

Web Services for Management (WS-Management)

February 2005

Authors

Akhil Arora, Sun Microsystems, Inc.
Josh Cohen, Microsoft Corporation.
Jim Davis, WBEM Solutions, Inc.
Eugene Golovinsky, BMC Software, Inc.
Jackson He, Intel Corporation
David Hines, Intel Corporation
Raymond McCollum, Microsoft Corporation (Editor)
Milan Milenkovic, Intel Corporation
Paul Montgomery, Advanced Micro Devices, Inc.
Jeffrey Schlimmer, Microsoft Corporation.
Enoch Suen, Dell Inc.
Vijay Tewari, Intel Corporation

Copyright Notice

(c) 2004, 2005 [Advanced Micro Devices, Inc.](#), [BMC Software, Inc.](#), [Dell, Inc.](#), [Intel Corporation](#), [Microsoft Corporation](#), [Sun Microsystems, Inc.](#), and [WBEM Solutions, Inc.](#) All rights reserved.

Permission to copy and display WS-Management, which includes its associated WSDL and Schema files and any other associated metadata (the "Specification"), in any medium without fee or royalty is hereby granted, provided that you include the following on ALL copies of the Specification that you make:

1. A link or URL to the Specification at one of the Co-Developers' websites.
2. The copyright notice as shown in the Specification.

Microsoft, Intel, AMD, Dell, BMC, WBEM Solutions and Sun (collectively, the "Co-Developers") each agree upon request to grant you a license, provided you agree to be bound by such license, under royalty-free and otherwise reasonable, non-discriminatory terms and conditions to their respective patent claims that would necessarily be infringed by an implementation of the Specification and solely to the extent necessary to comply with the Specification.

THE SPECIFICATION IS PROVIDED "AS IS," AND THE CO-DEVELOPERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE CO-DEVELOPERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE SPECIFICATIONS.

The name and trademarks of the Co-Developers may NOT be used in any manner, including advertising or publicity pertaining to the Specifications or their contents without specific, written prior permission. Title to copyright in the Specifications will at all times remain with the Co-Developers.

No other rights are granted by implication, estoppel or otherwise.

Abstract

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

Status

The first edition of this specification was published in October 2004, which was a pre-release version for public comment. This February 2005 edition constitutes the first standardized version of the messaging protocol.

Table of Contents

1.0 Introduction	5
1.1 Requirements	6
1.2 Notations and Terminology	6
1.3 Notational Conventions	6
1.4 Conformance	7
1.5 XML Namespaces	7
1.6 Terminology	8
2.0 Addressing	8
2.1 Endpoint References	8
2.2 Other WS-Addressing Headers	12
2.3 mustUnderstand Usage	12
2.4 To	13
2.5 ReplyTo	15
2.6 FaultTo	17
2.7 MessageID and RelatesTo	18
2.8 System	19
2.9 Selectors	20
2.10 Action	23
2.11 wsa:From	26
3.0 WS-Management Control Headers	26
3.1 Operation Timeout	27
3.2 Maximum Envelope Size	28
3.3 Locale	28
3.4 Options	29

4.0	Resource Access	31	
4.1	Introduction		31
4.2	WS-Transfer		32
4.3	Addressing Uniformity		33
4.4	WS-Transfer: Get		34
4.5	WS-Transfer: Delete		34
4.6	WS-Transfer: Create		35
4.7	WS-Transfer: Put		37
4.8	WS-Management: Rename		39
4.9	Fragment-Level WS-Transfer		43
4.10	Fragment-Level WS-Transfer: Get		45
4.11	Fragment-Level WS-Transfer: Put		47
4.12	Fragment-Level WS-Transfer: Create		49
4.13	Fragment-Level WS-Transfer: Delete		50
5.0	WS-Enumeration	51	
5.1	Introduction		51
5.2	WS-Enumeration: Enumerate		52
5.3	Filter Interpretation		53
5.4	WS-Enumeration: Pull		53
5.5	WS-Enumeration: Release		56
5.6	Ad-Hoc Queries and Fragment-Level Enumerations		56
6.0	Custom Actions (Methods)	57	
6.1	General		57
7.0	Eventing	59	
7.1	General		59
7.2	Subscribe		59
7.2.1	General		59
7.2.2	Filtering		60
7.2.3	Connection Retries		61
7.2.4	wse: SubscribeResponse		61
7.2.5	Heartbeats		62
7.2.6	Bookmarks		63
7.2.7	Delivery Modes		66
7.2.8	Event Action URI		67
7.2.9	Delivery Sequencing and Acknowledgement		67
7.2.10	Push Mode		68
7.2.11	PushWithAck Mode		69
7.2.12	Batched Delivery Mode		69
7.2.13	Pull Delivery Mode		73
7.3	GetStatus		75
7.4	Unsubscribe		75
7.5	Renew		75
7.6	SubscriptionEnd		76
7.7	Acknowledgement of Delivery		77
7.8	Refusal of Delivery		78
7.9	Dropped Events		79
8.0	Standard System Resources	81	
8.1	wsman:system/2005/02/this		81
8.2	Other Discovery		82

9.0	Security	82
9.1	Introduction	82
9.2	Security Profiles	82
9.3	Interoperation Conformance	83
9.4	wsman:secprofile/http/standard	83
9.5	wsman:secprofile/https/standard	84
9.6	wsman:secprofile/https/mutual	85
9.7	wsman:secprofile/https/dual	85
9.8	wsman:secprofile/https/selected	86
9.9	wsman:secprofile/https/mutual/rechallenge	87
9.10	Subscriptions	88
9.11	Standard profiles	89
9.12	wsman:secprofile/https/standard/userRef	89
9.13	wsman:secprofile/https/mutual/certhash	90
9.14	Correlation of Events with Subscription	91
9.15	Transport-Level Authentication Failure	91
10.0	Transports and Message Encoding	91
10.1	Introduction	91
10.2	HTTP(S) Encoding	92
10.3	SOAP	93
10.4	Lack of Response	94
10.5	Replay Of Messages	94
10.6	Encoding Limits	95
10.7	Binary Attachments	95
11.0	Faults	96
11.1	Introduction	96
11.2	Fault Encoding	96
11.3	NotUnderstood Faults	97
11.4	Degenerate Faults	98
11.5	Master Fault Table	99
11.5.1	wsman:AccessDenied	99
11.5.2	wsman:NoAck	99
11.5.3	wsa:ActionNotSupported	100
11.5.4	wsman:Concurrency	100
11.5.5	wsman:AlreadyExists	101
11.5.6	wsen:CannotProcessFilter	101
11.5.7	wse:DeliveryModeRequestedUnavailable	102
11.5.8	wsman:DeliveryRefused	103
11.5.9	wsa:DestinationUnreachable	103
11.5.10	wsman:EncodingLimit	104
11.5.11	wsa:EndpointUnavailable	106
11.5.12	wse:EventSourceUnableToProcess	106
11.5.13	wsen:FilterDialectRequestedUnavailable	107
11.5.14	wse:FilteringNotSupported	108
11.5.15	wsen:FilteringNotSupported	108
11.5.16	wse:FilteringRequestedUnavailable	109
11.5.17	wsman:IncompatibleEPR	110
11.5.18	wsman:InternalError	110
11.5.19	wsman:InvalidBookmark	111

11.5.20	wsen: InvalidEnumerationContext	112
11.5.21	wse: InvalidExpirationTime	112
11.5.22	wsen: InvalidExpirationTime	113
11.5.23	wse: InvalidMessage	113
11.5.24	wsa: InvalidMessageInformationHeader	114
11.5.25	wsman: InvalidOptions	115
11.5.26	wsman: InvalidParameter	115
11.5.27	wxf: InvalidRepresentation	116
11.5.28	wsman: InvalidSelectors	117
11.5.29	wsman: InvalidSystem	118
11.5.30	wsa: MessageInformationHeaderRequired	118
11.5.31	wsman: MetadataRedirect	119
11.5.32	wsman: QuotaLimit	119
11.5.33	wsman: RenameFailure	120
11.5.34	wsman: SchemaValidationError	120
11.5.35	wsen: TimedOut	121
11.5.36	wsman: TimedOut	121
11.5.37	wse: UnableToRenew	122
11.5.38	wse: UnsupportedExpirationType	123
11.5.39	wsen: UnsupportedExpirationType	123
11.5.40	wsman: UnsupportedFeature	123
12.0	WS-Management XSD	125
13.0	Acknowledgements	129
14.0	References	129

1.0 Introduction

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

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

- DISCOVER the presence of management resources and navigate between them.
- GET, PUT, CREATE, RENAME, and DELETE individual management resources, 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 may also choose not to support one or more

areas of functionality listed above if that functionality is not appropriate to the target device or system.

