code"/>
4312 (5)   </wsme:Delivery>
4313 (6) </wsme:Subscribe>
```

4314 NOTE: In this context, the wsman:Locale element (defined in 6.3) is not a SOAP header and mustUnderstand
 4315 cannot be used.

4316 **R10.2.1-7:** The service should accept a subscription with a wsman:ContentEncoding block
 4317 within the Delivery block of the Subscribe message. This block acts as a hint to indicate how the
 4318 delivered events are to be encoded. The two standard xs:language tokens defined for this
 4319 purpose are "UTF-8" or "UTF-16", although other encoding formats may be specified if
 4320 necessary. The service should attempt to encode the events using the requested language token,
 4321 as in the following example:

4322 EXAMPLE:

```
4323 (1) <wsme:Subscribe>
4324 (2)   <wsme:Delivery>
4325 (3)     ...
```

```

4326 (4) <wsme:NotifyTo> ... </wsme:NotifyTo>
4327 (5) <wsman:ContentEncoding> UTF-16 </wsman:ContentEncoding>
4328 (6) </wsme:Delivery>
4329 (7) </wsme:Subscribe>

```

4330 10.2.2 Filtering

4331 Filter expression is constrained to be a Boolean predicate. To support ad hoc queries including
 4332 projections, WS-Management defines a wsman:Filter element of exactly the same form as what is
 4333 used in the Subscribe operation except that the filter expression is not constrained to be a Boolean
 4334 predicate. This allows the use of subscriptions using existing query languages such as SQL and CQL,
 4335 which combine predicate and projection information in the same syntax. The use of projections is
 4336 defined by the filter dialect, not by WS-Management.

4337 If the filter dialect for either Filter or wsman:Filter used for the Subscribe message is
 4338 <http://www.w3.org/TR/1999/REC-xpath-19991116> (the default dialect in both cases), the context node
 4339 is the SOAP Envelope element.

4340 WS-Management defines the wsman:Filter element as a child of the Subscribe element.

4341 WS-Management defines the wsman:Filter element to allow projections, which is outlined as follows:

```

4342 (1) <wsman:Filter Dialect="xs:anyURI"?> xs:any </wsman:Filter>

```

4343 The Dialect attribute is optional. When not specified, it has the following implied value:

4344 <http://www.w3.org/TR/1999/REC-xpath-19991116>

4345 This dialect allows any full XPath expression or subset to be used.

4346 **R10.2.2-1:** If a service supports filtered subscriptions using Filter, it shall also support filtering
 4347 using wsman:Filter. This rule allows client stacks to always pick the wsman XML namespace for
 4348 the Filter element. Even though a service supports wsman:Filter, it is not required to support
 4349 projections.

4350 **R10.2.2-2:** If a service supports filtered subscriptions using wsman:Filter, it should also support
 4351 filtering using Filter.

4352 **R10.2.2-3:** If a Subscribe request contains both Filter and wsman:Filter, the service shall return
 4353 a wsa:InvalidMessage fault.

4354 To allow eventing filter expressions to be defined independently of the delivery mode,
 4355 WS-Management defines a new filter dialect that is the same as previously defined except that the
 4356 context node is defined as the element that would be returned as the first child of the SOAP Body
 4357 element if the Push delivery mode were used. The URI for this filter dialect is:

4358 <http://schemas.dmtf.org/wbem/wsman/1/wsman/filter/eventRootXPath>

4359 The context node for this expression is as follows:

- 4360 • **Context Node:** any XML element that could be returned as a direct child of the s:Body
 4361 element if the delivery mode was Push
- 4362 • **Context Position:** 1
- 4363 • **Context Size:** 1
- 4364 • **Variable Bindings:** none
- 4365 • **Function Libraries:** Core Function Library [\[XPath 1.0\]](#)

- **Namespace Declarations:** the [in-scope namespaces] property [\[XML Infoset\]](#) of /s:Envelope/s:Body/wsme:Subscribe/wsman:Filter

R10.2.2-4: Services should support this filter dialect when they want to use an XPath-based filter, rather than the default filter dialect defined in 10.2.1.

The considerations described in 8.3 regarding the [XPath 1.0](#) filter dialect also apply to the preceding eventing filter.

Resource-constrained implementations might have difficulty providing full XPath processing and yet still want to use a subset of XPath syntax. This does not require the addition of a new dialect if the expression specified in the filter is a true XPath expression. The use of the filter dialect URI does not imply that the service supports the entire specification for that dialect, only that the expression conforms to the rules of that dialect. Most services use XPath only for filtering, but they will not support the composition of new XML or removing portions of XML that would result in the XML fragment violating the schema of the event.

EXAMPLE 1: A typical example of the use of XPath in a subscription follows. Assume that each event that would be delivered has the following XML content:

```
(1) <s:Body>
(2)   <LowDiskSpaceEvent xmlns="...">
(3)     <LogicalDisk>C:</LogicalDisk>
(4)     <CurrentMegabytes>12</CurrentMegabytes>
(5)     <Megabytes24HoursAgo>17</Megabytes24HoursAgo>
(6)   </LowDiskSpaceEvent>
(7) </s:Body>
```

The event is wholly contained within the s:Body of the SOAP message. The anchor point for the XPath evaluation is the first element of each event, and it does not reference the <s:Body> element as such. The XPath expression is evaluated as if the event content were a separate XML document.

EXAMPLE 2: When used for simple document processing, the following four XPath expressions "select" the entire <LowDiskSpaceEvent> node:

```
(8) /
(9) /LowDiskSpaceEvent
(10) ../LowDiskSpaceEvent
(11) .
```

If used as a "filter", this XPath expression does not filter out any instances and is the same as selecting all instances of the event, or omitting the filter entirely.

EXAMPLE 3: However, using the following syntax, the XPath expression selects the XML node only if the test expression in brackets evaluates to logical "true":

```
(1) ../LowDiskSpaceEvent[LogicalDisk="C:"]
```

In this case, the event is selected if it refers to disk drive "C:"; otherwise the XML node is not selected. This XPath expression would filter out all <LowDiskSpaceEvent> events for other drives.

EXAMPLE 4: Full XPath implementations may support more complex test expressions:

```
(1) ../LowDiskSpaceEvent[LogicalDisk="C:" and CurrentMegabytes < "20"]
```

In essence, the XML form of the event is logically passed through the XPath processor to see if it would be selected. If so, it is delivered as an event. If not, the event is discarded and not delivered to the subscriber.

4409 [XPath 1.0](#) can be used simply for filtering or to send back subsets of the representation (or even the
 4410 values without XML wrappers). In cases where the result is not just filtered but is "altered," the
 4411 technique in 8.6 applies.

4412 If full XPath cannot be supported, a common subset for this purpose is described in ANNEX D of this
 4413 specification.

4414 **R10.2.2-5:** The wsman:Filter element shall contain either simple text or a single XML element
 4415 of a single or complex type. A service should reject any filter with mixed content or multiple peer
 4416 XML elements using a wsme:EventSourceUnableToProcess fault.

4417 **R10.2.2-6:** A conformant service may not support the entire syntax and processing power of
 4418 the specified filter dialect. The only requirement is that the specified filter is syntactically correct
 4419 within the definition of the dialect. Subsets are therefore legal. If the specified filter exceeds the
 4420 capability of the service, the service should return a wsman:CannotProcessFilter fault with text
 4421 explaining why the filter was problematic.

4422 **R10.2.2-7:** If a service requires complex initialization parameters in addition to the filter, these
 4423 should be part of the wsman:Filter block because they logically form part of the filter initialization,
 4424 even if some of the parameters are not strictly used in the filtering process. In this case, a unique
 4425 dialect URI shall be devised for the event source and the schema and usage published.

4426 **R10.2.2-8:** If the service supports composition of new XML or filtering to the point where the
 4427 resultant event would not conform to the original schema for that event, the event delivery should
 4428 be wrapped in the same way as content for the fragment-level access operations (see 7.7).

4429 Events, regardless of how they are filtered or reduced, need to conform to some kind of XML schema
 4430 definition when they are actually delivered. Simply sending out unwrapped XML fragments during
 4431 delivery is not legal.

4432 **R10.2.2-9:** If the service requires specific initialization XML in addition to the filter to formulate
 4433 a subscription, this initialization XML shall form part of the filter body and be documented as part
 4434 of the filter dialect.

4435 This rule promotes a consistent location for initialization content, which may be logically seen as part
 4436 of the filter. The filter XML schema is more understandable if it separates the initialization and filtering
 4437 parts into separate XML elements.

4438 For information about filtering over enumerations, see 8.3.

4439 10.2.3 Connection Retries

4440 Due to the nature of event delivery, the subscriber might not be reachable at event-time. Rather than
 4441 terminate all subscriptions immediately, typically the service attempts to connect several times with
 4442 suitable timeouts before giving up.

4443 **R10.2.3-1:** A service may observe any connection retry policy or allow the subscriber to define
 4444 it by including the following wsman:ConnectionRetry element in a subscription. If the service does
 4445 not accept the wsman:ConnectionRetry element, it should return a wsman:UnsupportedFeature
 4446 fault with the following detail code:

4447 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DeliveryRetries`

4448 This only applies to failures to *connect* and does not include replay of actual SOAP deliveries.

```
4449 (1) <wsme:Subscribe>
4450 (2)   <wsme:Delivery>
4451 (3)     <wsme:NotifyTo> ... </wsme:NotifyTo>
4452 (4)     <wsman:ConnectionRetry Total="count"> xs:duration
```

```

4453     </wsman:ConnectionRetry>
4454 (5)    </wsme:Delivery>
4455 (6)    </wsme:Subscribe>

```

4456 The following definitions provide additional, normative constraints on the preceding outline:

4457 wsman:ConnectionRetry

4458 an xs:duration for how long to wait between retries while trying to connect

4459 wsman:ConnectionRetry/@Total

4460 how many retries to attempt, observing the specified interval between the attempts

4461 **R10.2.3-2:** If the retry counts are exhausted, the subscription should be considered abnormally
 4462 terminated.

4463 The retry mechanism applies only to attempts to connect. Failures to deliver on an established
 4464 connection can result in terminating the connection according to the rules of the transport in use, and
 4465 terminating the subscription. Other Web services mechanisms can be used to synthesize reliable
 4466 delivery or safe replay of the actual deliveries.

4467 10.2.4 SubscribeResponse

4468 The service returns any service-specific reference parameters in the SubscriptionManager EPR, and
 4469 these are included by the subscriber (client) later when issuing Unsubscribe and Renew messages.

4470 **R10.2.4-1:** In SubscribeResponse, the service may specify any EPR for the
 4471 SubscriptionManager. However, it is recommended that the address contain the same wsa:To
 4472 address as the original Subscribe request and differ only in other parts of the address, such as
 4473 the reference parameters.

4474 **R10.2.4-2:** A conformant service may not return the Expires field in the response, but, as
 4475 specified in 10.2, this implies that the subscription does not expire until explicitly canceled.

4476 10.2.5 Heartbeats

4477 A typical problem with event subscriptions is a situation in which no event traffic occurs. It is difficult
 4478 for clients to know whether no events matching the subscription have occurred or whether the
 4479 subscription has simply failed and the client was not able to receive any notification.

4480 Because of this, WS-Management defines a "heartbeat" pseudo-event that can be sent periodically
 4481 for any subscription. This event is sent if no regular events occur so that the client knows the
 4482 subscription is still active. If the heartbeat event does not arrive, the client knows that connectivity is
 4483 bad or that the subscription has expired, and it can take corrective action.

4484 The heartbeat event is sent *in place of* the events that would have occurred and is *never* intermixed
 4485 with "real" events. In all modes, including batched, it occurs alone.

4486 To request heartbeat events as part of a subscription, the Subscribe request has an additional field in
 4487 the Delivery section:

```

4488 (1)    <wsme:Delivery>
4489 (2)    ...
4490 (3)    <wsman:Heartbeats> xs:duration </wsman:Heartbeats>
4491 (4)    ...
4492 (5)    </wsme:Delivery>

```

4493 wsman:Heartbeats specifies that heartbeat events are added to the event stream at the specified
4494 interval.

4495 **R10.2.5-1:** A service should support heartbeat events. If the service does not support them, it
4496 shall return a wsman:UnsupportedFeature fault with the following detail code:

4497 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats`

4498 Heartbeats apply to all delivery modes.

4499 Heartbeats apply to "pull" mode deliveries as well, in that they are a hint to the publisher about how
4500 often to expect a Pull request. The service can refuse to deliver events if the client does not regularly
4501 call back at the heartbeat interval. If no events are available at the heartbeat interval, the service
4502 simply includes a heartbeat event as the result of the Pull.

4503 **R10.2.5-2:** While a subscription with heartbeats is active, the service shall ensure that either
4504 real events or heartbeats are sent out within the specified wsman:Heartbeat interval. The service
4505 may send out heartbeats at this interval in addition to the events, as long as the heartbeat events
4506 are sent separately (not batched with other events). The goal is to ensure that some kind of event
4507 traffic always occurs within the heartbeat interval.

4508 **R10.2.5-3:** A conformant service may send out heartbeats at earlier intervals than specified in
4509 the subscription. However, the events should not be intermixed with other events when batching
4510 delivery modes are used. Typically, heartbeats are sent out *only when no real events occur*. A
4511 service may fail to produce heartbeats at the specified interval if real events have been delivered.

4512 **R10.2.5-4:** A conformant service shall not send out heartbeats asynchronously to any event
4513 deliveries already in progress. They shall be delivered in sequence like any other events,
4514 although they are delivered alone as single events or as the only event in a batch.

4515 In practice, heartbeat events are based on a countdown timer. If no events occur, the heartbeat is
4516 sent out alone. However, every time a real event is delivered, the heartbeat countdown timer is reset.
4517 If a steady stream of events occurs, heartbeats might never be delivered.

4518 Heartbeats need to be acknowledged like any other event if one of the acknowledged delivery modes
4519 is in effect.

4520 The client assumes that the subscription is no longer active if no heartbeats are received within the
4521 specified interval, so the service can proceed to cancel the subscription and send any requested
4522 SubscriptionEnd messages, because the client will likely resubscribe shortly. Used in combination
4523 with bookmarks (see 10.2.6), heartbeats can achieve highly reliable delivery with known latency
4524 behavior.

4525 The heartbeat event itself is simply an event message with no body and is identified by its wsa:Action
4526 URI as follows:

```
4527 (1) <s:Envelope ...>
4528 (2)   <s:Header>
4529 (3)     <wsa:To> .... </wsa:To>
4530 (4)     <wsa:Action s:mustUnderstand="true">
4531 (5)       http://schemas.dmtf.org/wbem/wsman/1/wsman/Heartbeat
4532 (6)     </wsa:Action>
4533 (7)     ...
4534 (8)   </s:Header>
4535 (9)   <s:Body/>
4536 (10) </s:Envelope>
```

4537 **10.2.6 Bookmarks**

4538 Reliable delivery of events is difficult to achieve, so management subscribers need to have a way to
 4539 be certain of receiving all events from a source. When subscriptions expire or when deliveries fail,
 4540 windows of time can occur in which the client cannot be certain whether critical events have occurred.
 4541 Rather than using a highly complex, transacted delivery model, WS-Management defines a simple
 4542 mechanism for ensuring that all events are delivered or that dropped events can be detected.

4543 This mechanism requires event sources to be backed by logs, whether short-term or long-term. The
 4544 client subscribes in the same way as a normal Subscribe operation, and specifies that bookmarks are
 4545 to be used. The service then sends a new bookmark with each event delivery, which the client is
 4546 responsible for persisting. This bookmark is essentially a context or a pointer to the logical event
 4547 stream location that matches the subscription filter. As each new delivery occurs, the client updates
 4548 the bookmark in its own space. If the subscription expires or is terminated unexpectedly, the client
 4549 can subscribe again, using the last known bookmark. In essence, the subscription filter identifies the
 4550 desired set of events, and the bookmark tells the service where to start in the log. The client may then
 4551 pick up where it left off.

4552 This mechanism is immune to transaction problems, because the client can simply start from any of
 4553 several recent bookmarks. The only requirement for the service is to have some type of persistent log
 4554 in which to apply the bookmark. If the submitted bookmark is too old (temporally or positionally within
 4555 the log), the service can fault the request, and at least the client reliably knows that events have been
 4556 dropped.

4557 **R10.2.6-1:** A conformant service may support the WS-Management bookmark mechanism. If
 4558 the service does not support bookmarks, it should return a wsman:UnsupportedFeature fault with
 4559 the following detail code:

4560 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Bookmarks`

4561 To request bookmark services, the client includes the wsman:SendBookmarks element in the
 4562 Subscribe request as follows:

```

4563 (1) <s:Body>
4564 (2)   <wsme:Subscribe>
4565 (3)     <wsme:Delivery>
4566 (4)       ...
4567 (5)     </wsme:Delivery>
4568 (6)     <wsman:SendBookmarks/>
4569 (7)   </wsme:Subscribe>
4570 (8) </s:Body>
```

4571 wsman:SendBookmarks instructs the service to send a bookmark with each event delivery.
 4572 Bookmarks apply to all delivery modes.

4573 The bookmark is a token that represents an abstract pointer in the event stream, but whether it points
 4574 to the last delivered event or the last event plus one (the upcoming event) makes no difference
 4575 because the token is supplied to the same implementation during a subsequent Subscribe operation.
 4576 The service can thus attach any service-specific meaning and structure to the bookmark with no
 4577 change to the client.

4578 If bookmarks are requested, each event delivery contains a new bookmark value as a SOAP header,
 4579 as shown in the following outline. The format of the bookmark is entirely determined by the service
 4580 and is treated as an opaque value by the client.