1.1 Requirements

This specification intends to meet the following requirements:

- Constrain Web services protocols and formats so Web services can be implemented in management services with a small footprint, in both hardware and software.
- Define minimum requirements for compliance without constraining richer implementations.
- Ensure composability with other Web services specifications
- Minimize additional mechanism beyond the current Web service architecture.

1.2 Notations and Terminology

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

1.3 Notational Conventions

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

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

- The syntax appears as an XML instance, but values in italics indicate data types instead of values.
- Characters are appended to elements and attributes to indicate cardinality:
 - "?" (0 or 1)
 - "*" (0 or more)
 - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.
- The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- An ellipsis (i.e. "...") indicates a point of extensibility that allows other child or attribute content. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. If a receiver does not recognize an extension, the receiver SHOULD NOT process the message and MAY fault.
- XML namespace prefixes (see Table 1) are used to indicate the namespace of the element being defined.

Throughout the document, whitespace is used for readability. However, in practice, whitespace should not be used in the XML values unless it is part of the value (Conformance rule R10.3-9 applies to all messaging).

1.4 Conformance

An implementation is not conformant with this specification if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined in the conformance rules for each section, as indicated by the following format:

Rnnnn: *Rule text*

General conformance rules:

R1.4-1: To be conformant, the service MUST comply with all the rules defined in this specification. Items marked with MUST are required, and items marked with SHOULD are highly advisable to maximize interoperability. Items marked with MAY indicate the preferred implementation in terms of expected features, but interoperability should not be affected if they are ignored.

R1.4-2: A SOAP Node MUST NOT use the XML namespace identifier for this specification (listed in **Section 1.5**) within SOAP envelopes unless it is conformant with this specification.

1.5 XML Namespaces

R1.5-1: The XML namespace URI that MUST be used by conformant services of this specification is:

(1) <http://schemas.xmlsoap.org/ws/2005/02/management>

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

Table 1: Prefixes and XML namespaces used in this specification.

Prefix	XML Namespace	Specification(s)
wsman	http://schemas.xmlsoap.org/ws/2005/02/management	This specification
s	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2 [SOAP 1.2]
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1 , 2]
wSDL	http://schemas.xmlsoap.org/wSDL	WSDL/1.1 [WSDL 1.1]
wsa	http://schemas.xmlsoap.org/ws/2004/08/addressing	WS-Addressing [WS-Addressing]
wse	http://schemas.xmlsoap.org/ws/2004/08/eventing	WS-Eventing [WS-Eventing]
wsen	http://schemas.xmlsoap.org/ws/2004/09/enumeration	WS-Enumeration [WS-Enumeration]

wxf	http://schemas.xmlsoap.org/ws/2004/09/transfer	WS-Transfer [WS-Transfer]
-----	--	---

1.6 Terminology

Client

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

Service

An application that provides management services to clients by exposing the web services defined in this document. A service typically is equivalent to the network "listener" and is associated with a physical transport address and is essentially a type of manageability access point.

Resource

An endpoint which represents a distinct type of management operation or value. A service exposes one or more resources and some resources can have more than one instance. In this sense, a resource is similar to a "class" or a database table, and an instance is similar to an instance of the class or a row in the table.

System

A top-level partitioning element identifies a named group of resources. Just as resources are similar to "tables" or "classes", a system is similar to a database or a namespace. A service may expose one or more systems. If no system is explicitly exposed, a default system is implied.

Selector

A resource-relative identifier and value which is an instance-level discriminant. This is essentially a filter which isolates the instance.

The relationship of services to systems to resources is as follows:

- A service consists of one or more systems. If there is only a single system, it is implied. Otherwise, it must be named.
- A system consists of one or more resources.
- A resource may contain one or more instances.
- If more than one instance for a resource exists, they are isolated or identified through the Selector mechanism.

2.0 Addressing

2.1 Endpoint References

WS-Management uses WS-Addressing *endpoint references* (also known as *EPRs*) as the addressing model for individual resources. To access a resource, the endpoint reference is used in the SOAP header, with the modifications and restrictions discussed in the following sections. This management endpoint reference uses a fixed representation, which is a tuple of the following SOAP headers:

- (1) **wsa:To** (required) : The transport address of the service suffixed by the URI of the targeted management resource.
- (2) **wsman:System** (optional) : The targeted system within the service
- (3) **wsman:SelectorSet** (optional) : Used to reference or select the specific instance of a resource, if there is more than one instance

The WS-Management endpoint reference is defined in SOAP as follows:

```

(1)   <wsa:EndpointReference>
(2)   <wsa:Address>
(3)     Network address URL ? ResourceURI=( Management Resource URI )
(4)   </wsa:Address>
(5)   <wsa:ReferenceParameters>
(6)     <wsman:System Name="SystemName">
(7)       ...optionally nested systems
(8)     </wsman:System> *
(9)     <wsman:SelectorSet>
(10)      <wsman:Selector Name="Selector=name"> +
(11)        Selector-value
(12)      </wsman:Selector>
(13)    </wsman:SelectorSet> ?
(14)  </wsa:ReferenceParameters>
(15) </wsa:EndpointReference>
(16) ...

```

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

wsa:Address

This is a compound URI, consisting of the network address URL suffixed by the ? character, suffixed by the token sequence **ResourceURI =** and the actual URI of the management resource enclosed in parentheses (line 3).

wsa:ReferenceParameters/wsman:System

Optionally, the targeted system name as described in 2.8. Typically a single value, but may be nested. This is used to route the message after it has left the SOAP channel.

wsa:ReferenceParameters/wsman:SelectorSet

The optional set of Selectors as described in 2.9. These are used to select an instance if the ResourceURI represents a multi-instanced target

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

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

As example, the following address definition

```

<wsa:EndpointReference>
  <wsa:Address> Address+URI </wsa:Address>
  <wsa:ReferenceProperties>

```

```

    <other:UserProp>prop-value</other:UserProp>
  </wsa:ReferenceProperties>
  <wsa:ReferenceParameters>
    <wsman:System>sysURI</wsman:System>
    <wsman:SelectorSet>
      <wsman:Selector Name="Selector=name">
        Selector-value
      </wsman:Selector>
    </wsman:SelectorSet>
    <other:UserParam> param </otherUserParam>
  </wsa:ReferenceParameters>
</wsa:EndpointReference>

```

...becomes the following when actually used in a SOAP message, in which `wsa:Address` becomes `wsa:To`, and the reference properties and reference parameters are unwrapped and juxtaposed:

```

<s:Envelope ...>
  <s:Header>
    <wsa:To> Address+URI </wsa:To>
    <other:UserProp>prop-value</other:UserProp>
    <wsman:System>sysURI</wsman:System>
    <wsman:SelectorSet>
      <wsman:Selector Name="Selector=name">
        Selector-value
      </wsman:Selector>
    </wsman:SelectorSet>
    <other:UserParam> param </otherUserParam>"
    ...