```

4581 (1) <s:Envelope
4582 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
4583 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
4584 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
```

```

4585 (5)      <s:Header>
4586 (6)      <wsa:To s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
4587 (7)      ...
4588 (8)      <wsman:Bookmark> xs:any </wsman:Bookmark>
4589 (9)      ...
4590 (10)     </s:Header>
4591 (11)     <s:Body>
4592 (12)     ...event content...
4593 (13)     </s:Body>
4594 (14)    </s:Envelope>

```

4595 wsman:Bookmark contains XML content supplied by the service that indicates the logical position of
 4596 this event or event batch in the event stream implied by the subscription.

4597 **R10.2.6-2:** If bookmarks are supported, the wsman:Bookmark element content shall be either
 4598 simple text or a single complex XML element. A conformant service shall not accept mixed
 4599 content of both text and elements, or multiple peer XML elements, under the wsman:Bookmark
 4600 element.

4601 **R10.2.6-3:** If bookmarks are supported, the service shall use a wsman:Bookmark element in
 4602 the header to send an updated bookmark with each event delivery. Bookmarks accompany only
 4603 event deliveries and are not part of any SubscriptionEnd message.

4604 After the subscription has terminated, for whatever reason, a subsequent Subscribe message on the
 4605 part of the client can include the bookmark in the subscription request. The service then knows where
 4606 to start.

4607 The last-known bookmark received by the client is added to the Subscribe message as a new block,
 4608 positioned after the child elements of Subscribe, as in the following outline:

```

4609 (1) <s:Body>
4610 (2) <wsme:Subscribe>
4611 (3) <wsme:Delivery> ... </wsme:Delivery>
4612 (4) <wsme:Expires> ... </wsme:Expires>
4613 (5) <wsman:Filter> ... </wsman:Filter>
4614 (6) <wsman:Bookmark>
4615 (7) ...last known bookmark from a previous delivery...
4616 (8) </wsman:Bookmark>
4617 (9) <wsman:SendBookmarks/>
4618 (10) </wsme:Subscribe>
4619 (11) </s:Body>

```

4620 The following definitions provide additional, normative constraints on the preceding outline:

4621 wsman:Bookmark
 4622 arbitrary XML content previously supplied by the service as a wsman:Bookmark during event
 4623 deliveries from a previous subscription

4624 wsman:SendBookmarks
 4625 an instruction to continue delivering updated bookmarks with each event delivery

4626 **R10.2.6-4:** The bookmark is a pointer to the last event delivery or batched delivery. The service
 4627 shall resume delivery at the first event or events after the event represented by the bookmark.
 4628 The service shall not replay events associated with the bookmark or skip any events since the
 4629 bookmark.

4630 **R10.2.6-5:** The service may support a short queue of previous bookmarks, allowing the
 4631 subscriber to start using any of several previous bookmarks. If bookmarks are supported, the

4632 service is required only to support the most recent bookmark for which delivery had apparently
 4633 succeeded.

4634 **R10.2.6-6:** If the bookmark cannot be honored, the service shall fault with a
 4635 wsman:InvalidBookmark fault with one of the following detail codes:

4636 • bookmark has expired (the source is not able to back up and replay from that point):

4637 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired>

4638 • format is unknown:

4639 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFormat>

4640 If multiple new subscriptions are made using a previous bookmark, the service can allow multiple
 4641 reuse or may limit bookmarks to a single subscriber, and can even restrict how long bookmarks can
 4642 be used before becoming invalid.

4643 The following predefined, reserved bookmark value indicates that the subscription starts at the
 4644 earliest possible point in the event stream backed by the publisher:

4645 <http://schemas.dmtf.org/wbem/wsman/1/wsman/bookmark/earliest>

4646 If a subscription is received with this bookmark, the event source replays all possible events that
 4647 match the filter and any events that subsequently occur for that event source. The absence of any
 4648 bookmark means "begin at the next available event".

4649 **R10.2.6-7:** A conformant service may support the reserved bookmark
 4650 <http://schemas.dmtf.org/wbem/wsman/1/wsman/bookmark/earliest> and not support any other type
 4651 of bookmark. If the <http://schemas.dmtf.org/wbem/wsman/1/wsman/bookmark/earliest> bookmark
 4652 is supported, the event source should send all previous and future events that match the filter
 4653 starting with the earliest such event.

4654 10.2.7 Delivery Modes

4655 While the general pattern of asynchronous, event-based messages is extremely common, different
 4656 applications often require different event message delivery mechanisms. For instance, in some cases
 4657 a simple asynchronous message is optimal, while other situations may work better if the event
 4658 consumer can poll for event messages in order to control the flow and timing of message arrival.
 4659 Some consumers require event messages to be wrapped in a standard "event" SOAP envelope,
 4660 while others prefer messages to be delivered unwrapped. Some consumers may require event
 4661 messages to be delivered reliably, while others may be willing to accept best-effort event delivery.

4662 In order to support this broad variety of event delivery requirements, this specification introduces an
 4663 abstraction called a Delivery Mode. This concept is used as an extension point, so that event sources
 4664 and event consumers may freely create new delivery mechanisms that are tailored to their specific
 4665 requirements. This specification provides a minimal amount of support for delivery mode negotiation
 4666 by allowing an event source to provide a list of supported delivery modes in response to a
 4667 subscription request specifying a delivery mode it does not support.

4668 A WS-Management implementation can support a variety of event delivery modes.

4669 In essence, delivery consists of the following items:

- 4670 • a delivery mode (how events are packaged)
- 4671 • an address (the transport and network location)
- 4672 • an authentication profile to use when connecting or delivering the events (security)

4673 The standard security profiles are discussed in clause 12 and may be required for subscriptions if the
4674 service needs hints or other indications of which security model to use at event-time.

4675 If the delivery mode is supported but not actually usable due to firewall configuration, the service can
4676 return a wsme:DeliveryModeRequestedUnavailable fault with additional detail to this effect.

4677 **R10.2.7-1:** For any given transport, a conformant service should support at least one of the
4678 following delivery modes to interoperate with standard clients:

4679 <http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push>

4680 <http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck>

4681 <http://schemas.dmtf.org/wbem/wsman/1/wsman/Events>

4682 <http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull>

4683 The delivery mode does *not* imply any specific transport.

4684 Modes describe SOAP message behavior and are unrelated to the transport that is in use. A delivery
4685 mode implies a specific SOAP message format, so a message that deviates from that format requires
4686 a new delivery mode.

4687 **R10.2.7-2:** The NotifyTo address in the Subscribe message shall support only a single delivery
4688 mode.

4689 This requirement is for the client because the service cannot verify whether this statement is true. If
4690 this requirement is not observed by the client, the service might not operate correctly. If the
4691 subscriber supports multiple delivery modes, the NotifyTo address needs to be differentiated in some
4692 way, such as by adding an additional reference parameter.

4693 10.2.8 Event Action URI

4694 Typically, each event type has its own wsa:Action URI to quickly identify and route the event. If an
4695 event type does not define its own wsa:Action URI, the following URI can be used as a default:

4696 <http://schemas.dmtf.org/wbem/wsman/1/wsman/Event>

4697 This URI can be used in cases where event types are inferred in real-time from other sources and not
4698 published as Web service events, and thus do not have a designated wsa:Action URI. This
4699 specification places no restrictions on the wsa:Action URI for events. More specific URIs can act as a
4700 reliable dispatching point. In many cases, a fixed schema can serve to model many different types of
4701 events, in which case the event "ID" is simply a field in the XML content of the event. The URI in this
4702 case might reflect the schema and be undifferentiated for all of the various event IDs that might occur
4703 or it might reflect the specific event by suffixing the event ID to the wsa:Action URI. This specification
4704 places no restrictions on the granularity of the URI, but careful consideration of these issues is part of
4705 designing the URIs for events.

4706 10.2.9 Delivery Sequencing and Acknowledgement

4707 The delivery mode indicates how the service will exchange events with interested parties. This clause
4708 describes delivery modes in detail.

4709 **10.2.9.1 General**

4710 For some event types, ordered and acknowledged delivery is important, but for other types of events
 4711 the order of arrival is not significant. WS-Management defines four standard delivery modes:

- 4712 • <http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push>
 4713 With this mode, each SOAP message has only one event and no acknowledgement or
 4714 SOAP response. The service can deliver events for the subscription asynchronously without
 4715 regard to any events already in transit. This mode is useful when the order of events does
 4716 not matter, such as with events containing running totals in which each new event can
 4717 replace the previous one completely and the time stamp is sufficient for identifying the most
 4718 recent event.
- 4719 • <http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck>
 4720 With this mode, each SOAP message has only one event, but each event is acknowledged
 4721 before another is sent. The service queues all undelivered events for the subscription and
 4722 delivers each new event only after the previous one has been acknowledged.
- 4723 • <http://schemas.dmtf.org/wbem/wsman/1/wsman/Events>
 4724 With this mode, each SOAP message can have many events, but each batch is
 4725 acknowledged before another is sent. The service queues all events for the subscription
 4726 and delivers them in that order, maintaining the order in the batches.
- 4727 • <http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull>
 4728 With this mode, each SOAP message can have many events, but each batch is
 4729 acknowledged. Because the receiver uses Pull to synchronously retrieve the events,
 4730 acknowledgement is implicit. The order of delivery is maintained.

4731 Ordering of events across subscriptions is not implied.

4732 The acknowledgement model is discussed in 10.8.

4733 **10.2.9.2 Push Mode**

4734 The standard delivery mode is
 4735 <http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push>, in which each delivery
 4736 consists of a single event. No acknowledgement occurs, so the delivery cannot be faulted to cancel
 4737 the subscription.

4738 Therefore, subscriptions made with this delivery mode can have short durations to prevent a situation
 4739 in which deliveries cannot be stopped if the SubscriptionManager content from the
 4740 SubscribeResponse information is corrupted or lost.

4741 To promote fast routing of events, the required wsa:Action URI in each event message can be distinct
 4742 for each event type, regardless of how strongly typed the event body is.

4743 **R10.2.9.2-1:** A service may support the
 4744 <http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push> delivery mode.

4745 **R10.2.9.2-2:** To precisely control how to deal with events that are too large, the service may
 4746 accept the following additional instruction in a subscription:

```

4747 (1) <wsme:Delivery>
4748 (2)   <wsme:NotifyTo> ... </wsme:NotifyTo>
4749 (3)   ...
4750 (4)   <wsman:MaxEnvelopeSize Policy="enumConstant">
4751 (5)     xs:positiveInteger
  
```

```

4752 (6)    </wsman:MaxEnvelopeSize>
4753 (7)    ...
4754 (8)    </wsme:Delivery>

```

4755 The following definitions provide additional, normative constraints on the preceding outline:

4756 wsme:Delivery/wsman:MaxEnvelopeSize

4757 the maximum number of octets for the entire SOAP envelope in a single event delivery

4758 wsme:Delivery/wsman:MaxEnvelopeSize/@Policy

4759 an optional value with one of the following enumeration values:

- 4760 • **CancelSubscription:** cancel on the first oversized event
- 4761 • **Skip:** silently skip oversized events
- 4762 • **Notify:** notify the subscriber that events were dropped as specified in 10.9

4763 **R10.2.9.2-3:** If wsman:MaxEnvelopeSize is requested, the service shall not send an event
 4764 body larger than the specified limit. The default behavior is to notify the subscriber as specified in
 4765 10.9, unless otherwise instructed in the subscription, and to attempt to continue delivery. If the
 4766 event exceeds any internal default maximums, the service should also attempt to notify as
 4767 specified in 10.9 rather than terminate the subscription, unless otherwise specified in the
 4768 subscription. If wsman:MaxEnvelopeSize is too large for the service, the service shall return a
 4769 wsman:EncodingLimit fault with the following detail code:

4770 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize>

4771 In the absence of any other Policy instructions, services are to deliver notifications of dropped events
 4772 to subscribers, as specified in 10.9.

4773 10.2.9.3 PushWithAck Mode

4774 This delivery mode is identical to the standard "Push" mode except that each delivery is
 4775 acknowledged. Each delivery still has one event, and the wsa:Action element indicates the event
 4776 type. However, a SOAP-based acknowledgement occurs as described in 10.7.

4777 The delivery mode URI is:

4778 <http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck>

4779 In every other respect except the delivery mode URI, this mode is identical to Push mode as
 4780 described in 10.2.9.2.

4781 **R10.2.9.3-1:** A service should support the
 4782 <http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck> delivery mode. If the delivery mode
 4783 is not supported, the service should return a fault of wsme:DeliveryModeRequestedUnavailable.

4784 10.2.9.4 Batched Delivery Mode

4785 Batching events is an effective way to minimize event traffic from a high-volume event source without
 4786 sacrificing event timeliness. WS-Management defines a custom event delivery mode that allows an
 4787 event source to bundle multiple outgoing event messages into a single SOAP envelope. Delivery is
 4788 always acknowledged, using the model defined in 10.7.

4789 **R10.2.9.4-1:** A service may support the <http://schemas.dmtf.org/wbem/wsman/1/wsman/Events>
 4790 delivery mode. If the delivery mode is not supported, the service should return a fault of
 4791 wsme:DeliveryModeRequestedUnavailable.

4792 For this delivery mode, the Delivery element has the following format:

```

4793 (1) <wsme:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
4794 (2)   <wsme:NotifyTo>
4795 (3)     wsa:EndpointReferenceType
4796 (4)   </wsme:NotifyTo>
4797 (5)   <wsman:MaxElements> xs:positiveInteger </wsman:MaxElements> ?
4798 (6)   <wsman:MaxTime> xs:duration </wsman:MaxTime> ?
4799 (7)   <wsman:MaxEnvelopeSize Policy="enumConstant">
4800 (8)     xs:positiveInteger
4801 (9)   </wsman:MaxEnvelopeSize> ?
4802 (10) </wsme:Delivery>

```

4803 The following definitions provide additional, normative constraints on the preceding outline:

4804 wsme:Delivery/@Mode

4805 required attribute that shall be defined as

4806 http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

4807 wsme:Delivery/wsme:NotifyTo

4808 required element that shall contain the EPR to which event messages are to be sent for this
4809 subscription

4810 wsme:Delivery/wsman:MaxElements

4811 optional element that contains a positive integer that indicates the maximum number of event
4812 bodies to batch into a single SOAP envelope

4813 The resource shall not deliver more than this number of items in a single delivery, although it
4814 may deliver fewer.

4815 wsme:Delivery/wsman:MaxEnvelopeSize

4816 optional element that contains a positive integer that indicates the maximum number of octets in
4817 the SOAP envelope used to deliver the events

4818 wsman:MaxEnvelopeSize/@Policy

4819 an optional attribute with one of the following enumeration values:

- 4820 • **CancelSubscription:** cancel on the first oversized event
- 4821 • **Skip:** silently skip oversized events
- 4822 • **Notify:** notify the subscriber that events were dropped as specified in 10.9

4823 wsme:Delivery/wsman:MaxTime

4824 optional element that contains a duration that indicates the maximum amount of time the service
4825 should allow to elapse while batching Event bodies

4826 This time may not be exceeded between the encoding of the first event in the batch and the
4827 dispatching of the batch for delivery. Some publisher implementations may choose more
4828 complex schemes in which different events included in the subscription are delivered at different
4829 latencies or at different priorities. In such cases, a specific filter dialect can be designed for the
4830 purpose and used to describe the instructions to the publisher. In such cases, wsman:MaxTime
4831 can be omitted if it is not applicable; if present, however, it serves as an override of anything
4832 defined within the filter.

4833 In the absence of any other instructions in any part of the subscription, services are to deliver
4834 notifications of dropped events to subscribers, as specified in 10.9.

4835 If a client wants to discover the appropriate values for wsman:MaxElements or
4836 wsman:MaxEnvelopeSize, the client can query for service-specific metadata. The format of such
4837 metadata is beyond the scope of this particular specification.

4838 **R10.2.9.4-2:** If batched mode is requested in a Subscribe message, and MaxElements,
4839 MaxEnvelopeSize, and MaxTime elements are not present, the service may pick any applicable
4840 defaults. The following faults apply:

- 4841 • If MaxElements is not supported, wsman:UnsupportedFeature is returned with the following
4842 fault detail code:

4843 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements`

- 4844 • If MaxEnvelopeSize is not supported, wsman:UnsupportedFeature is returned with the
4845 following fault detail code:

4846 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize`

- 4847 • If MaxTime is not supported, wsman:UnsupportedFeature is returned with the following fault
4848 detail code:

4849 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime`

- 4850 • If MaxEnvelopeSize/@Policy is not supported, wsman:UnsupportedFeature is returned with
4851 the following fault detail code:

4852 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy`

4853 **R10.2.9.4-3:** If wsman:MaxEnvelopeSize is requested, the service shall not send an event
4854 body larger than the specified limit. The default behavior is to notify the subscriber as specified in
4855 10.9, unless otherwise instructed in the subscription, and to attempt to continue delivery. If the
4856 event exceeds any internal default maximums, the service should also attempt notification as
4857 specified in 10.9 rather than terminate the subscription, unless otherwise specified in the
4858 subscription.

4859 If a subscription has been created using batched mode, all event delivery messages shall have
4860 the following format:

```

4861 (1) <s:Envelope ...>
4862 (2)   <s:Header>
4863 (3)     ...
4864 (4)     <wsa:Action>
4865 (5)       http://schemas.dmtf.org/wbem/wsman/1/wsman/Events
4866 (6)     </wsa:Action>
4867 (7)     ...
4868 (8)   </s:Header>
4869 (9)   <s:Body>
4870 (10)    <wsman:Events>
4871 (11)      <wsman:Event Action="event action URI">
4872 (12)        ...event body...
4873 (13)      </wsman:Event> +
4874 (14)    </wsman:Events>
4875 (15)  </s:Body>
4876 (16) </s:Envelope>

```

4877 The following definitions provide additional, normative constraints on the preceding outline:

4878 s:Envelope/s:Header/wsa:Action

4879 required element that shall be defined as

4880 http://schemas.dmtf.org/wbem/wsman/1/wsman/Events

4881 s:Envelope/s:Body/wsman:Events/wsman:Event

4882 required elements that shall contain the body of the corresponding event message, as if
4883 wsman:Event were the s:Body element

4884 s:Envelope/s:Body/wsman:Events/wsman:Event/@Action

4885 required attribute that shall contain the wsa:Action URI that would have been used for the
4886 contained event message

4887 **R10.2.9.4-4:** If batched mode is requested, deliveries shall be acknowledged as described in
4888 10.7.

4889 Dropped events (as specified in 10.9) are encoded with any other events.

4890 EXAMPLE: The following example shows batching parameters supplied to a Subscribe operation. The
4891 service is instructed to send no more than 10 items per batch, to wait no more than 20 seconds from the
4892 time the first event is encoded until the entire batch is dispatched, and to include no more than 8192 octets
4893 in the SOAP message.

```
4894 (1) ...
4895 (2) <wsme:Delivery
4896 (3)   Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events">
4897 (4)   <wsme:NotifyTo>
4898 (5)     <wsa:Address>http://2.3.4.5/client</wsa:Address>
4899 (6)   </wsme:NotifyTo>
4900 (7)   <wsman:MaxElements>10</wsman:MaxElements>
4901 (8)   <wsman:MaxTime>PT20S</wsman:MaxTime>
4902 (9)   <wsman:MaxEnvelopeSize>8192</wsman:MaxEnvelopeSize>
4903 (10) </wsme:Delivery>
```

4904 EXAMPLE: Following is an example of batched delivery that conforms to this specification:

```
4905 (1) <s:Envelope
4906 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
4907 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
4908 (4)   xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"
4909 (5)   xmlns:wsme="http://schemas.xmlsoap.org/ws/2004/08/eventing">
4910 (6)   <s:Header>
4911 (7)     <wsa:To s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
4912 (8)     <wsa:Action>
4913 (9)       http://schemas.dmtf.org/wbem/wsman/1/wsman/Events
4914 (10)    </wsa:Action>
4915 (11)    ...
4916 (12)   </s:Header>
4917 (13)   <s:Body>
4918 (14)     <wsman:Events>
4919 (15)       <wsman:Event
4920 (16)         Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
4921 (17)         <DiskChange
4922 (18)           xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
4923 (19)             <Drive> C: </Drive>
4924 (20)             <FreeSpace> 802012911 </FreeSpace>
```

```

4925 (21)      </DiskChange>
4926 (22)      </wsman:Event>
4927 (23)      <wsman:Event
4928 (24)          Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
4929 (25)          <DiskChange
4930 (26)              xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
4931 (27)                  <Drive> D: </Drive>
4932 (28)                  <FreeSpace> 1402012913 </FreeSpace>
4933 (29)          </DiskChange>
4934 (30)      </wsman:Event>
4935 (31)      </wsman:Events>
4936 (32)      </s:Body>
4937 (33) </s:Envelope>

```

4938 The Action URI in line 9 specifies that this is a batch that contains distinct events. The individual
 4939 event bodies are at lines 15–22 and lines 23–30. The actual Action attribute for the individual events
 4940 is an attribute of the wsman:Event wrapper.

4941 10.2.9.5 Pull Delivery Mode

4942 In some circumstances, polling for events is an effective way of controlling data flow and balancing
 4943 timeliness against processing ability. Also, in some cases, network restrictions prevent "push" modes
 4944 from being used; that is, the service cannot initiate a connection to the subscriber.

4945 WS-Management defines a custom event delivery mode, "pull mode," which allows an event source
 4946 to maintain a logical queue of event messages received by enumeration. This delivery mode borrows
 4947 the Pull message to retrieve events from the logical queue. However, all of the other pub/sub
 4948 operations defined in this clause can continue to be used. (For example, Unsubscribe, rather than
 4949 Release, is used to cancel a subscription.)

4950 For this delivery mode, the Delivery element has the following format:

```

4951 (1) <wsme:Delivery Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull">
4952 (2) ...
4953 (3) </wsme:Delivery>

```

4954 wsme:Delivery/@Mode shall be

4955 <http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull>

4956 **R10.2.9.5-1:** A service may support the <http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull>
 4957 delivery mode. If pull mode is requested but not supported, the service shall return a fault of
 4958 wsme:DeliveryModeRequestedUnavailable.

4959 wsman:MaxElements, wsman:MaxEnvelopeSize, and wsman:MaxTime do not apply in the Subscribe
 4960 message when using this delivery mode because the Pull message contains all of the necessary
 4961 functionality for controlling the batching and timing of the responses.

4962 **R10.2.9.5-2:** If a subscription incorrectly specifies parameters that are not compatible with pull
 4963 mode, the service should issue a wsman:UnsupportedFeature fault with the following detail code:

4964 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch>

4965 **R10.2.9.5-3:** If pull mode is requested in a Subscribe message and the event source accepts
 4966 the subscription request, the SubscribeResponse element in the REPLY message shall contain
 4967 an EnumerationContext element suitable for use in a subsequent Pull operation.

4968 EXAMPLE:

```

4969 (1) <s:Body ...>
4970 (2)   <wsme:SubscribeResponse ...>
4971 (3)     <wsme:SubscriptionManager>
4972 (4)       wsa:EndpointReferenceType
4973 (5)     </wsme:SubscriptionManager>
4974 (6)     <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires>
4975 (7)     <wsmen:EnumerationContext>...</wsmen:EnumerationContext>
4976 (8)     ...
4977 (9)   </wsme:SubscribeResponse>
4978 (10) </s:Body>

```

4979 The subscriber extracts the EnumerationContext and uses it thereafter in Pull requests.

4980 **R10.2.9.5-4:** If pull mode is active, Pull messages shall use the EPR of the subscription
 4981 manager obtained from the SubscribeResponse message. The EPR reference parameters are of
 4982 a service-specific addressing model, but may use the WS-Management default addressing model
 4983 if it is suitable.

4984 **R10.2.9.5-5:** If pull mode is active and a Pull request returns no events (because none have
 4985 occurred since the last "pull"), the service should return a wsman:TimedOut fault. The
 4986 EnumerationContext is still considered active, and the subscriber may continue to issue Pull
 4987 requests with the most recent EnumerationContext for which event deliveries actually occurred.

4988 **R10.2.9.5-6:** If pull mode is active and a Pull request returns events, the service may return an
 4989 updated EnumerationContext as specified for Pull, and the subscriber is expected to use the
 4990 update, if any, in the subsequent Pull, as specified for the Enumeration operations. Bookmarks, if
 4991 active, may also be returned in the header and shall also be updated by the service.

4992 In practice, the service might not actually change the EnumerationContext, but the client cannot
 4993 depend on it remaining constant. It is updated conceptually, if not actually.

4994 In pull mode, the Pull request controls the batching. If no defaults are specified, the batch size is 1
 4995 and the maximum envelope size and timeouts are service-defined.

4996 **R10.2.9.5-7:** If pull mode is active, the service shall not return an EndOfSequence element in
 4997 the event stream because no concept of a "last event" exists in this mode. Rather, the
 4998 enumeration context should become invalid if the subscription expires or is canceled for any
 4999 reason.

5000 **R10.2.9.5-8:** If pull mode is used, the service shall accept the wsman:MaxEnvelopeSize used
 5001 in the Pull as the limitation on the event size that can be delivered.

5002 The batching properties used in batched mode do not apply to pull mode. The client controls the
 5003 maximum event size using the normal mechanisms in Pull.

5004 10.3 GetStatus

5005 To get the status of a subscription, the subscriber sends a request of the following form to the
 5006 subscription manager:

```

5007 (1) <s:Envelope ...>
5008 (2)   <s:Header ...>
5009 (3)     <wsa:Action>
5010 (4)       http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus
5011 (5)     </wsa:Action>
5012 (6)     ...
5013 (7)   </s:Header>
5014 (8)   <s:Body ...>

```

```

5015 (9)      <wsme:GetStatus ...>
5016 (10)     ...
5017 (11)    </wsme:GetStatus>
5018 (12)    </s:Body>
5019 (13)   </s:Envelope>

```

5020 Components of the preceding outline are additionally constrained as for a request to renew a
 5021 subscription. Other components of the preceding outline are not further constrained by this
 5022 specification.

5023 If the subscription is valid and has not expired, the subscription manager shall reply with a response
 5024 of the following form:

```

5025 (1) <s:Envelope ...>
5026 (2) <s:Header ...>
5027 (3) <wsa:Action>
5028 (4) http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse
5029 (5) </wsa:Action>
5030 (6) ...
5031 (7) </s:Header>
5032 (8) <s:Body ...>
5033 (9) <wsme:GetStatusResponse ...>
5034 (10) <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
5035 (11) ...
5036 (12) </wsme:GetStatusResponse>
5037 (13) </s:Body>
5038 (14) </s:Envelope>

```

5039 Components of the preceding outline are constrained as for a response to a renew request. Other
 5040 components of the preceding outline are not further constrained by this specification.

5041 The wsme:GetStatus message is optional for WS-Management.

5042 **R10.3-1:** The wse:GetStatus message in a constrained environment is a candidate for exclusion.
 5043 If this message is not supported, then a wsa:ActionNotSupported fault shall be returned in
 5044 response to this request.

5045 Heartbeat support may be implemented rather than the wsme:GetStatus message.

5046 10.4 Unsubscribe

5047 Though subscriptions expire eventually, to minimize resources the subscribing event sink should
 5048 explicitly delete a subscription when it no longer wants notifications associated with the subscription.

5049 To explicitly delete a subscription, a subscribing event sink sends a request of the following form to
 5050 the subscription manager:

```

5051 (1) <s:Envelope ...>
5052 (2) <s:Header ...>
5053 (3) <wsa:Action>
5054 (4) http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe
5055 (5) </wsa:Action>
5056 (6) ...
5057 (7) </s:Header>
5058 (8) <s:Body>
5059 (9) <wsme:Unsubscribe ...>
5060 (10) ...
5061 (11) </wsme:Unsubscribe>
5062 (12) </s:Body>
5063 (13) </s:Envelope>

```

5064 Components of the preceding outline are additionally constrained only as for a request to renew a
 5065 subscription. For example, the faults listed there are also defined for a request to delete a
 5066 subscription.

5067 If the subscription manager accepts a request to delete a subscription, it shall reply with a response
 5068 of the following form:

```

5069 (1) <s:Envelope ...>
5070 (2)   <s:Header ...>
5071 (3)     <wsa:Action>
5072 (4)       http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse
5073 (5)     </wsa:Action>
5074 (6)     <wsa:RelatesTo>xs:anyURI</wsa:RelatesTo>
5075 (7)     ...
5076 (8)   </s:Header>
5077 (9)   <s:Body />
5078 (10) </s:Envelope>
  
```

5079 Components of the preceding outline are not further constrained by this specification.

5080 **R10.4-1:** If a service supports Subscribe, it shall implement the Unsubscribe message and
 5081 ensure that event delivery will be terminated if the message is accepted as valid. Delivery of
 5082 events may occur after responding to the Unsubscribe message as long as the event traffic stops
 5083 at some point.

5084 **R10.4-2:** A service may unilaterally cancel a subscription for any reason, including internal
 5085 timeouts, reconfiguration, or unreliable connectivity.

5086 Clients need to be prepared to receive any events already in transit even though they have issued an
 5087 Unsubscribe message. Clients have the option to either fault any such deliveries or accept them.

5088 The EPR to use for this message is received from the SubscribeResponse element in the
 5089 SubscriptionManager element.

5090 10.5 Renew

5091 To update the expiration for a subscription, subscription managers shall support requests to renew
 5092 subscriptions.

5093 To renew a subscription, the subscriber sends a request of the following form to the subscription
 5094 manager:

```

5095 (1) <s:Envelope ...>
5096 (2)   <s:Header ...>
5097 (3)     <wsa:Action>
5098 (4)       http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew
5099 (5)     </wsa:Action>
5100 (6)     ...
5101 (7)   </s:Header>
5102 (8)   <s:Body ...>
5103 (9)     <wsme:Renew ...>
5104 (10)      <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
5105 (11)     ...
5106 (12)   </wsme:Renew>
5107 (13) </s:Body>
5108 (14) </s:Envelope>
  
```

5109 Components of the preceding outline are additionally constrained as for a request to create a
 5110 subscription. Other components of the preceding outline are not further constrained by this
 5111 specification.

5112 If the subscription manager accepts a request to renew a subscription, it shall reply with a response
5113 of the following form:

```

5114 (1) <s:Envelope ...>
5115 (2)   <s:Header ...>
5116 (3)     <wsa:Action>
5117 (4)       http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse
5118 (5)     </wsa:Action>
5119 (6)   ...
5120 (7) </s:Header>
5121 (8) <s:Body ...>
5122 (9)   <wsme:RenewResponse ...>
5123 (10)    <wsme:Expires>[xs:dateTime | xs:duration]</wsme:Expires> ?
5124 (11)   ...
5125 (12)  </wsme:RenewResponse>
5126 (13) </s:Body>
5127 (14) </s:Envelope>

```

5128 Components of the preceding outline are constrained as for a response to a subscribe request with
5129 the following addition(s):

5130 /s:Envelope/s:Body/*wsme:Expires

5131 If the requested expiration is a duration, then the implied start of that duration is the time when
5132 the subscription manager starts processing the Renew request.

5133 If the subscription manager chooses not to renew this subscription, the request shall fail, and the
5134 subscription manager may generate a wsme:UnableToRenew fault indicating that the renewal was
5135 not accepted.

5136 Other components of the preceding outline are not further constrained by this specification.

5137 Processing of the Renew message is required, but it is not required to succeed.

5138 **R10.5-1:** Although a conformant service shall accept the Renew message as a valid action, the
5139 service may always fault the request with a wsme:UnableToRenew fault, forcing the client to
5140 subscribe from scratch.

5141 Renew has no effect on deliveries in progress, bookmarks, heartbeats, or other ongoing activity. It
5142 simply extends the lifetime of the subscription.

5143 The EPR to use for this message is received from the SubscribeResponse element in the
5144 SubscriptionManager element.

5145 10.6 SubscriptionEnd

5146 If the event source terminates a subscription unexpectedly, the event source should send a
5147 Subscription End SOAP message to the endpoint reference indicated when the subscription was
5148 created. The message shall be of the following form:

```

5149 (1) <s:Envelope ...>
5150 (2)   <s:Header ...>
5151 (3)     <wsa:Action>
5152 (4)       http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd
5153 (5)     </wsa:Action> ?
5154 (6)   ...
5155 (7) </s:Header>
5156 (8) <s:Body ...>
5157 (9)   <wsme:SubscriptionEnd ...>
5158 (10)    <wsme:SubscriptionManager>
5159 (11)      endpoint-reference

```

```

5160      (12)      </wsme:SubscriptionManager>
5161      (13)      <wsme:Status>
5162      (14)      [
5163      (15)      http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure |
5164      (16)      http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown |
5165      (17)      http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCancelling
5166      (18)      ]
5167      (19)      </wsme:Status>
5168      (20)      <wsme:Reason xml:lang="language identifier" >xs:string</wsme:Reason>
5169      (21)      ...
5170      (22)      </wsme:SubscriptionEnd>
5171      (23)      ...
5172      (24)      </s:Body>
5173      (25) </s:Envelope>

```

5175 The following describes additional, normative constraints on the preceding outline:

5176 /s:Envelope/s:Body/*wsme:SubscriptionManager

5177 Endpoint reference of the subscription manager. It is recommended that event sinks ignore this
5178 element as its usage requires the ability to compare EPRs for equality when no such mechanism
5179 exists. Event sinks are advised to use reference parameters in the
5180 /wsme:Subscribe/wsme:EndTo EPR if they wish to correlate this message against their
5181 outstanding subscriptions.

5182 /s:Envelope/s:Body/wsme:SubscriptionEnd/wsme:Status =
5183 "http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure"

5184 This value shall be used if the event source terminated the subscription because of problems
5185 delivering notifications.

5186 /s:Envelope/s:Body/wsme:SubscriptionEnd/wsme:Status =
5187 "http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown"

5188 This value shall be used if the event source terminated the subscription because the source is
5189 being shut down in a controlled manner (that is, if the event source is being shut down but has
5190 the opportunity to send a SubscriptionEnd message before it exits).

5191 /s:Envelope/s:Body/wsme:SubscriptionEnd/wsme:Status =
5192 "http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCancelling"

5193 This value shall be used if the event source terminated the subscription for some other reason
5194 before it expired.

5195 /s:Envelope/s:Body/wsme:SubscriptionEnd/wsme:Reason

5196 This optional element contains text, in the language specified by the @xml:lang attribute,
5197 describing the reason for the unexpected subscription termination.

5198 Other message information headers defined in 5.4 may be included in the message, according to the
5199 usage and semantics defined in 5.4.

5200 Other components of the preceding outline are not further constrained by this specification.

5201 This SubscriptionEnd message is optional for WS-Management. In effect, it is the "last event" for a
5202 subscription. Because its primary purpose is to warn a subscriber that a subscription has ended, it is
5203 not suitable for use with pull-mode delivery.

5204 **R10.6-1:** A conformant service may implement the SubscriptionEnd message.

5205 **R10.6-2:** A conformant service shall not implement the SubscriptionEnd message when event
5206 delivery is done using pull mode as defined in 10.2.9.4.

5207 **R10.6-3:** If SubscriptionEnd is supported, the message shall contain any reference parameters
5208 specified by the subscriber in the EndTo address in the original subscription.

5209 **R10.6-4:** This rule intentionally left blank.

5210 If the service delivers events over the same connection as the Subscribe operation, the client typically
5211 knows that a subscription has been terminated because the connection itself closes or terminates.

5212 When the delivery connection is distinct from the subscribe connection, a SubscriptionEnd message
5213 is highly recommended; otherwise, the client has no immediate way of knowing that a subscription is
5214 no longer active.

5215 **10.7 Acknowledgement of Delivery**

5216 To ensure that delivery is acknowledged at the application level, the original subscriber can request
5217 that the event sink physically acknowledge event deliveries, rather than relying entirely on transport-
5218 level guarantees.

5219 In other words, the transport might have accepted delivery of the events but not forwarded them to
5220 the actual event sink process, and the service would move on to the next set of events. System
5221 failures might result in dropped events. Therefore, a mechanism is needed in which a message-level
5222 acknowledgement can occur. This allows acknowledgement to be pushed up to the application level,
5223 increasing the reliability of event deliveries.

5224 The client selects acknowledged delivery by selecting a delivery mode in which each event has a
5225 response. In this specification, the two acknowledged delivery modes are

- 5226 • <http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck>
- 5227 • <http://schemas.dmtf.org/wbem/wsman/1/wsman/Events>

5228 **R10.7-1:** A conformant service may support the PushWithAck or Events delivery mode.
5229 However, if either of these delivery modes is requested, to maintain an ordered queue of events,
5230 the service shall wait for the acknowledgement from the client before delivering the next event or
5231 events that match the subscription.

5232 **R10.7-2:** If an acknowledged delivery mode is selected for the subscription, the service shall
5233 include the following SOAP headers in each event delivery:

```
5234 (1) <s:Header>
5235 (2)   <wsa:ReplyTo> where to send the acknowledgement </wsa:ReplyTo>
5236 (3)   <wsman:AckRequested/>
5237 (4)   ...
5238 (5) </s:Header>
```

5239 The following definitions provide additional, normative constraints on the preceding outline:

5240 **wsa:ReplyTo**

5241 address that shall always be present in the event delivery as a consequence of the presence of
5242 wsman:AckRequested

5243 The client extracts this address and sends the acknowledgement to the specified EPR as
5244 required by Addressing.

5245 **wsman:AckRequested**

5246 no content; requires that the subscriber acknowledge all deliveries as described later in this
5247 clause

5248 The client then replies to the delivery with an acknowledgement or a fault.

5249 **R10.7-3:** A service may request receipt acknowledgement by using the wsman:AckRequested
 5250 block and subsequently expect an http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack message.
 5251 If this message is not received as a reply, the service may terminate the subscription.

5252 The acknowledgement message format returned by the event sink (receiver) to the event source is
 5253 identical for all delivery modes. As shown in the following outline, it contains a unique wsa:Action, and
 5254 the wsa:RelatesTo field is set to the MessageID of the event delivery to which it applies:

```

5255 (1) <s:Envelope ...>
5256 (2)   <s:Header>
5257 (3)     ...
5258 (4)     <wsa:To> endpoint reference from the event ReplyTo field </wsa:To>
5259 (5)     <wsa:Action> http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack
5260 (6)     </wsa:Action>
5261 (7)     <wsa:RelatesTo> message ID of original event delivery
5262 (8)     </wsa:RelatesTo>
5263 (9)     ...
5264 (10)  </s:Header>
5265 (11)  <s:Body/>
5266 (12)  </s:Envelope>
  
```

5267 The following definitions provide additional, normative constraints on the preceding outline:

5268 s:Envelope/s:Header/wsa:Action

5269 URI that shall be defined as

5270 http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack

5271 s:Envelope/s:Header/wsa:RelatesTo

5272 element that shall contain the wsa:MessageID of the event delivery to which it refers

5273 wsa:RelatesTo is the critical item that ensures that the correct delivery is being acknowledged,

5274 and thus it shall not be omitted.

5275 s:Envelope/s:Header/wsa:To

5276 EPR address extracted from the ReplyTo field in the event delivery

5277 All reference parameters shall be extracted and added to the SOAP header as well.

5278 In spite of the request to acknowledge, the event sink can refuse delivery with a fault or fail to
 5279 respond with the acknowledgement. In this case, the event source can terminate the subscription and
 5280 send any applicable SubscriptionEnd messages.

5281 If the event sink does not support acknowledgement, it can respond with a

5282 wsman:UnsupportedFeature fault with the following detail code:

5283 http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack

5284 However, this action is just as difficult as acknowledging the delivery, so most clients can scan for the
 5285 wsman:AckRequested field and be prepared to acknowledge delivery or fault it.

5286 10.8 Refusal of Delivery

5287 With all acknowledged delivery modes as described in 10.7, an event sink can refuse to take delivery
 5288 of events, either for security reasons or a policy change. It then responds with a fault rather than an
 5289 acknowledgement.

5290 In this case, the event source needs to be prepared to end the subscription even though an

5291 Unsubscribe message is not issued by the subscriber.

5292 **R10.8-1:** During event delivery, if the receiver faults the delivery with a wsman:DeliveryRefused
 5293 fault, the service shall immediately cancel the subscription and may also issue a SubscriptionEnd
 5294 message to the EndTo endpoint in the original subscription, if supported.

5295 Thus, the receiver can issue the fault as a way to cancel the subscription when it does not have the
 5296 SubscriptionManager information.

5297 10.9 Dropped Events

5298 Events that cannot be delivered are not to be silently dropped from the event stream, or the
 5299 subscriber gets a false picture of the event history. WS-Management defines three behaviors for
 5300 events that cannot be delivered with push modes or that are too large to fit within the delivery
 5301 constraints requested by the subscriber:

- 5302 • Terminate the subscription.
- 5303 • Silently skip such events.
- 5304 • Send a special event in place of the dropped events.

5305 These options are discussed in 10.2.9.2 and 10.2.9.3.

5306 During delivery, the service might have to drop events for the following reasons:

- 5307 • The events exceed the maximum size requested by the subscriber.
- 5308 • The client cannot keep up with the event flow, and there is a backlog.
- 5309 • The service might have been reconfigured or restarted and the events permanently lost.

5310 In these cases, a service can inform the client that events have been dropped.

5311 **R10.9-1:** If a service drops events, it should issue an
 5312 <http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents> event, which indicates this drop
 5313 to the client. Any reference parameters specified in the NotifyTo address in the subscription shall
 5314 also be copied into this message. This event is normal and implicitly considered part of any
 5315 subscription.

5316 **R10.9-2:** If an <http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents> event is issued, it
 5317 shall take the ordinal position of the original dropped event in the delivery stream. The
 5318 DroppedEvents event is considered the same as any other event with regard to its location and
 5319 other behavior (bookmarks, acknowledged delivery, location in batch, and so on). It simply takes
 5320 the place of the event that was dropped.

5321 EXAMPLE:

```

5322 (1) <s:Envelope ...>
5323 (2)   <s:Header>
5324 (3)     ...subscriber endpoint-reference...
5325 (4)   </s:Header>
5326 (5)     <wsa:Action>
5327 (6)       http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents
5328 (7)     </wsa:Action>
5329 (8)   </s:Header>
5330 (9)   <s:Body>
5331 (10)    <wsman:DroppedEvents Action="wsa:Action URI of dropped event">
5332 (11)      xs:int
5333 (12)    </wsman:DroppedEvents>
5334 (13)    ...
5335 (14)  </s:Body>
5336 (15) </s:Envelope>

```

5337 The following definitions provide additional, normative constraints on the preceding outline:

5338 s:Envelope/s:Header/wsa:Action

5339 URI that shall be defined as

5340 `http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents`

5341 s:Body/wsman:DroppedEvents/@Action

5342 the Action URI of the event that was dropped

5343 s:Body/wsman:DroppedEvents

5344 a positive integer that represents the total number of dropped events since the subscription was
5345 created

5346 Renew has no effect on the running total of dropped events. Dropped events are like any other
5347 events and can require acknowledgement, affect the bookmark location, and so on.

5348 EXAMPLE: Following is an example of how a dropped event would appear in the middle of a batched
5349 event delivery:

```
5350 (1) <wsman:Events>
5351 (2)   <wsman:Event Action="https://foo.com/someEvent">
5352 (3)     ...event body
5353 (4)   </wsman:Event>
5354 (5)   <wsman:Event
5355 (6)     Action="http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents">
5356 (7)     <wsman:DroppedEvents Action="https://foo.com/someEvent">
5357 (8)       1
5358 (9)     </wsman:DroppedEvents>
5359 (10)   </wsman:Event>
5360 (11)  <wsman:Event Action="https://foo.com/someEvent">
5361 (12)    ...event body...
5362 (13)  </wsman:Event>
5363 (14) </wsman:Events>
```

5364 **R10.9-3:** If a service cannot deliver an event and does not support the
5365 `http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents` event, it should terminate the
5366 subscription rather than silently skipping events.

5367 Because this requirement cannot be enforced, and some dropped events are irrelevant when
5368 replaced by a subsequent event (running totals, for example), it is not a firm requirement that dropped
5369 events are signaled or that they result in a termination of the subscription.

5370 10.10 Access Control

5371 It is important for event sources to properly authorize requests. This is especially true for Subscribe
5372 requests, because otherwise the ability to subscribe on behalf of a third-party event sink could be
5373 used to create a distributed denial-of-service attack.

5374 Some possible schemes for validating Subscribe requests include:

- 5375 • Send a message to the event sink that describes the requested subscription, and then wait
5376 for a confirmation message to be returned by the event sink, before the event source
5377 accepts the subscription request. While this provides strong assurance that the event sink
5378 actually desires the requested subscription, it does not work for event sinks that are not
5379 capable of sending a confirmation, and requires additional logic on the event sink.
- 5380 • Require user authentication on the Subscribe request, and allow only authorized users to
5381 Subscribe.

5382 Other mechanisms are also possible. Be aware that event sources that are not reachable from the
5383 Internet have less need to control Subscribe requests.

5384 10.11 Implementation Considerations

5385 Implementations should generate expirations in Subscribe and Renew request and response
5386 messages that are significantly larger than expected network latency.

5387 Event sinks should be prepared to receive notifications after sending a Subscribe request but before
5388 receiving a Subscribe response message. Event sinks should also be prepared to receive
5389 notifications after receiving an Unsubscribe response message.

5390 10.12 Advertisement of Notifications

5391 An Event Source can choose to advertise the Notification messages that it might send by including a
5392 well-defined portType, called "EventSink", in its WSDL. Subscribers can examine this portType to
5393 determine which messages they might need to support. Each Notification appears as an independent
5394 operation within the portType, as shown in the following example:

5395 EXAMPLE:

```
5396 (1) <wsdl:portType name="EventSink">  
5397 (2)   <wsdl:operation name="WeatherReport">  
5398 (3)     <wsdl:input message="wr:ThunderStormMessage"  
5399 (4)       wsa:Action="urn:weatherReport:ThunderStorm"  
5400 (5)       wsam:Action="urn:weatherReport:ThunderStorm" />  
5401 (6)   <wsdl:input message="wr:TyphoonMessage"  
5402 (7)     wsa:Action="urn:weatherReport:Typhoon"  
5403 (8)     wsam:Action="urn:weatherReport:Typhoon" />  
5404 (9)   </wsdl:operation>  
5405 (10) </wsdl:portType>
```

5406 In the preceding example this Event Source can send two types of Notifications (a ThunderStorm and a Typhoon
5407 message).

5408 Unless otherwise noted, Event Sinks should assume that the Notifications will be sent using SOAP1.2
5409 and will use document-literal encoding.

5410 11 Metadata and Discovery

5411 The WS-Management protocol is compatible with many techniques for discovery of resources
5412 available through a service.

5413 In addition, this specification defines a simple request-response operation to facilitate the process of
5414 establishing communications with a WS-Management service implementation in a variety of network
5415 environments without prior knowledge of the protocol version or versions supported by the
5416 implementation. This operation is used to discover the presence of a service that is compatible with
5417 WS-Management, assuming that a transport address over which the message can be delivered is
5418 known. Typically, a simple HTTP address would be used.

5419 To ensure forward compatibility, the message content of this operation is defined in an XML
5420 namespace that is separate from the core protocol namespace and that will not change as the
5421 protocol evolves. Further, this operation does not depend on any SOAP envelope header or body
5422 content other than the types explicitly defined for this operation. In this way, WS-Management clients
5423 are assured of the ability to use this operation against all implementations and versions to confirm the
5424 presence of WS-Management services without knowing the supported protocol versions or features in
5425 advance.

5426 The request message is defined as follows:

```

5427 (1) <s:Envelope
5428 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
5429 (3)   xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/identity/1/
5430 (4)     wsmanidentity.xsd"
5431 (5)   <s:Header>
5432 (6)     ...
5433 (7)   </s:Header>
5434 (8)   <s:Body>
5435 (9)     <wsmid:Identify>
5436 (10)     ...
5437 (11)   </wsmid:Identify>
5438 (12) </s:Body>
5439 (13) </s:Envelope>

```

5440 The following definitions provide additional, normative constraints on the preceding outline:

5441 wsmid:Identify

5442 the body of the Identify request operation, which may contain additional vendor-specific
 5443 extension content, but is otherwise empty

5444 The presence of this body element constitutes the request.

5445 Notice the absence of any Addressing namespace, WS-Management namespace, or other version-
 5446 specific concepts. This message is compatible only with the basic SOAP specification, and the
 5447 presence of the wsmid:Identify block in the s:Body is the embodiment of the request operation.

5448 The response message is defined as follows:

```

5449 (13) <s:Envelope
5450 (14)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
5451 (15)   xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/identity/1/
5452 (16)     wsmanidentity.xsd">
5453 (17)   <s:Header>
5454 (18)     ...
5455 (19)   </s:Header>
5456 (20)   <s:Body>
5457 (21)     <wsmid:IdentifyResponse>
5458 (22)       <wsmid:ProtocolVersion> xs:anyURI </wsmid:ProtocolVersion> +
5459 (23)       <wsmid:ProductVendor> xs:string </wsmid:ProductVendor> ?
5460 (24)       <wsmid:ProductVersion> xs:string </wsmid:ProductVersion> ?
5461 (25)       <wsmid:InitiativeSupport>
5462 (26)         <wsmid:InitiativeName> xs:string </wsmid:InitiativeName> ?
5463 (27)         <wsmid:InitiativeVersion> xs:string </wsmid:InitiativeVersion> ?
5464 (28)       </wsmid:InitiativeSupport> ?
5465 (29)       <wsmid:SecurityProfiles>
5466 (30)         <wsmid:SecurityProfileName> xs:anyURI
5467 (31)         </wsmid:SecurityProfileName> *
5468 (32)       </wsmid:SecurityProfiles> ?
5469 (33)       <wsmid:AddressingVersionURI> xs:anyURI
5470 (34)       </wsmid:AddressingVersionURI> *
5471 (35)     ...
5472 (36)   </wsmid:IdentifyResponse>
5473 (37) </s:Body>
5474 (38) </s:Envelope>

```

- 5475 The following definitions provide additional, normative constraints on the preceding outline:
- 5476 `wsmid:IdentifyResponse`
- 5477 the body of the response, which packages metadata about the WS-Management implementation
- 5478 `wsmid:IdentifyResponse/wsmid:ProtocolVersion`
- 5479 a required element or elements, each of which is a URI whose value shall be equal to the core
- 5480 XML namespace that identifies a supported version of the WS-Management specification
- 5481 One element shall be provided for each supported version of the protocol. Services should also
- 5482 include the XML namespace URI for supported dependent specifications such as Addressing.
- 5483 For example, if a future version of WS-Management supports multiple versions of Addressing,
- 5484 the `IdentifyResponse` can indicate which of the versions are supported.
- 5485 `wsmid:IdentifyResponse/wsmid:ProductVendor`
- 5486 an optional element that identifies the vendor of the WS-Management service implementation by
- 5487 using a widely recognized name or token, such as the official corporate name of the vendor or its
- 5488 stock symbol
- 5489 Alternatively, a DNS name, e-mail address, or Web URL may be used.
- 5490 `wsmid:IdentifyResponse/wsmid:ProductVersion`
- 5491 an optional version string for the WS-Management implementation
- 5492 This specification places no constraints on the format or content of this element.
- 5493 `wsmid:IdentifyResponse/wsmid:InitiativeSupport`
- 5494 an optional element that identifies an initiative supported by the WS-Management
- 5495 implementation.
- 5496 `wsmid:IdentifyResponse/wsmid:InitiativeSupport/wsmid:InitiativeName`
- 5497 an element that identifies the name of an initiative supported by the WS-Management
- 5498 implementation.
- 5499 `wsmid:IdentifyResponse/wsmid:InitiativeSupport/wsmid:InitiativeVersion`
- 5500 an element that identifies the version of an initiative supported by the WS-Management
- 5501 implementation.
- 5502 In addition, vendor-specific content can follow the preceding standardized elements. After the vendor-
- 5503 specific content, the following elements can follow:
- 5504 `wsmid:IdentifyResponse/wsmid:SecurityProfiles`
- 5505 an optional element that identifies the set of security profiles supported by the WS-Management
- 5506 implementation.
- 5507 `wsmid:IdentifyResponse/wsmid:SecurityProfiles/wsmid:SecurityProfileName`
- 5508 an optional element which is a URI that identifies a security profile supported by the WS-
- 5509 Management implementation.
- 5510 `wsmid:IdentifyResponse/wsmid:AddressingVersionURI`
- 5511 an optional element which is a URI that identifies a version of Addressing supported by the WS-
- 5512 Management implementation.
- 5513 When a service supports this element, the value shall be the XML Schema namespace URI of
- 5514 the addressing version in use. XML Schema namespaces used in this specification are listed in
- 5515 ANNEX A. A service may support and advertise more than one version of addressing.
- 5516 **R11-1:** A WS-Management service should support the `wsmid:Identify` operation. A service
- 5517 implementation that supports the operation shall do so irrespective of the versions of
- 5518 WS-Management supported by that service. The operation shall be accessible at the same

- 5519 transport-level address at which the resource instances are made accessible.
- 5520 It is recommended that client applications not include any SOAP header content in the wsmid:Identify
 5521 operation delivered to the transport address against which the inquiry is being made. If SOAP header
 5522 elements are present, the s:mustUnderstand attribute on all such elements can be set to "false".
 5523 Doing otherwise reduces the likelihood of a successful, version-independent response from the
 5524 service.
- 5525 **R11-2:** A service that supports the wsmid:Identify operation shall not require the presence of any
 5526 SOAP header elements in order to dispatch execution of the request. If a service receives a
 5527 wsmid:Identify operation that contains unexpected or unsupported header content with the
 5528 s:mustUnderstand attribute set to "false", the service shall not fault the request and shall process
 5529 the body of the request as though the header elements were not present.
- 5530 **R11-3:** A service that is processing the wsmid:Identify request should not request the presence
 5531 of any Addressing header values, including the wsa:Action URI.
- 5532 The entire purpose of this mechanism is to be able to identify the presence of specific versions of
 5533 WS-Management (and the corresponding dependent protocols) in a version-independent manner.
- 5534 Because Addressing is not used, the address to which this message is delivered is defined entirely at
 5535 the transport level and not present in the SOAP content.
- 5536 If a client does not have any prior knowledge about a service including credentials, it is desirable to
 5537 allow a service to process an Identify message without requiring authentication.
- 5538 **R11-4:** A service that supports the wsmid:Identify operation may expose this operation without
 5539 requiring client or server authentication in order to process the message. In the absence of other
 5540 requirements, it is recommended that the network address be suffixed by the token sequence
 5541 /wsman-anon/identify.
- 5542 Services that support unauthenticated wsmid:Identify requests might choose not to reveal descriptive
 5543 information about protocol, vendor, or other versioning information that could potentially represent or
 5544 contribute to a vulnerability. To accommodate this scenario, this specification defines a URI that
 5545 services can use in place of a valid WS-Management protocol version URI. This value can be
 5546 returned as a value for the wsmid:ProtocolVersion element of the wsmid:IdentifyResponse message.
- 5547 **R11-5:** A service supporting an unauthenticated wsmid:Identify message may respond using the
 5548 following URI for the value of the wsmid:ProtocolVersion element:
- 5549 <http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity/NoAnonymousDisclosure>
- 5550 **R11-6:** A service that provides unauthenticated access to the wsmid:Identify operation but does
 5551 not respond to such requests with the WS-Management protocol versions that are supported by
 5552 the service shall support authenticated access to the wsmid:Identify operation. Such services
 5553 shall respond to authenticated requests with the WS-Management protocol version identifiers for
 5554 each version of the WS-Management protocol supported by the service.

5555 12 Security

5556 12.1 General

- 5557 In general, management operations and responses need to be protected against attacks such as
 5558 snooping, interception, replay, and modification during transmission. Authenticating the user who has
 5559 sent a request is also generally necessary so that access control rules can be applied to determine
 5560 whether to process a request.

5561 This specification establishes the minimum interoperation standards and predefined profiles using
5562 transport-level security.

5563 This approach provides the best balance between simple implementations (HTTP and HTTPS stacks
5564 are readily available, even for hardware) and the security mechanisms that sit in front of any SOAP
5565 message processing, limiting the attack surface.

5566 It is expected that more sophisticated transport and SOAP-level profiles, published separately from
5567 this specification, will be defined and used.

5568 Implementations that expect to interoperate can adopt one or more of the transport and security
5569 models defined in this clause and are free to define any additional profiles under different URI-based
5570 designators.

5571 **12.2 Security Profiles**

5572 For this specification, a profile is any arbitrary mix of transport or SOAP behavior that describes a
5573 common security need. In some cases, the profile is defined for documentation and metadata
5574 purposes, but might not be part of the actual message exchange. Rather, it *describes* the message
5575 exchange involved.

5576 Metadata retrieval can be employed to discover which profiles the service supports, and that is
5577 beyond the scope of this particular specification.

5578 For all predefined profiles, the transport is responsible for all message integrity, protection,
5579 authentication, and security.

5580 The authentication profiles do not appear in the SOAP traffic, with the exception of the Subscribe
5581 message when using any delivery mode that causes a new connection to be created from the event
5582 source to the event sink (push and batched modes, for example). When a subscription is created, the
5583 authentication technique for event-delivery needs to be specified by the subscriber, because the
5584 event sink has to authenticate the event source (acting as publisher) at event delivery-time.

5585 In this specification, security profiles are identified by a URI. As profiles are defined, they can be
5586 assigned a URI and published. WS-Management defines a set of standardized security profiles for
5587 the common transports HTTP and HTTPS as described in C.3.1.

5588 **12.3 Security Considerations for Event Subscriptions**

5589 When specifying the NotifyTo address in subscriptions, it is often important to hint to the service
5590 about which authentication model to use when delivering the event.

5591 If no hints are present, the service can simply infer from the wsa:To address what needs to be done.
5592 However, if the service can support multiple modes and has a certificate or password store, it might
5593 not know which authentication model to choose or which credentials to use without being told in the
5594 subscription.

5595 WS-Management provides a default mechanism to communicate the desired authentication mode
5596 and credentials. However, more sophisticated mechanisms are beyond the scope of this version of
5597 WS-Management. For example, the event sink service could export metadata that describes the
5598 available options, allowing the publisher to negotiate an appropriate option. Extension profiles can
5599 define other mechanisms enabled through a SOAP header with mustUnderstand="true".
5600 WS-Management defines an additional field in the Delivery block that can communicate
5601 authentication information, as shown in the following outline:

```
5602 (1) <s:Body>
5603 (2)   <wsme:Subscribe>
5604 (3)     <wsme:Delivery>
```

```

5605 (4)      <wsme:NotifyTo> Delivery EPR </wsme:NotifyTo>
5606 (5)      <wsman:Auth Profile="authentication-profile-URI"/>
5607 (6)      </wsme:Delivery>
5608 (7)      </wsme:Subscribe>
5609 (8)      </s:Body>

```

5610 The following definitions provide additional, normative constraints on the preceding outline:

5611 **wsman:Auth**

5612 block that contains authentication information to be used by the service (acting as publisher)
 5613 when authenticating to the event sink at event delivery time

5614 **wsman:Auth/@Profile**

5615 a URI that indicates which security profile to use when making the connection to deliver events

5616 If the **wsman:Auth** block is not present, by default the service infers what to do by using the **NotifyTo**
 5617 address and any preconfigured policy or settings it has available. If the **wsman:Auth** block is present
 5618 and no security-related tokens are communicated, the service needs to know which credentials to use
 5619 by its own internal configuration.

5620 If the service is already configured to use a specific certificate when delivering events, the subscriber
 5621 can request standard mutual authentication, as shown in the following outline:

```

5622 (1) <s:Body>
5623 (2)   <wsme:Subscribe>
5624 (3)     <wsme:Delivery>
5625 (4)       <wsme:NotifyTo> HTTPS address </wsme:NotifyTo>
5626 (5)       <wsman:Auth
5627 (6)         Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/
5628 (7)           mutual"/>
5629 (8)     </wsme:Delivery>
5630 (9)   </wsme:Subscribe>
5631 (10)  </s:Body>

```

5632 If the service knows how to retrieve a proper user name and password for event delivery, simple
 5633 HTTP Basic or Digest authentication can be used, as shown in the following outline:

```

5634 (1) <s:Body>
5635 (2)   <wsme:Subscribe>
5636 (3)     <wsme:Delivery>
5637 (4)       <wsme:NotifyTo> HTTP address </wsme:NotifyTo>
5638 (5)       <wsman:Auth
5639 (6)         Profile="http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/
5640 (7)           digest"/>
5641 (8)     </wsme:Delivery>
5642 (9)   </wsme:Subscribe>
5643 (10)  </s:Body>

```

5644 Services are not required to support any specific profile. The rest of this clause defines special-case
 5645 profiles for event delivery in which the service needs additional information to select the proper
 5646 credentials to use when delivering events.

5647 12.4 Including Credentials with a Subscription

5648 This clause intentionally left blank.

5649 12.5 Correlating Events with a Subscription

5650 In many cases, the subscriber will want to ensure that the event delivery corresponds to a valid
 5651 subscription issued by an authorized party. In this case, it is recommended that reference parameters
 5652 be introduced into the NotifyTo definition.

5653 EXAMPLE: At subscription time, a UUID could be supplied as a correlation token:

```

5654 (1) <s:Body>
5655 (2)   <wsme:Subscribe>
5656 (3)     <wsme:Delivery>
5657 (4)       <wsme:NotifyTo>
5658 (5)         <wsa:Address> address <wsa:Address>
5659 (6)         <wsa:ReferenceParameters>
5660 (7)           <MyNamespace:uuid>
5661 (8)             uuid:b0f685ec-e5c9-41b5-b91c-7f580419093e
5662 (9)           </MyNamespace:uuid>
5663 (10)        </wsa:ReferenceParameters>
5664 (11)       </wsme:NotifyTo>
5665 (12)      ...
5666 (13)     </wsme:Delivery>
5667 (14)    ...
5668 (15)   </wsme:Subscribe>
5669 (16) </s:Body>
  
```

5670 This definition requires that the service include the MyNamespace:uuid value as a SOAP header with
 5671 each event delivery (see 5.1). The service can use this value to correlate the event with any
 5672 subscription that it issued and to validate its origin.

5673 This is not a transport-level or SOAP-level authentication mechanism as such, but it does help to
 5674 maintain and synchronize valid lists of subscriptions and to determine whether the event delivery is
 5675 authorized, even though the connection itself could have been authenticated.

5676 This mechanism still can require the presence of the wsman:Auth block to specify which security
 5677 mechanism to use to actually authenticate the connection at event-time.

5678 Each new subscription can receive at least one unique reference parameter that is never reused,
 5679 such as the illustrated UUID, for this mechanism to be of value.

5680 Other reference parameters can be present to help route and correlate the event delivery as required
 5681 by the subscriber.

5682 12.6 Transport-Level Authentication Failure

5683 Because transports typically go through their own authentication mechanisms prior to any SOAP
 5684 traffic occurring, the first attempt to connect might result in a transport-level authentication failure. In
 5685 such cases, SOAP faults will not occur, and the means of communicating the denial to the client is
 5686 implementation- and transport-specific.

5687 12.7 Security Implications of Third-Party Subscriptions

5688 Without proper authentication and authorization, WS-Management implementations can be
 5689 vulnerable to distributed denial-of-service attacks through third-party subscriptions to events. This
 5690 vulnerability is discussed in 10.10.

13 Transports and Message Encoding

This clause describes encoding rules that apply to all transports.

13.1 SOAP

WS-Management qualifies the use of SOAP as indicated in this clause.

R13.1-1: A service shall at least receive and send [SOAP 1.2](#) SOAP Envelopes.

R13.1-2: A service may reject a SOAP Envelope with more than 32,767 octets.

R13.1-3: A service should not send a SOAP Envelope with more than 32,767 octets unless the client has specified a wsman:MaxEnvelopeSize header that overrides this limit.

Large SOAP Envelopes are expected to be serialized using attachments.

R13.1-4: Any Request Message may be encoded using either Unicode 3.0 (UTF-16) or UTF-8 encoding. A service shall accept the UTF-8 encoding type for all operations and should accept UTF-16 as well.

R13.1-5: A service shall emit Responses using the same encoding as the original request. If the service does not support the requested encoding or cannot determine the encoding, it should use UTF-8 encoding to return a wsman:EncodingLimit fault with the following detail code:

`http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/CharacterSet`

R13.1-6: For UTF-8 encodings, the service may fail to process any message that begins with the UTF-8 BOM (0xEF 0xBB 0xBF), and shall send UTF-8 responses without the BOM.

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

R13.1-7: If UTF-16 is the encoding, the service shall support either byte-order mark (BOM) U+FEFF (big-endian) or U+FFFE (little-endian) as defined in the [Unicode 3.0](#) specification as the first character in the message (see the [Unicode BOM FAQ](#)).

R13.1-8: If a request includes contradictory encoding information in the BOM and HTTP charset header or if the information does not fully specify the encoding, the service shall fault with an HTTP status of "bad request message" (400).

Repeated headers with the same QName but different values that imply contradictory behavior are considered a defect originating on the client side of the conversation. Returning a fault helps identify faulty clients. However, an implementation might be resource-constrained and unable to detect duplicate headers, so the repeated headers can be ignored. Repeated headers with the same QName that contains informational or non-contradictory instructions are possible, but none are defined by this specification or its dependencies.

R13.1-9: If a request contains multiple SOAP headers with the same QName from WS-Management, Addressing, or clause 10 of this specification, the service should not process them and should issue a wsa:InvalidMessageInformationHeaders fault if they are detected. (No SOAP headers are defined in clause 7 "Resource Access" or clause 8 "Enumeration of Datasets".)

R13.1-10: By default, a compliant service should not fault requests with leading and trailing whitespace in XML element values and should trim such whitespace by default as if the whitespace had not occurred. Services should not emit messages containing leading or trailing whitespace within element values unless the whitespace values are properly part of the value. If the service cannot accept whitespace usage within a message because the XML schema

5733 establishes other whitespace usage, the service should emit a wsman:EncodingLimit fault with
5734 the following detail code:

5735 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Whitespace>

5736 Clients can send messages with leading or trailing whitespace in the values, and services are
5737 permitted to eliminate unneeded "cosmetic" whitespace on both sides of the element value without
5738 faulting. (See [XML Schema Part 2: Datatypes](#).)

5739 **R13.1-11:** Services should not fault messages that contain XML comments, because this is part
5740 of the XML standard. Services may emit messages that contain comments that relate to the origin
5741 and processing of the message or add comments for debugging purposes.

5742 13.2 Lack of Response

5743 If an operation succeeds but a response cannot be computed or actually delivered because of run-
5744 time difficulties or transport problems, no response is sent and the connection is terminated.

5745 This behavior is preferable to attempting a complex model for sending responses in a delayed
5746 fashion. Implementations can generally keep a log of all requests and their results, and allow the
5747 client to reconnect later to enumerate the operation log (using Enumerate) if it failed to get a
5748 response. The format and behavior of such a log is beyond the scope of this specification. In any
5749 case, the client needs to be coded to take into account a lack of response; all abnormal message
5750 conditions can safely revert to this scenario.

5751 **R13.2-1:** If correct responses or faults cannot be computed or generated due to internal service
5752 failure, a response should not be sent.

5753 Regardless, the client has to deal with cases of no response, so the service can simply force the
5754 client into that mode rather than send a response or fault that is not defined in this specification.

5755 13.3 Replay of Messages

5756 This section intentionally left blank.

5757 **R13.3-1:** This rule intentionally left blank.

5758 13.4 Encoding Limits

5759 Most of the following limits are in characters. However, the maximum overall SOAP envelope size is
5760 defined in octets. Implementations are free to exceed these limits. A service is considered conformant
5761 if it observes these limits. Any limit violation results in a wsman:EncodingLimit fault.

5762 **R13.4-1:** A service may fail to process any URI with more than 2048 characters and should
5763 return a wsman:EncodingLimit fault with the following detail code:

5764 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/URILimitExceeded>

5765 **R13.4-2:** A service should not generate a URI with more than 2048 characters.

5766 **R13.4-3:** A service may fail to process an Option Name of more than 2048 characters.

5767 **R13.4-4:** A service may fail to process an Option value of more than 4096 characters.

5768 **R13.4-5:** A service may fault any operation that would require a single reply exceeding 32,767
5769 octets.

5770 **R13.4-6:** A service may always emit faults that are 4096 octets or less in length, regardless of
 5771 any requests by the client to limit the response size. Clients need to be prepared for this minimum
 5772 in case of an error.

5773 **R13.4-7:** When the default addressing model is in use, a service may fail to process a Selector
 5774 Name of more than 2048 characters.

5775 **R13.4-8:** A service may have a maximum number of selectors that it can process. If the request
 5776 contains more selectors than this limit, the service should return a wsman:EncodingLimit fault
 5777 with the following detail code:

5778 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/SelectorLimit`

5779 **R13.4-9:** A service may have a maximum number of options that it can process. If the request
 5780 contains more options than this limit, the service should return a wsman:EncodingLimit fault with
 5781 the following detail code:

5782 `http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionLimit`

5783 13.5 Binary Attachments

5784 SOAP Message Transmission Optimization Mechanism (MTOM) is used to support binary
 5785 attachments to WS-Management. If a service supports attachments, the following rules apply:

5786 **R13.5-1:** A conformant service may optionally support binary attachments to any operation using
 5787 the [SOAP MTOM](#) proposal.

5788 **R13.5-2:** If a service supports attachments, the service shall support the Abstract Transmission
 5789 Optimization Feature.

5790 **R13.5-3:** If a service supports attachments, the service shall support the Optimized MIME
 5791 Multipart Serialization Feature.

5792 Other attachment types are not prohibited. Specific transports can impose additional encoding rules.

5793 13.6 Case-Sensitivity

5794 While XML and SOAP are intrinsically case-sensitive with regard to schematic elements,
 5795 WS-Management can be used with many underlying systems that are not intrinsically case-sensitive.
 5796 This support primarily applies to values, but can also apply to schemas that are automatically and
 5797 dynamically generated from other sources.

5798 A service can observe any case usage required by the underlying execution environment.

5799 The only requirement is that messages are able to pass validation tests against any schema
 5800 definitions. At any time, a validation engine could be interposed between the client and server in the
 5801 form of a proxy, so schematically valid messages are a practical requirement.

5802 Otherwise, this specification makes no requirements as to case usage. A service is free to interpret
 5803 values in a case-sensitive or case-insensitive manner.

5804 It is recommended that case usage not be altered in transit by any part of the WS-Management
 5805 processing chain. The case usage established by the sender of the message is to be retained
 5806 throughout the lifetime of that message.

5807 **14 Faults**

5808 Many of the operations outlined in WS-Management can generate faults. This clause describes how
 5809 these faults should be formatted into SOAP messages.

5810 **14.1 Introduction**

5811 Faults are returned when the SOAP message is successfully delivered by the transport and
 5812 processed by the service, but the message cannot be processed properly. If the transport cannot
 5813 successfully deliver the message to the SOAP processor, a transport error occurs.

5814 **R14.1-1:** A service should support only [SOAP 1.2](#) (or later) faults.

5815 Generally, faults are not to be issued unless they are expected as part of a request-response pattern.
 5816 For example, it would not be valid for a client to issue a Get message, receive the GetResponse
 5817 message, and then *fault* that response.

5818 **14.2 Fault Encoding**

5819 This clause discusses XML fault encoding.

5820 **R14.2-1:** A conformant service shall use the following fault encoding format and normative
 5821 constraints for faults in the WS-Management space or any of its dependent specifications:

```

5822 (1) <s:Envelope>
5823 (2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
5824 (3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
5825 (4)   <s:Header>
5826 (5)     <wsa:Action>
5827 (6)       http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
5828 (7)   </wsa:Action>
5829 (8)   <wsa:MessageID>
5830 (9)     uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
5831 (10)  </wsa:MessageID>
5832 (11)  <wsa:RelatesTo>
5833 (12)    uuid:d9726315-bc91-430b-9ed8-ce5ffb858a85
5834 (13)  </wsa:RelatesTo>
5835 (14) </s:Header>
5836 (15)
5837 (16) <s:Body>
5838 (17)   <s:Fault>
5839 (18)     <s:Code>
5840 (19)       <s:Value> [Code] </s:Value>
5841 (20)     <s:Subcode>
5842 (21)       <s:Value> [Subcode] </s:Value>
5843 (22)     </s:Subcode>
5844 (23)   </s:Code>
5845 (24)   <s:Reason>
5846 (25)     <s:Text xml:lang="en"> [Reason] </s:Text>
5847 (26)   </s:Reason>
5848 (27)   <s:Detail>
5849 (28)     [Detail]
5850 (29)   </s:Detail>
5851 (30) </s:Fault>
5852 (31) </s:Body>
5853 (32) </s:Envelope>

```

- 5854 The following definitions provide additional, normative constraints on the preceding outline:
- 5855 s:Envelope/s:Header/wsa:Action
- 5856 a valid fault Action URI from the relevant specification that defined the fault
- 5857 s:Envelope/s:Header/wsa:MessageId
- 5858 element that shall be present for the fault, like any non-fault message
- 5859 s:Envelope/s:Header/wsa:RelatesTo
- 5860 element that shall, like any other reply, contain the MessageID of the original request that
- 5861 caused the fault
- 5862 s:Body/s:Fault/s:Value
- 5863 element that shall be either s:Sender or s:Receiver, as specified in 14.6 in the "Code" field
- 5864 s:Body/s:Fault/s:Subcode/s:Value
- 5865 for WS-Management-related messages, shall be one of the subcode QNames defined in 14.6
- 5866 If the service exposes custom methods or other messaging, this value may be another QName
- 5867 not in the Master Faults described in 14.6.
- 5868 s:Body/s:Fault/s:Reason
- 5869 optional element that should contain localized text that explains the fault in more detail
- 5870 Typically, this text is extracted from the "Reason" field in the Master Fault tables (14.6).
- 5871 However, the text may be adjusted to reflect a specific circumstance. This element may be
- 5872 repeated for multiple languages. The xml:lang attribute shall be present on the s:Text element.
- 5873 s:Body/s:Fault/s:Detail
- 5874 optional element that should reflect the recommended content from the Master Fault tables
- 5875 (14.6)
- 5876 The preceding fault template is populated by examining entries from the Master Fault tables in 14.6,
- 5877 which includes all relevant faults from WS-Management and its underlying specifications.
- 5878 s:Reason and s:Detail are always optional, but they are recommended. In addition, s:Reason/s:Text
- 5879 contains an xml:lang attribute to indicate the language used in the descriptive text.
- 5880 **R14.2-2:** Fault wsa:Action URI values vary from fault to fault. The service shall issue a fault
- 5881 using the correct URI, based on the specification that defined the fault. Faults defined in this
- 5882 specification shall have the following URI value:
- 5883 http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
- 5884 The Master Fault tables in 14.6 contain the relevant wsa:Action URIs. The URI values are directly
- 5885 implied by the QName for the fault.

5886 14.3 NotUnderstood Faults

- 5887 There is a special case for faults relating to mustUnderstand attributes on SOAP headers. SOAP
- 5888 specifications define the fault differently than the encoding in 14.2 (see 5.4.8 in [SOAP 1.2](#)). In
- 5889 practice, the fault varies only in indicating the SOAP header that was not understood, the QName,
- 5890 and the namespace (see line 5 in the following outline).