```

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

Note that as of this writing the WS-Addressing specification is being refined, and the reference property mechanism may be removed. WS-Management makes no use of reference properties per se (although it uses reference parameters), but future implementations should examine the status of WS-Addressing before building richer implementations which make use of reference properties.

Note that the `wsa:To`, `wsman:System`, and `wsman:SelectorSet` work together to *reference* the resource to be managed, but the actual *method* or *operation* to be executed against this resource is indicated by the `wsa:Action` header.

Here is an example of WS-Management headers in an actual message:

```

(17)  <s:Envelope>
(18)    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(19)    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(20)    xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(21)  <s:Header>

```

```

(22)     ...
(23)     <wsa:To> http://123.99.222.36/wsman?ResourceURI=
(24)         (http://acme.org/hardware/2005/02/storage/physDisk)
(25)     </wsa:To>
(26)     <wsman:System> wsman:acme.org/2005/02/systems/main </wsman:System>
(27)     <wsman:SelectorSet>
(28)         <wsman:Selector Name="LUN"> 2 </wsman:Selector>
(29)     </wsman:SelectorSet>
(30)     <wsa:Action> http://schemas.xmlsoap.org/ws/2004/09/transfer/Get </wsa:Action>
(31)     <wsa:MessageID> uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91 </wsa:MessageID>
(32)     ...
(33) </s:Header>
(34) <s:Body> ...

```

Definitions:

wsa:To

The network (or transport-level) address of the service suffixed by the ResourceURI of the management resource to be accessed.

wsman:System

The optional system within the service. The system is used to name a group of resources. If no System is specified, then a default system is implied.

wsman:SelectorSet

A wrapper for the Selectors.

wsman:SelectorSet/Selector

Identifies or 'selects' the resource instance to be accessed, if more than one instance of the resource exists. In this case, the Selector is "LUN" (logical unit number) and the selected device is unit number "2".

wsa:Action

Identifies which operation is to be carried out against the resource, in this case a "Get".

wsa:MessageID

Identifies this specific message uniquely for tracking and correlation purposes.

R2.1-1: All messages MUST contain an endpoint reference. This applies especially to continuation messages such as wsen:Pull or wsen:Release, which continue an operation begun in a previous message. Even though there is contextual information in such messages binding it to a previous operation, the WS-Addressing endpoint reference is still required in the message to help route it to the correct handler.

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

2.2 Other WS-Addressing Headers

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

R2.2-1: A conformant service **MUST** recognize and process the following WS-Addressing header blocks. Any others are optional as specified in WS-Addressing and **MAY** be present, but a conformant service **MAY** reject any additional headers and fail to process the message, issuing a soap:NotUnderstood fault.

- **wsa:ReplyTo** (required)
- **wsa:FaultTo** (optional)
- **wsa:MessageID** (required)
- **wsa:Action** (required)
- **wsa:RelatesTo** (required in responses)

The usage of these is discussed in subsequent sections.

2.3 mustUnderstand Usage

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

Obviously, if the service cannot understand the primary endpoint reference of the resource (the *ResourceURI*) it will not be able to service the request in any case. Similarly, if the *wsa:Action* is not understood, the implementation will not know how to process the message. So, for the following elements, the omission or inclusion of *mustUnderstand* has no real effect on the message in practice, as *mustUnderstand* is implied:

- *wsa:To*
- *wsman:SelectorSet*
- *wsman:System*
- *wsa:MessageID*
- *wsa:RelatesTo*
- *wsa:Action*

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

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

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

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

2.4 To

In requests the `wsa:To` address contains both the network address and the management resource URI as a suffix. In responses and event deliveries, only the network address portion needs to appear.

The syntax is:

```
(35) <wsa:To> NetworkTransport [ ? ResourceURI=( resourceURI ) ] </wsa:To>
```

R2.4-1: The `wsa:To` header **MUST** be present in all messages and conform the the syntax above.

R2.4-2: When the WS-Management ResourceURI is required, the following tokens must be suffixed to the network address URL:

?ResourceURI=(resourceURIliteral)

Transports other than HTTP(S) may require that the `?` be omitted or some other token used to introduce the ResourceURI token. This decision is relegated to bindings specifications for those transports. The URI literal **MAY** contain parentheses as long as they are not unbalanced, since they are used as delimiters in the above syntax.

R2.4-3: In messages with the following `wsa:Action` URIs, the ResourceURI **MUST** be suffixed to the network address:

Action URI

`http://schemas.xmlsoap.org/ws/2004/09/transfer/Get`

`http://schemas.xmlsoap.org/ws/2004/09/transfer/Put`

`http://schemas.xmlsoap.org/ws/2004/09/transfer/Create`

`http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete`

`http://schemas.xmlsoap.org/ws/2005/02/management/Rename`

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

`http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull`

`http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew`

`http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus`

`http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release`

<http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe>

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

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

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

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

R2.4-4: For the request message of custom actions, the ResourceURI **MUST** be affixed to the network address to help route the message to the correct handler. (See also R10.2-8)

R2.4-5: The ResourceURI **SHOULD NOT** appear suffixed to the the network address in other messages, such as responses or events.

Note that `wsa:To` must be present in *all* messages, including replies and faults, even though it appears redundant for many transports. In practice, the ResourceURI is only required as a suffix to the the network address URI in requests to reference the target resource. Responses are not accessing the WS-Management space, so the ResourceURI is not suffixed to the address.

The ResourceURI itself indicates the targeted management value, management action (method), or event source within the scope of the service and system.

R2.4-6: If the ResourceURI is missing, the service **MUST** issue a `wsa:DestinationUnreachable` fault with a detail code of `wsman:faultDetail/InvalidResourceURI`.

R2.4-7: The ResourceURI **MUST** only be used to indicate the identity of a resource, but **MAY NOT** be used to indicate the action being applied to that resource, which is properly expressed using the `wsa:Action` URI. This is any valid RFC 2396-conformant URI. The scheme may be `wsman:` if no other scheme is applicable; this scheme implies that the item may only be retrieved using WS-Management itself.

The format and syntax of the ResourceURI is any valid URI according to RFC 2396. While there is no default scheme, `wsman:` or `http:` are common defaults. If `http:` is used, users may expect to find web-based documentation of the resource at that address.

R2.4-8: The ResourceURI **MUST** be unique and unambiguous within the scope of a service if the service supports only a default System, and **MUST** be unique within the scope of a System if multiple systems are supported.

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

R2.4-9: If a valid request is received but the resource is not available at that time, a service SHOULD issue either a `wsa:DestinationUnreachable` fault, but MAY issue a `wsa:EndpointUnavailable` fault if it can be determined that the resource is actually offline as opposed to being incorrectly formatted or non-existent.

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

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

While the URI may be used to define an instance of a multi-instance resource and the Selector mechanism in 2.9 ignored, it is recommended that the URI be limited to identifying the resource type, and that Selectors be used to reference the instance. If the resource only consists of a single instance, then the URI refers to the singleton instance.

The following faults apply:

R2.4-10: The service SHOULD issue faults in the following situations relating to the resource:

- a) If the resource is offline, a `wsa:EndpointUnavailable` fault is returned with a detail code of `wsman:faultDetail/ResourceOffline`
- b) If the resource cannot be located ("not found"), or the `ResourceURI` is of the incorrect form or missing completely, a `wsa:DestinationUnreachable` fault is returned with a detail code of `wsman:faultDetail/InvalidResourceURI`.
- c) If the resource is valid, but internal errors occur, a `wsman:Internal` fault is returned.
- d) If the resource cannot be accessed for security reasons, a `wsman:AccessDenied` fault is returned.

2.5 ReplyTo

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

R2.5-1: A `wsa:ReplyTo` header MUST be present in all request messages when a reply is required (some messages do not require replies and may omit this). This MUST be either be a valid address for a new connection using any transport supported by the service, or the URI **`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`** [see WS-Addressing] which indicates that the reply is to be delivered over the same connection that the request arrived on. If the `wsa:ReplyTo` is missing, a `wsa:MessageInformationHeaderRequired` fault is returned.

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

R2.5-2: A conformant service MAY require that all responses be delivered over the same connection on which the request arrives. In this case, the URI

discussed in R2.5-1 MUST be used to indicate this. Otherwise, the service MUST return a wsman:UnsupportedFeature fault with a detail code of wsman:faultDetail/AddressingMode.

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

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

This is used in cases where the client included suffixes on the HTTP or HTTPS address which are not understood by the service. The service must return these suffixes nonetheless.

R2.5-5: Any reference properties and reference parameters supplied in the wsa:ReplyTo address MUST be included in the actual response message as top-level headers as specified in WS-Addressing.

WS-Addressing allows clients to include client-defined reference properties and reference parameters in *ReplyTo* headers. The WS-Addressing specification requires that these be extracted from requests and placed in the responses by removing the *ReferenceParameters* and *ReferenceProperties* wrappers, placing all of the values as top-level SOAP headers in the response as discussed in section 2.1. This allows clients to better correlate responses with the original requests. This step cannot be omitted. In the example below, the "user-defined content" must be included in the reply message:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5)   <s:Header>
(6)     ...
(7)     <wsa:To> http://1.2.3.4/wsman </wsa:To>
(8)     <wsa:ReplyTo>
(9)       <wsa:Address>
(10)        http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(11)      </wsa:Address>
(12)      <wsa:ReferenceParameters>
(13)        user-defined content
(14)      </wsa:ReferenceParameters>
(15)    </wsa:ReplyTo>
(16)    ...
```

R2.5-6: If the wsa:ReplyTo address is not usable or is missing, the service should not reply to the request, as there is no way to properly reply and it should close or terminate the connection according to the rules of the current network transport. In these cases, the service SHOULD locally log some type of entry to help locate the client defect later.

2.6 FaultTo

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

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

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

R2.6-3: A conformant service MAY require that all faults be delivered to the client over the same transport or connection on which the request arrives. In this case, the URI MUST be **`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`** [see `WS-Addressing`]. If services do not support separately-addressed fault delivery and the `wsa:FaultTo` is any other address, a `wsman:UnsupportedFeature` fault MUST be returned with a detail code of `wsman:faultDetail/AddressingMode`.

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

R2.6-4: Any reference properties and reference parameters supplied in the `wsa:FaultTo` address MUST be included in the actual response message as top-level headers as specified in `WS-Addressing`.

`WS-Addressing` allows clients to include client-defined reference properties and reference parameters in `wsa:FaultTo` headers. The `WS-Addressing` specification requires that these be extracted from requests and placed in the faults by removing the *ReferenceParameters* and *ReferenceProperties* wrappers, placing all of the values as top-level SOAP headers in the fault. This allows clients to better correlate faults with the original requests. This step cannot be omitted. In the following example, the "user-defined content" MUST appear in the fault, if it occurs:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5) <s:Header>
(6)   ...
(7) <wsa:To> http://1.2.3.4/wsman </wsa:To>
(8) <wsa:FaultTo>
(9)   <wsa:Address>
```

```

(10)      http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(11)      </wsa:Address>
(12)      <wsa:ReferenceParameters>
(13)          user-defined content
(14)      </wsa:ReferenceParameters>
(15)      </wsa:FaultTo>
(16)      ...

```

R2.6-5: If the `wsa:FaultTo` address is not usable, the service should not reply to the request, as there is no way to properly return the fault. Similarly, if no `FaultTo` address is supplied, and the service does not have sufficient information to properly fault the response, it should not reply and should close the network connection. In these cases, the service **SHOULD** locally log some type of entry to help locate the client defect later.

R2.6-6: The service **MUST** properly duplicate the `wsa:Address` of the `wsa:FaultTo` element in the `wsa:To` of the reply, even if some of the information is not understood by the service.

This is used in cases where the client included suffixes on the HTTP or HTTPS address which are not understood by the service. The service must return these suffixes nonetheless.

2.7 MessageID and RelatesTo

R2.7-1: The `MessageID` and `RelatesTo` URIs **MUST** be of the form

uuid:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx

...in which each **x** is an upper- or lower-case hexadecimal digit; there are no spaces or other tokens. It is **NOT REQUIRED** that this formally be a DCE-style GUID with provable uniqueness properties, although this is typical. It is critical to note that case is **significant**, since this value is treated as a URI within the WS-Addressing space. A test for identity must be done with a *case-sensitive* string or memory comparison.

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

While services and clients should not issue different `MessageIDs` which differ only by case, the service is not required to detect this. These values are treated as true URI values.

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

R2.7-4: If the `MessageID` is not of the specified format or is not parsable, a `wsa:InvalidMessageInformationHeader` fault should be returned.

Example:

```
(1) <wsa:MessageID>
(2)   uuid:d9726315-bc91-430b-9ed8-ce5ffb858a91
(3) </wsa:MessageID>
```

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

2.8 System

Services may act as logical dispatching services over multiple underlying logical systems. A System is nothing more than a named partitioning or grouping of related resources, in which the system name is indicated by a URI. Alternately, the System value may be seen as a routing ticket for use once the message leaves the SOAP channel.

If the service supports multiple systems, the *ResourceURI* alone will not contain sufficient routing information to properly dispatch the request and the System header must be present.

R2.8-1: A service MAY support an optional *wsman:System* header to route the request to the right system. This header forms part of the endpoint reference and is a reference parameter. The value is any legal URI according to RFC 2396.

R2.8-2: If a service does not actually support multiple systems, the `wsa:System` header MUST be omitted. When multiple systems are supported, omission of the *wsman:System* header implies access to the default system for that service. The meaning and scope of the default system is implementation-specific.

This mechanism is primarily intended for services which act as a front-end to multiple devices, resource databases, or proxies to real machines. The *System* name is used to reference which machine or device to which to route the request once the message is moved out of band to the SOAP stack. While this same concept may be modeled using a more complex *To* address, it is distinct in that the *System* header applies to a logical address outside of the SOAP space (within the information system being exposed by the service) and is a logical extension to the *ResourceURI* itself, whereas the *To* address implies an address reachable by the SOAP message.

Using the relational database model as an analogy, the System URI may be seen as a 'database' identifier, whereas the *ResourceURI* models the concept of a table within the database, and the *Selectors* element identifies which "row" in the table to access.