```

5891 (1) <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
5892 (2)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
5893 (3)
5894 (4)   <s:Header>
5895 (5)     <s:NotUnderstood qname="QName of header" xmlns:ns="XML namespace of
5896       header"/>

```

```

5897 (6)      <wsa:Action>
5898 (7)      http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
5899 (8)      </wsa:Action>
5900 (9)      <wsa:MessageID>
5901 (10)     urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
5902 (11)     </wsa:MessageID>
5903 (12)     <wsa:RelatesTo>
5904 (13)     urn:uuid:d9726315-bc91-430b-9ed8-ce5ffb858a85
5905 (14)     </wsa:RelatesTo>
5906 (15)    </s:Header>
5907 (16)
5908 (17)    <s:Body>
5909 (18)    <s:Fault>
5910 (19)    <s:Code>
5911 (20)    <s:Value>s:MustUnderstand</s:Value>
5912 (21)    </s:Code>
5913 (22)    <s:Reason>
5914 (23)    <s:Text xml:lang="en-US">Header not understood</s:Text>
5915 (24)    </s:Reason>
5916 (25)    </s:Fault>
5917 (26)    </s:Body>
5918 (27)
5919 (28) </s:Envelope>

```

5920 The preceding fault template can be used in all cases of failure to process mustUnderstand attributes.
 5921 Lines 5–8 show the important content, indicating which header was not understood and including a
 5922 generic wsa:Action that specifies that the current message is a fault.

5923 The wsa:RelatesTo element is included so that the client can correlate the fault with the original
 5924 request. Over transports other than HTTP in which requests might be interlaced, this might be the
 5925 only way to respond to the correct sender.

5926 If the original wsa:MessageID itself is faulty and the connection is request-response oriented, the
 5927 service can attempt to send back a fault without the wsa:RelatesTo field, or can simply fail to
 5928 respond, as discussed in 14.4.

5929 **14.4 Degenerate Faults**

5930 In rare cases, the SOAP message might not contain enough information to properly generate a fault.
 5931 For example, if the wsa:MessageID is garbled, the service will have difficulty returning a fault that
 5932 references the original message. Some transports might not be able to reference the sender to return
 5933 the fault.

5934 If the transport guarantees a simple request-response pattern, the service can send back a fault with
 5935 no wsa:RelatesTo field. However, in some cases, there is no guarantee that the sender can be
 5936 reached (for example, if the wsa:FaultTo contains an invalid address, so there is no way to deliver the
 5937 fault).

5938 In all cases, the service can revert to the rules of 13.3, in which no response is sent. The service can
 5939 attempt to log the requests in some way to help identify the defective client.

5940 **14.5 Fault Extensibility**

5941 A service can include additional fault information beyond what is defined in this specification. The
 5942 appropriate extension element is the s:Detail element, and the service-specific XML can appear at
 5943 any location within this element, provided that it is properly mapped to an XML namespace that
 5944 defines the schema for that content. WS-Management makes use of this extension technique for the
 5945 wsman:FaultDetail URI values, as shown in the following outline:

```

5946 (1) <s:Detail>
5947 (2)   <wsman:FaultDetail>... </wsman:FaultDetail>
5948 (3)   <ExtensionData xmlns="vendor-specific-namespace">...</ExtensionData>
5949 (4)   ...
5950 (5) </s:Detail>

```

5951 The extension data elements can appear before or after any WS-Management-specific extensions
 5952 mandated by this specification. More than one extension element is permitted.

5953 14.6 Master Faults

5954 This clause includes all faults from this specification and all underlying specifications. This list is the
 5955 normative fault list for WS-Management.

5956 **R14.6-1:** A service shall return faults from the following list when the operation that caused them
 5957 was a message in this specification for which faults are specified. A conformant service may
 5958 return other faults for messages that are not part of WS-Management.

5959 It is critical to client interoperability that the same fault be used in identical error cases. If each service
 5960 returns a distinct fault for "Not Found", for example, constructing interoperable clients would be
 5961 impossible. In Table 5 through Table 43, the source specification of a fault is based on its QName.

5962 **Table 5 – wsman:AccessDenied**

Fault Subcode	wsman:AccessDenied
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The sender was not authorized to access the resource.
Detail	None
Comments	This fault is returned generically for all access denials that relate to authentication or authorization failures. This fault does not indicate locking or concurrency conflicts or other types of denials unrelated to security by itself.
Applicability	Any message
Remedy	The client acquires the correct credentials and retries the operation.

5963

Table 6 – wsa:ActionNotSupported

Fault Subcode	wsa:ActionNotSupported
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	The action is not supported by the service.
Detail	<pre><s:Detail> <wsa:Action> <i>Incorrect Action URI</i> </wsa:Action> </s:Detail> <!-- The unsupported Action URI is returned, if possible --></pre>
Comments	This fault means that the requested action is not supported by the implementation. As an example, read-only implementations (supporting only Get and Enumerate) return this fault for any operations other than these two. If the implementation never supports the action, the fault can be generated as shown in the "Detail" row of this table. However, if the implementation supports the action in a general sense, but it is not an appropriate match for the resource, an additional detail code can be added to the fault, as follows: <pre><s:Detail> <wsa:Action> <i>The offending Action URI</i> </wsa:Action> <wsman:FaultDetail> http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ActionMismatch </wsman:FaultDetail> </s:Detail></pre> This situation can occur when the implementation supports Put, for example, but the client attempts to update a read-only resource.
Applicability	All messages
Remedy	The client consults metadata provided by the service to determine which operations are supported.

5964

Table 7 – wsman:AlreadyExists

Fault Subcode	wsman:AlreadyExists
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The sender attempted to create a resource that already exists.
Detail	None
Comments	This fault is returned in cases where the user attempted to create a resource that already exists.
Applicability	Create
Remedy	The client uses Put or creates a resource with a different identity.

5965

Table 8 – wsman:CannotProcessFilter

Fault Subcode	wsman:CannotProcessFilter
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The requested filter could not be processed.
Detail	<s:Detail> <wsman:SupportedSelectorName> Valid selector name for use in filter expression </wsman:SupportedSelectorName> * </s:Detail>
Comments	This fault is returned for syntax errors or other semantic problems with the filter. For use with the SelectorFilter dialect (see ANNEX E), the service can include one or more SupportedSelectorName elements to provide a list of supported selector names in the event that the client has requested filtering on one or more unsupported selector names. If the filter is valid, but the service cannot execute the filter due to misconfiguration, lack of resources, or other service-related problems, more specific faults can be returned, such as wsman:QuotaLimit or wsman:InternalError.
Applicability	Enumerate
Remedy	The client fixes the filter problem and tries again.

5966

Table 9 – wsman:CannotProcessFilter

Fault Subcode	wsman:CannotProcessFilter
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The requested filter could not be processed.
Detail	<s:Detail> <wsman:SupportedSelectorName> Valid selector name for use in filter expression </wsman:SupportedSelectorName> * </s:Detail>
Comments	This fault is returned for syntax errors or other semantic problems with the filter such as exceeding the subset supported by the service. For use with the SelectorFilter dialect (see ANNEX E), the service can include one or more SupportedSelectorName elements to provide a list of supported selector names in the event that the client has requested filtering on one or more unsupported selector names. If the filter is valid, but the service cannot execute the filter due to misconfiguration, lack of resources, or other service-related problems, more specific faults can be returned, such as wsman:QuotaLimit, wsman:InternalError, or wsme:EventSourceUnableToProcess.
Applicability	Subscribe, fragment-level resource access operations
Remedy	The client fixes the filter problem and tries again.

5967

Table 10 – wsman:Concurrency

Fault Subcode	wsman:Concurrency
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The action could not be completed due to concurrency or locking problems.
Detail	None
Comments	This fault means that the requested action could not be carried out either due to internal concurrency or locking problems or because another user is accessing the resource. This fault can occur if a resource is being enumerated using Enumerate and another client attempts operations such as Delete, which would affect the result of the enumeration in progress.
Applicability	All messages
Remedy	The client waits and tries again.

5968

Table 11 – wsme:DeliveryModeRequestedUnavailable

Fault Subcode	wsme:DeliveryModeRequestedUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The requested delivery mode is not supported.
Detail	<pre><s:Detail> <wsme:SupportedDeliveryMode>... </wsme:SupportedDeliveryMode> <wsme:SupportedDeliveryMode>...</wsme:SupportedDeliveryMode> ... </s:Detail></pre> <!-- This is a simple, optional list of one or more supported delivery mode URIs. It may be left empty. -->
Comments	This fault is returned for unsupported delivery modes for the specified resource. If the stack supports the delivery mode in general, but not for the specific resource, this fault is still returned. Other resources might support the delivery mode. The fault does not imply that the delivery mode is not supported by the implementation.
Applicability	Subscribe
Remedy	The client selects one of the supported delivery modes.

5969

Table 12 – wsman:DeliveryRefused

Fault Subcode	wsman:DeliveryRefused
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Receiver
Reason	The receiver refuses to accept delivery of events and requests that the subscription be canceled.
Detail	None
Comments	This fault is returned by event receivers to force a cancellation of a subscription. This fault can happen when the client tried to Unsubscribe, but failed, or when the client lost knowledge of active subscriptions and does not want to keep receiving events that it no longer owns. This fault can help clean up spurious or leftover subscriptions when clients are reconfigured or reinstalled and their previous subscriptions are still active.
Applicability	Any event delivery message in any mode
Remedy	The service stops delivering events for the subscription and cancels the subscription, sending any applicable SubscriptionEnd messages.

5970

Table 13 – wsa:DestinationUnreachable

Fault Subcode	wsa:DestinationUnreachable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	No route can be determined to reach the destination role defined by the Addressing To header.
Detail	<s:Detail> <wsman:FaultDetail> http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI </wsman:FaultDetail> ? </s:Detail> When the default addressing model is in use, the wsman:FaultDetail field may contain http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidResourceURI.
Comments	This fault is returned as the general "Not Found" case for a resource, in which the resource EPR cannot be mapped to the real-world resource. This fault is not used merely to indicate that the resource is temporarily offline, which is indicated by wsa:EndpointUnavailable.
Applicability	All request messages
Remedy	The client attempts to diagnose the version of the service, query any metadata, and perform other diagnostic operations to determine why the request cannot be routed.

5971

Table 14 – wsman:EncodingLimit

Fault Subcode	wsman:EncodingLimit
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	An internal encoding limit was exceeded in a request or would be violated if the message were processed.
Detail	<s:Detail> <wsman:FaultDetail> Optional; one of the following enumeration values </wsman:FaultDetail> ...any service-specific additional XML content... </s:Detail> Possible enumeration values in the <wsman:FaultDetail> element are as follows: Unsupported character set: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/CharacterSet Unsupported MTOM or other encoding types: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EncodingType Requested maximum was too large: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize Requested maximum envelope size was too small: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MinimumEnvelopeLimit Too many options: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OptionLimit Used when the default addressing model is in use and indicates that too many selectors were used for the corresponding ResourceURI: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/SelectorLimit Service reached its own internal limit when computing response: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ServiceEnvelopeLimit Operation succeeded and cannot be reversed, but result is too large to send: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnreportableSuccess Request contained a character outside of the range that is supported by the service: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnsupportedCharacter URI was too long: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/URILimitExceeded Client-side whitespace usage is not supported: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Whitespace
Comments	This fault is returned when a system limit is exceeded, whether a published limit or a service-specific limit.
Applicability	All request messages
Remedy	The client sends messages that fit the encoding limits of the service.

5972

Table 15 – wsa:EndpointUnavailable

Fault Subcode	wsa:EndpointUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Receiver
Reason	The specified endpoint is currently unavailable.
Detail	<pre><s:Detail> <wsa:RetryAfter> xs:duration </wsa:RetryAfter> <!-- optional --> ...optional service-specific XML content <wsman:FaultDetail> A detail URI value </wsman:FaultDetail> </s:Detail></pre> http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ResourceOffline Used when the resource is known, but temporarily unavailable
Comments	This fault is returned if the message was correct and the EPR was valid, but the specified resource is offline. In practice, it is difficult for a service to distinguish between "Not Found" cases and "Offline" cases. In general, wsa:DestinationUnreachable is preferable.
Applicability	All request messages
Remedy	The client can retry later, after the resource is again online.

5973

Table 16 – wsman:EventDeliverToUnusable

Fault Subcode	wsman:EventDeliverToUnusable
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The event source cannot process the subscription because it cannot connect to the event delivery endpoint as requested in the Delivery element.
Detail	<pre><s:Detail> ...any service-specific content to identify the error... </s:Detail></pre>
Comments	This fault is limited to cases of connectivity issues in contacting the "deliver to" address. These issues include: • The NotifyTo address is not usable because it is incorrect (system or device not reachable, badly formed address, and so on). • Permissions cannot be acquired for event delivery (for example, the wsman:Auth element does not refer to a supported security profile, and so on). • The credentials associated with the NotifyTo are not valid (for example, the account does not exist, the certificate thumbprint is not a hex string, and so on). The service can include extra information that describes the connectivity error to help in troubleshooting the connectivity problem.
Applicability	Subscribe
Remedy	The client ensures connectivity from the service computer back to the event sink including firewalls and authentication/authorization configuration.

5974

Table 17 – wsme:EventSourceUnableToProcess

Fault Subcode	wsme:EventSourceUnableToProcess
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Receiver
Reason	The event source cannot process the subscription.
Detail	None
Comments	This event source is not capable of fulfilling a Subscribe request for local reasons unrelated to the specific request.
Applicability	Subscribe
Remedy	The client retries the subscription later.

5975

Table 18 – wsmen:FilterDialectRequestedUnavailable

Fault Subcode	wsmen:FilterDialectRequestedUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The requested filtering dialect is not supported.
Detail	<s:Detail> <wsmen:SupportedDialect> </wsmen:SupportedDialect> + </s:Detail>
Comments	This fault is returned when the client requests a filter type or query language not supported by the service. The filter dialect can vary from resource to resource or can apply to the entire service.
Applicability	Enumerate
Remedy	The client switches to a supported dialect or performs a simple enumeration with no filter.

5976

Table 19 – wsme:FilteringNotSupported

Fault Subcode	wsme:FilteringNotSupported
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	Filtering over the event source is not supported.
Detail	None
Comments	This fault is returned when the service does not support filtered subscriptions for the specified event source, but supports only simple delivery of all events for the resource. NOTE: The service might support filtering over a different event resource or might not support filtering for any resource. The same fault applies.
Applicability	Subscribe
Remedy	The client subscribes using unfiltered delivery.

5977

Table 20 – wsmen:FilteringNotSupported

Fault Subcode	wsmen:FilteringNotSupported
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	Filtered enumeration is not supported.
Detail	None
Comments	This fault is returned when the service does not support filtering of enumerations at all, but supports only simple enumeration. If enumeration as a whole is not supported, the correct fault is wsa:ActionNotSupported. NOTE: The service might support filtering over a different enumerable resource or might not support filtering for <i>any</i> resource. The same fault applies.
Applicability	Enumerate
Remedy	The client switches to a simple enumeration.

5978

Table 21 – wsme:FilteringRequestedUnavailable

Fault Subcode	wsme:FilteringRequestedUnavailable
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The requested filter dialect is not supported.
Detail	<s:Detail> <wsme:SupportedDialect> ..</wsme:SupportedDialect> + <wsman:FaultDetail> ..the following URI, if applicable </wsman:FaultDetail> </s:Detail> Possible URI value: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired
Comments	This fault is returned when the client requests a filter dialect not supported by the service. In some cases, a subscription <i>requires</i> a filter because the result of an unfiltered subscription may be infinite or extremely large. In these cases, the URI http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired needs to be included in the s:Detail element.
Applicability	Subscribe
Remedy	The client switches to a supported filter dialect or uses no filtering.

5979

Table 22 – wsman:FragmentDialectNotSupported

Fault Subcode	wsman:FragmentDialectNotSupported
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The requested fragment filtering dialect or language is not supported.
Detail	<s:Detail> <wsman:FragmentDialect> xs:anyURI </wsman:FragmentDialect> <wsman:FragmentDialect> xs:anyURI </wsman:FragmentDialect> </s:Detail> The preceding optional URI values indicate supported dialects.
Comments	This fault is returned when the service does not support the requested fragment-level filtering dialect. If the implementation supports the fragment dialect in general, but not for the specific resource, this fault is still returned. Other resources might support the fragment dialect. This fault does not imply that the fragment dialect is not supported by the implementation.
Applicability	Enumerate, Get, Create, Put, Delete
Remedy	The client uses a supported filtering dialect or no filtering.

5980

Table 23 – wsman:InternalError

Fault Subcode	wsman:InternalError
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Receiver
Reason	The service cannot comply with the request due to internal processing errors.
Detail	<s:Detail> ...service-specific extension XML elements.... </s:Detail>
Comments	This fault is a generic error for capturing internal processing errors within the service. For example, this is the correct fault if the service cannot load necessary executable images, its configuration is corrupted, hardware is not operating properly, or any unknown or unexpected internal errors occur. It is expected that the service needs to be reconfigured, restarted, or reinstalled, so merely asking the client to retry will not succeed.
Applicability	All messages
Remedy	The client repairs the service out-of-band to WS-Management.

5981

Table 24 – wsman:InvalidBookmark

Fault Subcode	wsman:InvalidBookmark
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The bookmark supplied with the subscription is not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: The service is not able to back up and replay from that point: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Expired The service is not able to decode the bookmark: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFormat
Comments	This fault is returned if a bookmark has expired, is corrupt, or is otherwise unknown.
Applicability	Subscribe
Remedy	The client issues a new subscription without any bookmarks or locates the correct bookmark.

5982

Table 25 – wsmen:InvalidEnumerationContext

Fault Subcode	wsmen:InvalidEnumerationContext
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Receiver
Reason	The supplied enumeration context is invalid.
Detail	None
Comments	An invalid enumeration context was supplied with the message. Typically, this fault will happen with Pull. The enumeration context may be invalid due to expiration, an invalid format, or reuse of an old context no longer being tracked by the service. The service also can return this fault for any case where the enumerator has been terminated unilaterally on the service side, although one of the more descriptive faults is preferable, because this usually happens on out-of-memory errors (wsman:QuotaLimit), authorization failures (wsman:AccessDenied), or internal errors (wsman:InternalError).
Applicability	Pull, Release (whether a pull-mode subscription, or a normal enumeration)
Remedy	The client abandons the enumeration and lets the service time it out, because Release will fail as well.

5983

Table 26 – wsme:InvalidExpirationTime

Fault Subcode	wsme:InvalidExpirationTime
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The expiration time is not valid.
Detail	None
Comments	The expiration time is not valid at all or within the limits of the service. This fault is used for outright errors (expirations in the past, for example) or expirations too far into the future. If the service does not support expiration times at all, a wsman:UnsupportedFeature fault can be returned with the correct detail code.
Applicability	Subscribe
Remedy	The client issues a new subscription with a supported expiration time.

5984

Table 27 – wsmen:InvalidExpirationTime

Fault Subcode	wsmen:InvalidExpirationTime
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The expiration time is not valid.
Detail	None
Comments	Because WS-Management recommends against implementing the Expiration feature, this fault might not occur with most implementations. See clause 8 for more information.
Applicability	Enumerate
Remedy	Not applicable

5985

Table 28 – wsme:InvalidMessage

Fault Subcode	wsme:InvalidMessage
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The request message has unknown or invalid content and cannot be processed.
Detail	None
Comments	This fault is generally not used in WS-Management, although it can be used for cases not covered by other faults. If the content violates the schema, a wsman:SchemaValidationError fault can be sent. If specific errors occur in the subscription body, one of the more descriptive faults can be used. This fault is not to be used to indicate unsupported features, only unexpected or unknown content in violation of this specification.
Applicability	Pub/sub request messages
Remedy	The client issues valid messages that comply with this specification.

5986

Table 29 – wsa:InvalidMessageInformationHeader

Fault Subcode	wsa:InvalidMessageInformationHeader
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	A message information header is not valid, and the message cannot be processed.
Detail	<s:Detail> ...the invalid header... </s:Detail>
Comments	This fault can occur with any type of SOAP header error. The header might be invalid in terms of schema or value, or it might constitute a semantic error. This fault is not to be used to indicate an invalid resource address (a "not found" condition for the resource), but to indicate actual structural violations of the SOAP header rules in this specification. Examples are repeated MessageIDs, missing RelatesTo on a response, badly formed addresses, or any other missing header content.
Applicability	All messages
Remedy	The client reformats message using the correct format, values, and number of message information headers.

5987

Table 30 – wsman:InvalidOptions

Fault Subcode	wsman:InvalidOptions
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	One or more options are not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/NotSupported http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue
Comments	This fault generically covers all cases where the option names or values are not valid, or they are used in incorrect combinations.
Applicability	All request messages
Remedy	The client discovers supported option names and valid values by consulting metadata or other mechanisms. Such metadata is beyond the scope of this specification.

5988

Table 31 – wsman:InvalidParameter

Fault Subcode	wsman:InvalidParameter
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	An operation parameter is not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidName
Comments	This fault is returned when a parameter to a custom action is not valid. This fault is a default for new implementations that need to have a generic fault for this case. The method can also return any specific fault of its own.
Applicability	All messages with custom actions
Remedy	The client consults the WSDL for the operation and determines how to supply the correct parameter.

5989

Table 32 – wsmt:InvalidRepresentation

Fault Subcode	wsmt:InvalidRepresentation
Action URI	http://schemas.xmlsoap.org/ws/2004/09/transfer/fault
Code	s:Sender
Reason	The XML content is not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MissingValues http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidFragment
Comments	This fault may be returned when the input XML is not valid semantically or uses the wrong schema for the resource. However, a wsman:SchemaValidationError fault can be returned if the error is related to XML schema violations as such, as opposed to invalid semantic values. Note the anomalous case in which a schema violation does not occur, but the namespace is simply the wrong one; in this case, http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidNamespace is returned.
Applicability	Put, Create
Remedy	The client corrects the request XML.

5990

Table 33 – wsman:InvalidSelectors

Fault Subcode	wsman:InvalidSelectors
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The selectors for the resource are not valid.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsufficientSelectors http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/UnexpectedSelectors http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/TypeMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InvalidValue http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DuplicateSelectors
Comments	This fault covers all cases where the specified selectors were incorrect or unknown for the specified resource.
Applicability	All request messages
Remedy	The client retrieves documentation or metadata and corrects the selectors.

5991

Table 34 – wsa:MessageInformationHeaderRequired

Fault Subcode	wsa:MessageInformationHeaderRequired
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	A required header is missing.
Detail	<s:Detail> The XML QName of the missing header </s:Detail>
Comments	A required message information header (To, MessageID, or Action) is not present.
Applicability	All messages
Remedy	The client adds the missing message information header.

5992

Table 35 – wsman:NoAck

Fault Subcode	wsman:NoAck
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The receiver did not acknowledge the event delivery.
Detail	None
Comments	This fault is returned when the client (subscriber) receives an event with a wsman:AckRequested header and does not (or cannot) acknowledge the receipt. The service stops sending events and terminates the subscription.
Applicability	Any event delivery action (including heartbeats, dropped events, and so on) in any delivery mode
Remedy	For subscribers, the subscription is resubmitted without the acknowledgement option. For services delivering events, the service cancels the subscription immediately.

5993

Table 36 – wsman:QuotaLimit

Fault Subcode	wsman:QuotaLimit
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The service is busy servicing other requests.
Detail	None
Comments	This fault is returned when the SOAP message is otherwise correct, but the service has reached a resource or quota limit.
Applicability	All messages
Remedy	The client can retry later.

5994

Table 37 – wsman:SchemaValidationError

Fault Subcode	wsman:SchemaValidationError
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The supplied SOAP violates the corresponding XML schema definition.
Detail	None
Comments	This fault is used for any XML parsing failure or schema violations. Full validation of the SOAP against schemas is not expected in real-time, but processors might in fact notice schema violations, such as type mismatches. In all of these cases, this fault applies. In debugging modes where validation is occurring, this fault can be returned for <i>all</i> errors noted by the validating parser.
Applicability	All messages
Remedy	The client corrects the message.

5995

Table 38 – wsmen:TimedOut

Fault Subcode	wsmen:TimedOut
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Receiver
Reason	The enumerator has timed out and is no longer valid.
Detail	None
Comments	This fault is not to be used in WS-Management due to overlap with wsman:TimedOut, which covers all the other messages.
Applicability	Pull
Remedy	The client can retry the Pull request.

5996

Table 39 – wsman:TimedOut

Fault Subcode	wsman:TimedOut
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Receiver
Reason	The operation has timed out.
Detail	None
Comments	The operation could not be completed within the wsman:OperationTimeout value, or an internal override timeout was reached by the service while trying to process the request. This fault is also returned in all enumerations when no content is available for the current Pull request. Clients can simply retry the Pull request again until a different fault is returned.
Applicability	All requests
Remedy	The client can retry the operation. If the operation is a write (delete, create, or custom operation), the client can consult the system operation log before blindly attempting a retry or attempt a Get or other read operation to try to discover the result of the previous operation.

5997

Table 40 – wsme:UnableToRenew

Fault Subcode	wsme:UnableToRenew
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The subscription could not be renewed.
Detail	None
Comments	This fault is returned in all cases where the subscription cannot be renewed but is otherwise valid.
Applicability	wsme:Renew
Remedy	The client issues a new subscription.

5998

Table 41 – wsme:UnsupportedExpirationType

Fault Subcode	wsme:UnsupportedExpirationType
Action URI	http://schemas.xmlsoap.org/ws/2004/08/eventing/fault
Code	s:Sender
Reason	The specified expiration type is not supported.
Detail	None
Comments	A specific time for expiration (as opposed to duration) is not supported. This fault is not to be used if the value itself is incorrect; it is only to be used if the <i>type</i> is not supported.
Applicability	Subscribe
Remedy	The client corrects the expiration to use a duration time.

5999

Table 42 – wsmen:UnsupportedExpirationType

Fault Subcode	wsmen:UnsupportedExpirationType
Action URI	http://schemas.xmlsoap.org/ws/2004/09/enumeration/fault
Code	s:Sender
Reason	The specified expiration type is not supported.
Detail	None
Comments	The specified expiration type is not supported. For example, a specific time-based expiration type might not be supported (as opposed to a duration-based expiration type). This fault is not to be used if the value itself is incorrect; it is only to be used if the <i>type</i> is not supported.
Applicability	Enumerate
Remedy	The client corrects the expiration time or omits it and retries.

6000

Table 43 – wsman:UnsupportedFeature

Fault Subcode	wsman:UnsupportedFeature
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The specified feature is not supported.
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> Possible URI values: http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Ack http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AddressingMode http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/AsynchronousRequest http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Bookmarks http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/DeliveryRetries http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EnumerationMode http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/ExpirationTime http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FilteringRequired http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FormatMismatch http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/FragmentLevelAccess http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Heartbeats http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/InsecureAddress http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Locale http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxElements http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopePolicy http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxEnvelopeSize http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/MaxTime http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/OperationTimeout
Comments	This fault indicates that an unsupported feature was attempted.
Applicability	Any message
Remedy	The client corrects or removes the unsupported feature request and retries.

6001

Table 44 – wsme:UnsupportedExpirationType

Fault Subcode	wsme:UnsupportedExpirationType
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	Only expiration durations are supported.
Detail	None
Comments	This fault is sent when a Subscribe request specifies an expiration time and the event source is only capable of accepting expiration durations; for instance, if the event source does not have access to absolute time.
Applicability	Subscribe, wsme:Renew
Remedy	

6002

Table 45 – wsmen:UnableToRenew

Fault Subcode	wsmen:UnableToRenew
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	<i>Text explaining the failure; e.g., "The event source has too many subscribers".</i>
Detail	None
Comments	This fault is sent when the event source is not capable of fulfilling a Renew request for local reasons unrelated to the specific request.
Applicability	wsmen:Renew
Remedy	

6003

Table 46 – wsa:InvalidMessage

Fault Subcode	wsa:InvalidMessage
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	<i>The message is not valid and cannot be processed.</i>
Detail	<i>The invalid message</i>
Comments	If a request message does not comply with the corresponding outline in the previous row, the request shall fail and the event source or subscription manager may generate this fault indicating that the request is invalid.
Applicability	Subscribe, Renew, wsme:GetStatus, Unsubscribe
Remedy	

6004

Table 47 – wsme:CannotProcessFilter

Fault Subcode	wsme:CannotProcessFilter
Action URI	http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
Code	s:Sender
Reason	<i>Cannot filter as requested</i>
Detail	None
Comments	A filter was specified can not be processed.
Applicability	Subscribe
Remedy	

6005

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013

ANNEX A (informative)

Notational Conventions

6010 This annex specifies the notations and namespaces used in this specification.

6011 This specification uses the following syntax to define normative outlines for messages:

- 6012 • The syntax appears as an XML instance, but values in italics indicate data types instead of
6013 values.
- 6014 • Characters are appended to elements and attributes to indicate cardinality:
6015 – "?" (0 or 1)
6016 – "*" (0 or more)
6017 – "+" (1 or more)
- 6018 • The character "|" indicates a choice between alternatives.
- 6019 • The characters "[" and "]" indicate that enclosed items are to be treated as a group with
6020 respect to cardinality or choice.
- 6021 • An ellipsis ("...") indicates a point of extensibility that allows other child or attribute content.
6022 Additional children and attributes may be added at the indicated extension points but must
6023 not contradict the semantics of the parent or owner, respectively. If a receiver does not
6024 recognize an extension, the receiver should not process the message and may fault.
- 6025 • XML namespace prefixes (see Table A-1) indicate the namespace of the element being
6026 defined.

6027 Throughout the document, whitespace within XML element values is used for readability. In practice,
6028 a service can accept and strip leading and trailing whitespace within element values as if whitespace
6029 had not been used.

6030 A.1 XML Namespaces

6031 Table A-1 lists XML namespaces used in this specification. The choice of any namespace prefix is
6032 arbitrary and not semantically significant. Unless otherwise noted, the XML Schema for each
6033 specification can be retrieved by resolving the XML namespace URI for each specification listed in
6034 Table A-1.

6035

Table A-1 – Prefixes and XML Namespaces Used in This Specification

Prefix	XML Namespace	Specification
wsman	http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd	This specification
wsmid	http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd	This specification – discovery of supported protocol versions
s	http://www.w3.org/2003/05/soap-envelope	<u>SOAP 1.2</u>
xs	http://www.w3.org/2001/XMLSchema	<u>XML Schema 1</u> , <u>XML Schema 2</u>
wsdl	http://schemas.xmlsoap.org/wsdl	WSDL/1.1
wsa	Either wsa04 or wsa10	Either wsa04 or wsa10
wsa04	http://schemas.xmlsoap.org/ws/2004/08/addressing	Clause 5 of this specification
wsa10	http://www.w3.org/2005/08/addressing	<u>WS-Addressing W3C Recommendation</u>
wsam	http://www.w3.org/2007/05/addressing/metadata	<u>WS-Addressing Metadata W3C Recommendation</u>
wsme	http://schemas.xmlsoap.org/ws/2004/08/eventing	Clause 10 of this specification
wsmen	http://schemas.xmlsoap.org/ws/2004/09/enumeration	Clause 8 of this specification
wsmt	http://schemas.xmlsoap.org/ws/2004/09/transfer	Clause 7 of this specification
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	<u>WS-Policy</u>

6036

6037

ANNEX B

6038

(normative)

6039

6040

Conformance

6041 This annex specifies the conformance rules used in this specification.

6042 An implementation is not conformant with this specification if it fails to satisfy one or more of the
6043 "shall" or "required" level requirements defined in the conformance rules for each section, as indicated
6044 by the following format:

6045 **Rnnnn**: Rule text

6046 General conformance rules are defined as follows:

6047 **RB-1:** To be conformant, the service shall comply with all the rules defined in this
6048 specification. Items marked with shall are required, and items marked with should are highly
6049 advised to maximize interoperation. Items marked with may indicate the preferred implementation
6050 for expected features, but interoperation is not affected if they are ignored.

6051 **RB-2:** Conformant services of this specification shall use this XML namespace Universal
6052 Resource Identifier:

6053 (1) <http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd>

6054 **RB-3:** A SOAP node shall not use the XML namespace identifier for this specification unless it
6055 complies with the conformance rules in this specification.

6056 This specification does not mandate that all messages and operations need to be supported. It only
6057 requires that any supported message or operation obey the conformance rules for that message or
6058 operation. It is important that services not use the XML namespace identifier for WS-Management in
6059 SOAP operations in a manner that is inconsistent with the rules defined in this specification.

ANNEX C (normative)

HTTP(S) Transport and Security Profile

C.1 General

Although WS-Management is a SOAP protocol and not tied to a specific network transport, interoperation requires some common standards to be established. This clause centers on establishing common usage over HTTP 1.1 and HTTPS. In addition to HTTP and HTTPS, this specification allows any SOAP-enabled transport to be used as a carrier for WS-Management messages.

For identification and referencing, each transport is identified by a URI, and each authentication mechanism defined in this specification is also identified by a URI.

As new transports are standardized, they can also acquire a URI for referencing purposes, and any new authentication mechanisms that they expose can also be assigned URIs for publication and identification purposes in XML documents. As new transports are standardized for WS-Management, the associated transport-specific requirements can be defined and published to ensure interoperability.

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

The SOAP HTTP binding described in section 7 of [SOAP Version 1.2 Part 2: Adjuncts](#) is used for WS-Management encoding over HTTP and HTTPS.

C.2 HTTP(S) Binding

This clause clarifies how SOAP messages are bound to HTTP(S).

RC.2-1: A service that supports the SOAP HTTP(S) binding shall at least support it using HTTP 1.1.

RC.2-2: A service shall at least implement the Responding SOAP Node of the SOAP Request-Response Message Exchange Pattern:

<http://www.w3.org/2003/05/soap/mep/request-response/>

RC.2-3: A service may choose not to implement the Responding SOAP Node of the SOAP Response Message Exchange Pattern:

<http://www.w3.org/2003/05/soap/mep/soap-response/>

RC.2-4: A service may choose not to support the SOAP Web Method Feature.

RC.2-5: A service shall at least implement the Responding SOAP Node of an HTTP one-way Message Exchange Pattern where the SOAP Envelope is carried in the HTTP Request and the HTTP Response has a Status Code of 202 Accepted and an empty Entity Body (no SOAP Envelope).

The message exchange pattern described in RB.2-5 is used to carry SOAP messages that require no response.

- 6098 **RC.2-6:** A service shall at least support Request Message SOAP Envelopes and one-way
6099 SOAP Envelopes delivered using HTTP Post.
- 6100 **RC.2-7:** In cases where the service cannot respond with a SOAP message, the HTTP error
6101 code 500 (Internal Server Error) should be returned and the client side should close the
6102 connection.
- 6103 **RC.2-8:** For services that support HTTPS (TLS 1.0), the service shall at least implement
6104 TLS_RSA_WITH_RC4_128_SHA. It is recommended that the service also support
6105 TLS_RSA_WITH_AES_128_CBC_SHA.
- 6106 **RC.2-9:** When delivering faults, an HTTP status code of 500 should be used in the response
6107 for s:Receiver faults, and a code of 400 should be used for s:Sender faults.
- 6108 **RC.2-10:** The URL used with the HTTP-Post operation to deliver the SOAP message is not
6109 required to have the same content as the wsa:To URI used in the SOAP address. Often, the
6110 HTTP URL has the same content as the wsa:To URI in the message, but may additionally contain
6111 other message routing fields suffixed to the network address using a service-defined separator
6112 token sequence. It is recommended that services require only the wsa:To network address URL
6113 to promote uniform client-side processing and behavior, and to include service-level routing in
6114 other parts of the address.
- 6115 **RC.2-11:** In the absence of other requirements, it is recommended that the path portion of the
6116 URL used with the HTTP-POST operation be /wsman for resources that require authentication
6117 and /wsman-anon for resources that do not require authentication. If these paths are used,
6118 unauthenticated requests should not be supported for /wsman and authentication must not be
6119 required for /wsman-anon.
- 6120 **RC.2-12:** If the SOAPAction header is present in an HTTP/HTTPS-based request that carries a
6121 SOAP message, it must match the wsa:Action URI present in the SOAP message. The
6122 SOAPAction header is optional, and a service must not fault a request if this header is missing.
- 6123 Because WS-Management is based on SOAP 1.2, the optional SOAPAction header is merely
6124 used as an optimization. If present, it shall match the wsa:Action URI used in the SOAP
6125 message. The service is permitted to fault the request by simply examining the SOAPAction
6126 header, if the action is not valid, without examining the SOAP content. However, the service may
6127 not fault the request if the SOAPAction header is omitted.
- 6128 **RC.2-13:** If a service supports attachments, the service shall support the HTTP Transmission
6129 Optimization Feature.
- 6130 **RC.2-14:** If a service cannot process a message with an attachment or unsupported encoding
6131 type, and the transport is HTTP or HTTPS, it shall return HTTP error 415 as its response
6132 (unsupported media).
- 6133 **RC.2-15:** If a service cannot process a message with an attachment or unsupported encoding
6134 type using transports other than HTTP/HTTPS, it should return a wsman:EncodingLimit fault with
6135 the following detail code:
- 6136 <http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/EncodingType>

C.3 HTTP(S) Security Profiles

This specification defines a set of security profiles for use with HTTP and HTTPS. Conformant services need not support HTTP or HTTPS, but if supported these predefined profiles provide the client with at least one way to access the service. Other specifications can define additional profiles for use with HTTP or HTTPS.

RC.3-1: A conformant service that supports HTTP shall support one of the predefined HTTP-based profiles.

RC.3-2: A conformant service that supports HTTPS shall support one of the predefined HTTPS-based profiles.

RC.3-3: A conformant service should not expose WS-Management over a completely unauthenticated HTTP channel except for situations such as Identify (see clause 11), debugging, or as determined by the service.

The service is not required to export only a single HTTP or HTTPS address. The service can export multiple addresses, each of which supports a specific security profile or multiple profiles.

If clients support all predefined profiles, they are assured of some form of secure access to a WS-Management implementation that supports HTTP, HTTPS, or both.

C.3.1 <http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic>

This profile is essentially the "standard" profile, but it is limited to Basic authentication.

The typical sequence is shown in Table C-1.

Table C-1 – Basic Authentication Sequence

	Client		Service
1	Client connects with no authorization header.	→	Service sees no header.
2		←	Service sends 401 return code, listing Basic as the authorization mode.
3	Client provides Basic authorization header.	→	Service authenticates the client.

This behavior is normal for HTTP. If the client connects with a Basic authorization header initially and if it is valid, the request immediately succeeds.

Basic authentication is not recommended for unsecured transports. If used with HTTP alone, for example, the transmission of the password constitutes a security risk. However, if the HTTP transport is secured with IPSec, for example, the risk is substantially reduced.

Similarly, Basic authentication is suitable when performing testing, prototyping, or diagnosis.

6163 **C.3.2 <http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest>**

6164 This profile is essentially the same as the "standard" profile, but it is limited to the use of Digest
6165 authentication.

6166 The typical sequence is shown in Table C-2.

6167 **Table C-2 – Digest Authentication Sequence**

	Client		Service
1	Client connects with no authorization header.	→	Service sees no header.
2		←	Service sends 401 return code, listing Digest as the authorization mode.
3	Client provides Digest authorization header.	→	
4		←	Service begins authorization sequence of secure token exchange.
5	Client continues authorization sequence.	→	Service authenticates client.

6168 This behavior is normal for HTTP. If the client connects with a Digest authorization header initially and
6169 if it is valid, the token exchange sequence begins.

6170 **C.3.3 <http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/basic>**

6171 This profile establishes the use of Basic authentication over HTTPS. This profile is used when only a
6172 server-side certificate encrypts the connection, but the service still needs to authenticate the client.

6173 The typical sequence is shown in Table C-3.

6174 **Table C-3 – Basic Authentication over HTTPS Sequence**

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service sees no header, but establishes an encrypted connection.
2		←	Service sends 401 return code, listing Basic as the authorization mode.
3	Client provides Basic authorization header.	→	Service authenticates the client.

6175 If the client connects with a Basic authorization header initially and if it is valid, the request
6176 immediately succeeds.

C.3.4 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/digest

This profile establishes the use of Digest authentication over HTTPS. This profile is used when only a server-side certificate encrypts the connection, but the service still needs to authenticate the client.

The typical sequence is shown in Table C-4.

Table C-4 – Digest Authentication over HTTPS Sequence

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service sees no header, but establishes an encrypted connection.
2		←	Service sends 401 return code, listing Digest as the auth mode.
3	Client provides Digest authorization header.	→	
4		←	Service begins authorization sequence of secure token exchange.
5	Client continues authorization sequence.	→	Service authenticates client.

This behavior is normal for HTTPS. If the client connects with a Digest authorization header initially and if it is valid, the token exchange sequence begins.

C.3.5 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual

In this security mode, the client supplies an X.509 certificate that is used to authenticate the client. No HTTP or HTTPS authorization header is required in the HTTP-Post request.

However, as a hint to the service, the following HTTP/HTTPS authorization header may be present.

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

Because the service can be configured to always look for the certificate, this authorization header is not required.

This simple sequence is shown in Table C-5.

Table C-5 – HTTPS with Client Certificate Sequence

	Client		Service
1	Client connects with no authorization header but supplies an X.509 certificate.	→	Service ignores the authorization header and retrieves the client-side certificate used in the TLS 1.0 handshake.
2		←	Service accepts or denies access with 403.7 or 403.16 return codes.

C.3.6 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic

In this profile, the http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual profile is used first to authenticate both sides using X.509 certificates. Individual operations are subsequently authenticated using HTTP Basic authorization headers.

6199 This profile authenticates both the client and service initially and provides one level of security,
 6200 typically at the machine or device level. The second level of authentication typically performs
 6201 authorization for specific operations, although it can act as a simple, secondary authentication
 6202 mechanism with no authorization semantics.

6203 The typical sequence is shown in Table C-6.

6204 **Table C-6 – Basic Authentication over HTTPS with Client Certificate Sequence**

	Client		Service
1	Client connects with certificate and special authorization header.	→	Service queries for client certificate and authenticates. If certificate is missing or invalid, the sequence stops here with 403.7 or 403.16 return codes.
2		←	After authenticating the certificate, the service sends 401 return code, listing available Basic authorization mode as a requirement.
3	Client selects Basic as the authorization mode to use and includes it in the Authorization header, as defined for HTTP 1.1.	→	Service authenticates the client again before performing the operation.

6205 In the initial request, the HTTPS authorization header must be as follows:

6206 Authorization: http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic

6207 This indicates to the service that this special mode is in use, and that it can query for the client
 6208 certificate to ensure that subsequent requests are properly challenged for Basic authorization if the
 6209 HTTP Authorization header is missing from a request.

6210 The Authorization header is treated as normal HTTP basic:

6211 Authorization: Basic ...user/password encoding

6212 This use of Basic authentication is secure (unlike its normal use in HTTP) because the transmission
 6213 of the user name and password is performed over a TLS 1.0 encrypted connection.

6214 **C.3.7 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/ 6215 mutual/digest**

6216 This profile is the same as
 6217 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic, except that the HTTP
 6218 Digest authentication model is used after the initial X.509 certificate-based mutual authentication is
 6219 completed.

6220 In the initial request, the HTTPS authorization header must be as follows:

6221 Authorization:
 6222 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/digest

6223 **C.3.8 http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/ 6224 spnego-kerberos**

6225 In this profile, the client connects to the server using HTTPS with only server-side certificates to
 6226 encrypt the connection.

6227 Authentication is carried out based on [RFC 4559](#), which describes the use of GSSAPI SPNEGO over
 6228 HTTP (Table C-7). This mechanism allows HTTP to carry out the negotiation protocol of [RFC 4178](#) to
 6229 authenticate the user based on Kerberos Version 5.

6230 **Table C-7 – SPNEGO Authentication over HTTPS Sequence**

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service sees no header, but establishes an encrypted connection.
2		←	Service sends 401 return code, listing Negotiate as an available HTTP authentication mechanism.
3	Client uses the referenced Internet draft to start a SPNEGO sequence to negotiate for Kerberos V5.	→	...
4	...	←	Service engages in SPNEGO sequence to authenticate client using Kerberos V5.
5	Client is authenticated.	→	Service authenticates client.

6231 **C.3.9 [http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/](http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/spnego-kerberos)**
 6232 **[mutual/spnego-kerberos](http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/spnego-kerberos)**

6233 This mode is the same as [http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego-](http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego-kerberos)
 6234 [kerberos](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
 6235 beginning the Kerberos authentication sequence (Table C-8). See [RFC 4178](#) for details.

6236 **Table C-8 – SPNEGO Authentication over HTTPS with Client Certificate Sequence**

	Client		Service
1	Client connects with no authorization header using HTTPS.	→	Service queries for client certificate and authenticates. If certificate is missing or invalid, the sequence stops here with 403.7 or 403.16 return codes.
2		←	After the mutual certificate authentication sequence, service sends 401 return code, listing Negotiate as an available HTTP authentication mechanism.
3	Client uses the referenced Internet draft to start a SPNEGO sequence to negotiate for Kerberos V5.	→	...
4	...	←	Service engages in SPNEGO sequence to authenticate client using Kerberos V5.
5	Client is authenticated.	→	Service authenticates client.

6237 Typically, this is used to mutually authenticate devices or machines, and then subsequently perform
 6238 user- or role-based authentication.

6239 **C.3.10 <http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/spnego>**
6240 **-kerberos**

6241 This profile is the same as <http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/spnego>-
6242 kerberos except that it is performed over an HTTP connection. See [RFC 4178](#) for details.

6243 Although this profile supports secure authentication, because it is not encrypted, it represents security
6244 risks such as information disclosure because the SOAP traffic is in plain text. It is not to be used in
6245 environments that require a high level of security.

6246 **C.4 IPSec and HTTP**

6247 HTTP with Basic authentication is weak on an unsecured network. If IPSec is in use, however, this
6248 weakness is no longer an issue. IPSec provides high-quality cryptographic security, data origin
6249 authentication, and anti-replay services.

6250 Because IPSec is intended for machine-level authentication and network traffic protection, it is
6251 insufficient for real-world management in many cases, which can require additional authentication of
6252 specific users to authorize access to resource classes and instances. IPSec needs to be used in
6253 conjunction with one of the profiles in this clause for user-level authentication. However, it obviates
6254 the need for HTTPS-based traffic and allows safe use of HTTP-based profiles.

6255 From the network perspective, the use of HTTP Basic authentication when the traffic is carried over a
6256 network secured by IPSec is intrinsically safe and equivalent to using HTTPS with server-side
6257 certificates. For example, the wsman security profile
6258 <http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual/basic> (using HTTPS) is
6259 equivalent to simple <http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/basic> (using
6260 HTTP) if the traffic is actually secured by IPSec.

6261 Other specifications can define IPSec security profiles that combine IPSec with appropriate
6262 authentication mechanisms.

IECNORM.COM : Click to view the full PDF of ISO/IEC 17963:2013

6263
6264
6265
6266

ANNEX D (informative)

XPath Support

6267 D.1 General

6268 Implementations typically need to support XPath for several purposes, such as fragment-level access
6269 (7.7), datasets (8), and filtering (10.2.2). Because the full [XPath 1.0](#) specification is large, subsets are
6270 typically required in resource-constrained implementations.

6271 The purpose of this clause is to identify the minimum set of syntactic elements that implementations
6272 can provide to promote maximum interoperability. In most cases, implementations provide large
6273 subsets of full XPath, but they need additional definitions to ensure that the subsets meet minimum
6274 requirements. The Level 1 and Level 2 BNF definitions in this annex establish such minimums for use
6275 in the WS-Management space.

6276 This specification defines two subset profiles for XPath: Level 1 with basic node selector support and
6277 no filtering (for supporting Fragment-level access as described in 7.7), and Level 2 with basic filtering
6278 support (for enumerating and receiving notifications). Level 2 is a formal superset of Level 1.

6279 The following BNFs both are formal LL(1) grammars. A parser can be constructed automatically from
6280 the BNF using an appropriate tool, or a recursive-descent parser can be implemented manually by
6281 inspection of the grammar.

6282 Within the grammars, non-terminal tokens are surrounded by angled brackets, and terminal tokens
6283 are in uppercase and not surrounded by angled brackets.

6284 XML namespace support is explicitly absent from these definitions. Processors that meet the syntax
6285 requirements can provide a mode in which the elements are processed without regard to XML
6286 namespaces, but can also provide more powerful, namespace-aware processing.

6287 The default execution context of the XPath is specified explicitly in 8.4 and 10.2.2.

6288 For the following dialects, XML namespaces and QNames are not expected to be supported by
6289 default and can be silently ignored by the implementation.

6290 These dialects are for informational purposes only and are not intended as Filter Dialects in actual
6291 SOAP messages. Because they are XPath compliant (albeit subsets), the Filter Dialect in the SOAP
6292 messages is still that of full XPath:

6293 <http://www.w3.org/TR/1999/REC-xpath-19991116>

6294 **D.2 Level 1**

6295 Level 1 contains just the necessary XPath to identify nodes within an XML document or fragment and
 6296 is targeted for use with Fragment-level access (7.7) of this specification.

6297 EXAMPLE:

```

6298 (1) <path> ::= <root_selector> TOKEN_END_OF_INPUT;
6299 (2) <root_selector> ::= TOKEN_SLASH <element_sequence>;
6300 (3) <root_selector> ::= <attribute>;
6301 (4) <root_selector> ::= <relpath> <element_sequence>;
6302 (5) <root_selector> ::= TOKEN_DOT
6303 (6) <relpath> ::= <>;
6304 (7) <relpath> ::= TOKEN_DOT TOKEN_SLASH;
6305 (8) <relpath> ::= TOKEN_DOT_DOT TOKEN_SLASH;
6306 (9) <element_sequence> ::= <element> <optional_filter_expression> <more>;
6307 (10) <more> ::= TOKEN_SLASH <follower>;
6308 (11) <more> ::= <>;
6309 (12) <follower> ::= <attribute>;
6310 (13) <follower> ::= <text_function>;
6311 (14) <follower> ::= <element_sequence>;
6312 (15) <optional_filter_expression> ::=
6313 (16)   TOKEN_OPEN_BRACKET <filter_expression> TOKEN_CLOSE_BRACKET;
6314 (17) <optional_filter_expression> ::= <>;
6315 (18) <attribute> ::= TOKEN_AT_SYMBOL <name>;
6316 (19) <element> ::= <name>;
6317 (20) <text_function> ::=
6318 (21)   TOKEN_TEXT TOKEN_OPEN_PAREN TOKEN_CLOSE_PAREN;
6319 (22) <name> ::= TOKEN_XML_NAME;
6320 (23) <filter_expression> ::= <array_location>;
6321 (24) <array_location> ::= TOKEN_UNSIGNED_POSITIVE_INTEGER;

```

6322 This dialect allows selecting any XML node based on its name or array position, or any attribute by its
 6323 name. Optionally, the text() NodeTest can trail the entire expression to select only the raw value of
 6324 the name, excluding the XML element name wrapper.

6325 Terminals in the grammar are defined as shown in Table D-1.

6326 **Table D-1 – XPath Level 1 Terminals**

TOKEN_SLASH	The character '/'
TOKEN_DOT	The character '.'
TOKEN_DOT_DOT	The characters '..'
TOKEN_END_OF_INPUT	End of input
TOKEN_OPEN_BRACKET	The character '['
TOKEN_CLOSE_BRACKET	The character ']'
TOKEN_AT_SYMBOL	The character '@'
TOKEN_XML_NAME	Equivalent to XML Schema type xs:token
TOKEN_UNSIGNED_POSITIVE_INTEGER	Values in the subrange 1..4294967295
TOKEN_TEXT	The characters 'text'
TOKEN_OPEN_PAREN	The character '('
TOKEN_CLOSE_PAREN	The character ')'

6327 Using the following XML fragment, some examples are shown assuming that the element "a" is the
 6328 context node (that is, represents the resource or event document).

6329 **EXAMPLE 1:**

```

6330 (1) <Envelope>
6331 (2)   <Body>
6332 (3)     <a>
6333 (4)       <b x="y"> 100 </b>
6334 (5)       <c>
6335 (6)         <d> 200 </d>
6336 (7)       </c>
6337 (8)       <c>
6338 (9)         <d> 300 </d>
6339 (10)        <d> 400 </d>
6340 (11)      </c>
6341 (12)    </a>
6342 (13)  </Body>
6343 (14) </Envelope>

```

6344 **EXAMPLE 2:**

```

6345 (1) / // Selects <a> and all its content
6346 (2) /a // Selects <a> and all its content
6347 (3) . // Selects <a> and all its content
6348 (4) ../a // Selects <a> and all its content
6349 (5) b // Selects <b x="y"> 100 </b>
6350 (6) c // Selects both <c> nodes, one after the other
6351 (7) c[1] // Selects <c><d>200</d></c>
6352 (8) c[2]/d[2] // Selects <d> 400 </d>
6353 (9) c[2]/d[2]/text() // Selects 400
6354 (10) b/text() // Selects 100
6355 (11) b/@x // Selects x="y"

```