Format:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5) <s:Header>
(6)   ...
(7) <wsa:To> Service transport address and Resource URI </wsa:To>
```

```
(8) <wsman:System Name="URI representing the destination system"/>
(9) ...
```

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

wsman:System/@Name

A named scope within the domain of the service implied by the *To* address. The Name may contain any valid URI according to RFC 2396. The URI is service- and system-specific. Only one wsman:System element may be present at the top level, but it MAY be nested:

```
(10) <wsman:System Name="SystemA">
(11)   <wsman:System Name="SystemB">
(12)     <wsman:System Name="SystemC"/>
(13)   </wsman:System>
(14) </wsman:System>
```

The following faults apply:

R2.8-3: The service SHOULD issue faults in the following situations relating to the System:

- a) If the System is offline, a wsa:EndpointUnavailable fault is returned with a detail code of wsman:faultDetail/SystemOffline
- b) If the System cannot be located ("not found"), a wsa:DestinationUnreachable fault is returned.
- c) If the System is valid, but internal errors occur, a wsman:Internal fault is returned.
- d) If the System cannot be accessed for security reasons, a wsman:AccessDenied fault is returned.
- e) If the URI format is not correct for the service, a wsman:InvalidSystem fault is returned with a detail code of wsman:faultDetail/InvalidSystem.

2.9 Selectors

Selectors are optional elements to reference a specific Resource instance in the set of all instances implied by the *ResourceURI*.

In practice, because the *ResourceURI* often acts as a table or a 'class', the SelectorSet element is a discriminant used to reference a specific 'row' or 'instance'. If there is only one instance of a resource implied by the *ResourceURI*, the SelectorSet may be omitted. If more than one Selector value is required, the entire set of Selectors is interpreted by the service in order to reference the targeted object. The service is free to examine as many Selectors as required and ignore those which are redundant once the instance has been identified.

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

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

R2.9-1: If a resource has more than one instance, a `wsman:SelectorSet` element MAY be used to distinguish which instance is targeted if the common endpoint reference model is in use. Selector names should be treated in a case-insensitive manner. Values MAY be treated case-insensitively or case-sensitively by the service.

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

R2.9-2: All content within the `SelectorSet` element is to be treated as a single reference parameter with a scope relative to the `ResourceURI`.

R2.9-3: The service SHOULD examine whatever Selectors are required to reference the target, and MAY ignore any additional Selectors once the target has been identified. In this sense, the set of Selectors is logically ANDed and the service has the option to ignore any additional redundant Selectors. If the set of Selectors is insufficient to reference a resource or contain invalid names or values for the targeted resource, then a `wsman:InvalidSelectors` fault should be returned to the client with the following detail codes:

- a) `wsman:faultDetail/InsufficientSelectors` if Selectors are missing
- b) `wsman:faultDetail/TypeMismatch` if Selector values are the wrong types
- c) `wsman:faultDetail/InvalidValue` if the Selector value is of the correct type from the standpoint of XML types, but out of range or otherwise illegal in the specific information domain.
- d) `wsman:faultDetail/AmbiguousSelectors` if the Selectors cannot isolate a single instance.
- e) `wsman:faultDetail/UnexpectedSelectors` if the Name is not a recognized Selector name.

R2.9-4: The Selector Name attribute MUST NOT be duplicated at the same level of nesting. The service SHOULD return a `wsman:InvalidSelector` fault with a detail code of `wsman:faultDetail/DuplicateSelectors` if this occurs.

This specification does not mandate the use of Selectors. Some implementations may decide to use complex URI schemes in which the `ResourceURI` itself implicitly identifies the instance. However, most information domains will benefit from the separation of type and instance identities into separate addressing elements.

Format:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5) <s:Header>
(6)   ...
(7) <wsa:To> service transport address + ResourceURI </wsa:To>
```

```

(8)    <wsman:SelectorSet> ?
(9)    <wsman:Selector Name="name"> value </wsman:Selector> +
(10)   </wsman:Selector>
(11)   ...
(12)   </s:Header>
(13)   ...

```

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

wsman:To

Network address and ResourceURI suffix

wsman:SelectorSet

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

wsman:SelectorSet/Selector

Used to describe the Selector and its value. If more than one Selector is required, then there is one Selector element for each part of the overall Selector. The value of this element is the Selector value.

wsman:SelectorSet/Selector/@Name

The name of the Selector (to be treated in a case-insensitive manner).

The value of a Selector may be a nested endpoint reference. In the example below, the Selector on line 22 is a part of a SelectorSet consisting of nested EPR (lines 23-30) with its own Address+ResourceURI and SelectorSet elements:

```

(14)   <s:Envelope
(15)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(16)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(17)     xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(18)   <s:Header>
(19)     ...
(20)   <wsman:SelectorSet>
(21)     <wsman:Selector Name="Primary"> 123 </wsman:Selector>
(22)     <wsman:Selector Name="EPR">
(23)       <wsa:EndpointReference>
(24)         <wsa:Address> address + ResourceURI </wsa:Address>
(25)         <wsa:ReferenceParameters>
(26)           <wsman:SelectorSet>
(27)             <wsman:Selector Name="name"> value </wsman:Selector>
(28)           </wsman:SelectorSet>
(29)         </wsa:ReferenceParameters>
(30)       </wsa:EndpointReference>
(31)     </wsman:Selector>
(32)   </wsman:SelectorSet>
(33)   ...
(34)

```

R2.9-5: The value of a wsman:Selector MUST be one of

(a) A simple type as defined in the XML schema namespace

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

(b) A nested `wsa:EndpointReference` using the WS-Management addressing model.

A service MAY fault Selector usage with `wsman:InvalidSelector` if the Selector is not a simple type or of a supported schema.

R2.9-6: A conformant service MAY reject any Selector or nested Selector with a nested endpoint reference whose `wsa:Address` value is not the same as the primary `wsa:To` value or is not <http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous>.

The primary purpose for this nesting mechanism is to allow resources which can answer questions about other resources. If there were a ResourceURI which could provide historical statistics, and we needed to find a statistic about how many times a resource was accessed, it would need the EPR of that resource, plus an indication of which statistic was needed:

```
(35) <wsa:To> http://1.2.3.4/wsman?ResourceURI=(  
(36)     http://acme.com/system/2005/02/AccessStatistics)  
(37) </wsa:To>  
(38) <wsman:SelectorSet>  
(39)   <wsman:Selector Name="TargetedEPR">  
(40)     ...EPR of item we are querying about...  
(41)   </wsman:Selector>  
(42)   <wsman:Selector Name="TotalAccessCountInRecentDays"> 3 </wsman:Selector>  
(43) </wsman:SelectorSet>
```

...with which we query the system to find out how many times the specified resource was accessed in the last 3 days.

2.10 Action

The WS-Addressing:Action URI is typically used to indicate the "method" being invoked against the resource. So, the ResourceURI indicates *what* is being accessed, and the Action indicates *which method or operation* is being applied.

R2.10-1: The `wsa:Action` URI MUST NOT be used to indicate the identity of the specific management endpoint, but only the operation (method) to use against that endpoint.

R2.10-2: For all resource endpoints, a service MUST return a `wsa:ActionNotSupported` fault (defined in WS-Addressing) if a requested action is not supported by the service for the specified resource.

In other words, to model the "Get" of item "Disk", the ResourceURI defines the reference to "Disk" (using Selectors to indicate which disk), but the `wsa:Action` URI is what will contain the "Get". Implementations are free to additionally support custom methods which combine the notion of "Get" and "Disk" into a single "GetDisk" action, as long as they strive to support the separated form to maximize interoperability. One of the main points behind WS-Management is to unify common methods wherever possible.

R2.10-3: If a service exposes any of the following types of capabilities, a conformant service **MUST** at least expose that capability using the definitions in the following table according to the rules of this specification. The service **MAY** **OPTIONALLY** expose additional similar functionality using a distinct `wsa:Action` URI.

Action URI	Description
http://schemas.xmlsoap.org/ws/2004/09/transfer/Get	Models any simple single item retrieval
http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse	Response to the above "Get"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Put	Models an update of an entire item
http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse	Response to "Put"
http://schemas.xmlsoap.org/ws/2004/09/transfer/Create	Models creation of a new item
http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse	The 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	The response to the delete
http://schemas.xmlsoap.org/ws/2005/02/management/Rename	Renames the item
http://schemas.xmlsoap.org/ws/2005/02/management/RenameResponse	Response to above message
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate	Begins an enumeration or query
http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse	Response to above enumeration
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull	Retrieve the next batch of results from enumeration
http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse	Response to the above "Pull"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew	Renews an enumerator which may have timed out [not required in WS-Management]
http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse	Response to the "Renew" [not required in WS-Management]
http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus	Gets the status of the enumerator [not required in WS-

	Management]
http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse	Response to the "GetStatus" request [not required in WS-Management]
http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release	Releases an active enumerator
http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse	Response to the above "Release"
http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd	Notification than an enumerator has terminated [not required in WS-Management]
http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe	Models a subscription to an event source
http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse	Response to the above "Subscribe"
http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew	Renews a subscription prior to its expiration
http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse	Response to the renew request
http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus	Requests the status of a subscription
http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse	The response to the "GetStatus" message
http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe	Used to remove an active subscription
http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse	The response to the Unsubscribe operation
http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd	A message delivered to indicate that a subscription has terminated.
http://schemas.xmlsoap.org/ws/2005/02/management/Events	A batched delivery of events based on a subscription
http://schemas.xmlsoap.org/ws/2005/02/management/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.xmlsoap.org/ws/2005/02/management/DroppedEvents	A pseudo-event which indicates that the real event was dropped.
http://schemas.xmlsoap.org/ws/2005/02/management/Ack	Used by event subscribers to acknowledge receipt of events. Allows event streams to be strictly sequenced.
http://schemas.xmlsoap.org/ws/2005/02/management/Event	Used for a singleton event which does not define its own action.

R2.10-4: A custom action MAY BE supported if the operation is a custom method whose semantic meaning is not present in the table if the item is an event.

R2.10-5: All *event* deliveries MUST contain a unique action URI which identifies the type of the event delivery. For singleton deliveries where there is only one event per message (the delivery mode <http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push>), the `wsa:Action` URI defines the event type. For other delivery modes, the Action varies, as described in section 6 of this specification.

2.11 wsa:From

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

R2.11-1: A conformant service MAY include a `wsa:From` address in message, but is NOT REQUIRED to do so. A conformant service SHOULD process any incoming message which has a `wsa:From` element.

R2.11-2: A conformant service SHOULD NOT fault any message with a `wsa:From` element, whether or not `mustUnderstand` is included

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

3.0 WS-Management Control Headers

3.1 Operation Timeout

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

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

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

- R3.1-1:** All request messages MAY contain a wsman:OperationTimeout header element that indicates the maximum amount of time the client is willing to wait for the service to issue a response. The service SHOULD interpret the timeout countdown as beginning from the point the message is processed until a response can be generated.
- R3.1-2:** The Service SHOULD issue a wsman:TimedOut fault if this time is exceeded and the operation is not yet complete. If the OperationTimeout value is not valid, then a wsman:InvalidHeader fault should be generated with a detail code of wsman:faultDetail/InvalidTimeout.
- R3.1-3:** If the services does not support user-defined timeouts, a wsman:fault/UnsupportedFeature fault SHOULD be returned with a detail code of wsman:faultDetail/OperationTimeout.
- R3.1-4:** If the wsman:OperationTimeout is omitted, the service MAY interpret this as an instruction to block indefinitely until a response is available, or MAY impose a default timeout.

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

If the timeout occurs in such a manner that the service has already performed some of the work associated with the request, an anomalous condition is reached in terms of service state. This specification does not attempt to address behavior in this situation. Clearly, services should undo the affects of any partially complete operations if possible, but this is not always practical. It is recommended that the service keep a local log of requests and operations which can be queried in such cases later by the client.

- R3.1-5:** If mustUnderstand is applied to the wsman:OperationTimeout, then the service MUST comply with it or return the fault specified in R3.1-2. If mustUnderstand is omitted, then the value is treated as a hint. The service MAY timeout earlier than the time requested by the client and is under no obligation to wait until the timeout interval has expired before returning a fault.

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

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

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

3.2 Maximum Envelope Size

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

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

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

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

- R3.2-1:** All request messages MAY contain a `wsman:MaxEnvelopeSize` header element that indicates the maximum number of octets (not characters) in the entire SOAP envelope in the response. If the service cannot compose a reply within the requested size, a `wsman:EncodingLimit` fault should be returned with a detail code of `wsman:faultDetail/MaxEnvelopeSize`.
- R3.2-2:** If `mustUnderstand` is set to true, the service MUST comply with the request. If the response would exceed the maximum size, then a `wsman:EncodingLimit` fault should be returned. Since a service may execute the operation prior to knowing the response size, the service SHOULD undo any effects of the operation prior to issuing the fault. If the operation cannot be reversed (such as a destructive `wxf:Put` or a `wxf>Delete`, or a `wxf>Create`), the service MUST indicate that the operation succeeded in the `wsman:EncodingLimit` fault with a detail code of `wsman:faultDetail/UnreportableSuccess`.
- R3.2-3:** If `mustUnderstand` is set to false, the service MAY treat the value as advisory.
- R3.2-4:** The service SHOULD reject any value of `MaxEnvelopeSize` which is less than 4096 octets. This number is the safe minimum in which faults can be reliably encoded for all character sets. If the requested size is less than this, the service SHOULD return a `wsman:EncodingLimit` fault with a detail code of `wsman:faultDetail/MinimumEnvelopeLimit`.

Note that if the service exceeds its own encoding limit (independently of what is specified by the client), the fault is `wsman:EncodingLimit` with a detail code of `wsman:faultDetail/ServiceEnvelopeLimit`.

3.3 Locale

Management operations often span locales, and many items in responses can require

translation. Typically, this applies to descriptive information intended for human readers which is sent back in the response. If the client requires such output to be translated to a specific language, it may employ the optional `wsman:Locale` header which makes use of the standard XML attribute `xml:lang`:

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

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

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

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

R3.3-4: In any response, event, or singleton message, the service SHOULD include the **xml:lang** attribute in the `s:Envelope` (or other elements) to signal to the receiver that localized content appears in the `<Body>` of the message. This may be omitted if no descriptive content appears in the `Body`. It is not an error to always include it, even if no descriptive content occurs:

```
(1) s:Envelope
(2)   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.xmlsoap.org/ws/2005/02/management">
(6) <s:Header>
(7)   ...
```

The `xml:lang` attribute may appear on any content in the message, although it is simplest for the client to always check for it in one place, the `s:Envelope` wrapper.

3.4 Options

The *Options* header is used to pass a set of switches to server to modify or refine the nature of the request. This facility is intended to help the service to understand the nature of the output requested by the client, but *not* to reference the output or to 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.

Options

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

- R3.4-2: The service SHOULD NOT send responses, unacknowledged events, or singleton messages containing wsman:OptionSet headers, unless it is acting in the role of a client to another service. They are intended for request messages alone to which a subsequent response is expected, including acknowledged events.
- R3.4-3: An *Option* MUST NOT be used to help identify a resource or to act as an addressing modifier. It is only used to constrain the nature of the response. Therefore, if **mustUnderstand** is omitted from the Options block, the entire block MAY be ignored. If mustUnderstand is included, the service MUST comply with the entire Options block.
- R3.4-4: Each resource MAY observe its own set of options. There is no requirement to support consistent option usage across resource boundaries.
- R3.4-5: Any number of individual *Option* elements may appear under the *Options* wrapper. Option *Names* MAY be repeated if appropriate. The content MUST be a simple string (xs:string). This specification places no restrictions on whether the names or values are to be treated in a case-sensitive or case-insensitive manner.

Case usage of option names and values should be discovered by the client by retrieving the appropriate metadata from the service. The format of such metadata is beyond the scope of this specification.

- R3.4-6: Options SHOULD NOT be used as a replacement for the documented parameterization technique for the message, but SHOULD only be used as a modifier for it.
- R3.4-7: The following faults should be returned by the service:
- a) wsman:InvalidOptions with a detail code of wsman:faultDetail/NotSupported in cases where Options are not supported.
 - b) wsman:InvalidOptions with a detail code of wsman:faultDetail/InvalidName in cases where one or more options names were not valid or not supported by the specific resource.
 - c) wsman:InvalidOptions with a detail code of wsman:faultDetail/InvalidValue in cases where the value was not correct for the Option name.

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

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

Example of wsman:OptionSet:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5)   <s:Header>
(6)     ...
(7)     <wsman:OptionSet>
(8)       <wsman:Option Name="VerbosityLevel">
(9)         Level3
(10)      </wsman:Option>
(11)    </wsman:OptionSet>
(12)    ...
(13)
```

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

wsman:OptionSet

Used to wrap individual option blocks.

wsman:OptionSet/Option@Name

Identifies the option (an xs:string), which may be a simple name or a URI. This name is scoped to the resource to which it applies. The Name MAY be repeated in subsequent elements. Name cannot be blank and should be a short non-colliding URI which is vendor-specific.

wsman:OptionSet/Option

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

```
<wsman:Option Name="Verbose" />
```

Options are logically false if they are not present in the message. All other cases require an explicit string to indicate the option value. The reasoning for allowing the same option to repeat is to allow specification of a list of options of the same name. Interpretation of the option with regard to case sensitivity is up to the service and the definition of the specific option. This is because the value must usually be passed through to real-world subsystems which inconsistently expose case usage.

4.0 Resource Access

4.1 Introduction

Resource access applies to all synchronous operations regarding getting, setting, and enumerating values. The WS-Transfer specification is used as a basis for simple unary resource access: Get, Put, Delete, and Create. Multi-instance retrieval is achieved using WS-Enumeration messages. This specification does not define any messages or techniques for doing batched operations, such as batched Get or Delete. All such operations must be sent as a series of single messages.

4.2 WS-Transfer

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

A full example of a hypothetical wxf:Get request and associated response follow:

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

Note that the wsa:ReplyTo occurs on the same connection as the request (line 8), the action is a wxf:Get (line 12), and the ResourceURI (line 8) and wsman:SelectorSet (line 21) are used to address the requested management information. The operation is expected to be completed in 30 seconds or a fault should be returned to the client (line 24).

There is no s:Body in a wxf:Get request.

A hypothetical response could be:

```
(29) <s:Envelope
(30)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(31)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(32)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(33)   <s:Header>
(34)     <wsa:To>
```



```

(35)      http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(36)    </wsa:To>
(37)    <wsa:Action s:mustUnderstand="true">
(38)      http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(39)    </wsa:Action>
(40)    <wsa:MessageID s:mustUnderstand="true">
(41)      uuid:d9726315-bc91-430b-9ed8-ce5ffb858a88
(42)    </wsa:MessageID>
(43)    <wsa:RelatesTo>
(44)      uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(45)    </wsa:RelatesTo>
(46)  </s:Header>
(47)  <s:Body>
(48)    <PhysicalDisk xmlns="http://schemas.acme.com/2005/02/samples/physDisk">
(49)      <Manufacturer> Acme, Inc. </Manufacturer>
(50)      <Model> 123-SCSI 42 GB Drive </Model>
(51)      <LUN> 2 </LUN>
(52)      <Cylinders> 16384 </Cylinders>
(53)      <Heads> 80 </Heads>
(54)      <Sectors> 63 </Sectors>
(55)      <OctetsPerSector> 512 </OctetsPerSector>
(56)      <BootPartition> 0 </BootPartition>
(57)    </PhysicalDisk>
(58)  </s:Body>
(59) </s:Envelope>

```

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

The Body (on lines 47-58) contains the requested resource representation.

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

4.3 Addressing Uniformity

In general, the service should expose addressing usage which is identical for the operations. In other words, the `ResourceURI` and `wsman:SelectorSet` should be the same whether a `wxf:Get`, `wxf:Delete`, `wxf:Put`, or `wxf:Create` is being used. This is not a strict requirement, but reduces the education and training required to construct and use tools.

It is clear that `wxf:Create` is a special case, in that the `Selectors` are often not known until the resource is actually created. For example, while it may be possible to return running process information using a hypothetical *ProcessID* as a `Selector`, it is typically not possible to assert the *ProcessID* during the creation phase, so `wxf:Create` would not have the same `Selector` values as the corresponding `wxf:Get` or `wxf:Delete` (which would presumably terminate the process).

A good model to follow would be to use the same Selector values for wxf:Get, wxf:Put and wxf>Delete when working with the same instance. wsen:Enumerate should use just the ResourceURI in isolation with no Selectors, since it is going to return multiple instances, and wxf:Create would either use no Selectors or special create-only Selectors.

This usage is not a strict requirement, just a guideline. The service may use distinct Selectors for every operation and may require Selectors even for wsen:Enumerate.

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

4.4 WS-Transfer:Get

The wxf:Get is used to retrieve resource representations. The ResourceURI may map to a complex XML Infoset (an "object"), or the ResourceURI may map to a single, simple value. The nature and complexity of the representation is not constrained by this specification.

R4.4-1: A conformant service is REQUIRED to support wxf:Get in order to service metadata requests about the service itself or to verify the result of a previous action or operation.

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

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

R4.4-3: If an object cannot be retrieved, due to locking conditions, simultaneous access, or similar conflicts, a wsman:Concurrency fault should be returned.

In practice, wxf:Get is designed to return fragments or chunks of XML which correspond to real-world objects. To retrieve individual property values, the client must either postprocess the XML content for the desired value, or the service can support Fragment-level WS-Transfer (4.9).

Fault usage is generally as described in chapters 2 and 3. Not being able to locate or access the resource is equivalent to problems with the SOAP message, the ResourceURI, System, or SelectorSet. There are no 'get-specific' faults.

4.5 WS-Transfer>Delete

The WS-Transfer>Delete is used to delete resources. In general, the addressing should be the same as for a corresponding wxf:Get for uniformity, but this is not absolutely required.

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

R4.5-2: A conformant service SHOULD support wxf>Delete using the same addressing (ResourceURI, Selectors, etc.) as a corresponding wxf:Get or other messages, but this is NOT REQUIRED if the deletion mechanism for

a resource is semantically distinct.

R4.5-3: If deletion is supported and the corresponding resource can be retrieved using `wxf:Get`, a conformant service **SHOULD** support deletion using `wxf>Delete`. The service **MAY** additionally export a custom action for deletion.

R4.5-4: If an object cannot be deleted, due to locking conditions, simultaneous access, or similar conflicts, a `wsman:Concurrency` fault should be returned.

In practice, `wxf>Delete` is designed to delete entire real-world objects. To delete individual property values within an object which itself is not to be deleted, the client must either do a `wxf:Put` with those properties removed, or the service can support Fragment-Level WS-Transfer (4.9).

Fault usage is generally as described in chapters 2 and 3. Not being able to locate or access the resource is equivalent to problems with the SOAP message, `ResourceURI`, `System`, or `SelectorSet`. There are no 'delete-specific' faults.

4.6 WS-Transfer:Create

The WS-Transfer:Create is used to create resources; it models a logical "constructor". In general, the addressing is not the same as that used for `wxf:Get` or `wxf>Delete` in that the `SelectorSet` assigned to a newly created instance for subsequent access is not necessarily part of the XML content used for creating the resource. Since the `SelectorSet` may often be assigned by the service or one of its underlying systems, the `CreateResponse` must contain the applicable `SelectorSet` of the newly created instance.

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

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

R4.6-3: If a single resource can be created using a SOAP message and that resource can be subsequently retrieved using `wxf:Get`, then a service **SHOULD** support creation of the resource using `wxf:Create`. The service **MAY** additionally export a custom method for instance creation.

R4.6-4: If the supplied `Body` does not have the correct content in order for the resource to be created, the service **SHOULD** return a `wxf:InvalidRepresentation` fault and detail codes of

- a) `wsman:faultDetail/InvalidValues` if one or more values in the `<Body>` was not correct
- b) `wsman:faultDetail/MissingValues` if one or more values in the `<Body>` was missing
- c) `wsman:faultDetail/InvalidNamespace` if the wrong XML schema namespace was used and is not recognized by the service

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

Note that there is no requirement that the message body for wxf:Create use the same schema as that returned via a wxf:Get for the resource. Often, the values required to create a resource are different from those retrieved using a wxf:Get or those used for update using wxf:Put.

Note that WS-Transfer specifies that the wxf:CreateResponse must contain the initial representation of the object. The ability to return the new representation in an interlocked fashion is distinct from doing a subsequent wxf:Get, as the values may change due to other out-of-band operations. It is often important for the client to know the result of the create operation, independent of any other clients subsequently acting on that same resource.

If a service needs to support creation of individual values within a representation (property-level creation, array insertion, etc.), then it should support Fragment-Level WS-Transfer (4.9).

Since the values in a the SelectorSet may be assigned by the service and may not be part of the wxf:Create representation, they must be returned in the wxf:CreateResponse message if they are required for subsequent access.

All applicable Selectors should be returned, even if not all of them are required for subsequent access. Similarly, the ResourceURI used to create the object may not even be the one used to retrieve it subsequently using wxf:Get.

R4.6-6: The wxf:CreateResponse to a wxf:Create message MUST contain the new endpoint reference of the created resource in the wxf:ResourceCreated element, including the applicable SelectorSet. The wsa:Address of that resource SHOULD be simply copied from the wsa:To address of the original request, suffixed with the ResourceURI. The anonymous address from WS-Addressing MAY be used to communicate an address which is not transport-specific.

R4.6-7: The response MUST contain the initial representation of the object, as the first element after the wxf:ResourceCreated element. The service SHOULD NOT add additional elements after the element containing the initial representation.

Hypothetical example of a response for a newly created virtual drive:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsmn="http://schemas.xmlsoap.org/ws/2005/02/management"
(5)   xmlns:wxf="http://schemas.xmlsoap.org/ws/2004/09/transfer">
(6) <s:Header>
(7)   ...
(8) <wsa:Action>
(9)   http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse
(10) </wsa:Action>
(11)   ...
(12) <s:Body>
(13)   <wxf:ResourceCreated>
(14)     <wsa:Address>
(15)       http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous?
(16)       ResourceURI=(wsmn:samples.org/2005/02/virtualDrive)
(17)     </wsa:Address>
```

```

(18)     <wsa:ReferenceParameters>
(19)         <wsman:SelectorSet>
(20)             <wsman:Selector Name="ID"> F: </wsman:Selector>
(21)         </wsman:SelectorSet>
(22)     </wsa:ReferenceParameters>
(23) </wxf:ResourceCreated>
(24)
(25)     <Drive xmlns="samples.org/2005/02/virtualDrive">
(26)         <ID> F: </ID>
(27)         <Freespace> 123000140 </Freespace>
(28)         <PhysicalDiskLUN> 3 </PhysicalDiskLUN>
(29)     </Drive>
(30)
(31) </s:Body>

```

Note that the response contains two sections, the wxf:ResourceCreated (lines 13-23) which contains the new endpoint reference of the created resource, including its ResourceURI and the correct SelectorSet, and the initial representation (lines 25-29) of the object. This is essentially the same representation that would be returned by an immediate subsequent wxf:Get. Since values can change in between the creation and subsequent retrieval using wxf:Get (because of multi-user access or highly dynamic values), it is not guaranteed that a wxf:Get would contain the same values as returned in the wxf:Create.

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

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

This is to allow wxf:Get, wxf:Put and wxf:Create to share the same schema. Note that wxf:Put also allows the service to ignore read-only properties during an update.

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

4.7 WS-Transfer:Put

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

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

R4.7-2: If a single management resource can be updated (within the constraints of its

schema) using a SOAP message and that resource can be subsequently retrieved using wxf:Get, then a service SHOULD support update of the resource using wxf:Put. The service MAY additionally export a custom method for updates.

R4.7-3: If a single management resource contains a mix of read-only and read-write values, the wxf:Put message MAY contain both the read-only and read-write values, subject to the legality of the XML content with regard to its XML schema namespace. In such cases, the service MUST ignore the read-only values during the update operation. If none of the values are writeable, the service SHOULD return a wsman:ActionNotSupported fault.

Note that if a service wishes to support wxf:Put in which read-only values should not appear, there are several solutions. First, the schema can ensure that read-only values are always optional, so that during the wxf:Put they can be omitted. Alternately, a separate schema for wxf:Put (as opposed to the one used by wxf:Get) can be defined. Finally, fragment-level transfer can be used (see 4.9).

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

R4.7-5: If the supplied Body does not have the correct content in order for the resource to be created, the service SHOULD return a wxf:InvalidRepresentation fault and detail codes of

- a) wsman:faultDetail/InvalidValues if one or more values in the s:Body was not correct
- b) wsman:faultDetail/MissingValues if one or more values in the s:Body was missing
- c) wsman:faultDetail/InvalidNamespace if the wrong XML schema namespace was used and is not recognized by the service

R4.7-6: If an object cannot be updated, due to locking conditions, simultaneous access, or similar conflicts, a wsman:Concurrency fault should be returned.

Note that it is not always legal to only supply the values which will be updated in a wxf:Put request, as the XML schema would have to allow the values to be missing, and it may not be defined that way. The Body of the wxf:Put message must not violate the constraints of the associated XML schema. For example, if a wxf:Get would return

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

And the corresponding XML schema defined A, B, and C as minOccurs="1",

```
(8) <xs:element name="MyObjecct">
(9)   <xs:complexType>
```

```

(10)      <xs:sequence>
(11)          <xs:element name="A" type="xs:int" minOccurs="1" maxOccurs="1"/>
(12)          <xs:element name="B" type="xs:int" minOccurs="1" maxOccurs="1"/>
(13)          <xs:element name="C" type="xs:int" minOccurs="1" maxOccurs="1"/>
(14)      ...

```

...then the corresponding wxf:Put must have all three elements, since the schema mandates that all three be present. Even if the only value being updated is , the client would have to supply all three values. This usually means that the client would have to issue a wxf:Get first, in order to preserve the current values of <A> and <C>, change to the desired value, and then write the object using wxf:Put. As noted in R4.7-3, the service should ignore attempts to update values which are read-only with regard to the underlying real-world object.

Because of this problem, service may expose Fragment-level WS-Transfer (see 4.9).

- R4.7-7:** A wxf:Put operation SHOULD NOT result in a change to the endpoint reference (ResourceURI and Selectors) for the resource. The wsman:Rename operation should be used for this.
- R4.7-8:** The service MAY return the new representation in all cases. It is often difficult to know whether the new representation is different than the requested update. This is because resource-constrained implementations may not have sufficient resources to determine the equivalence of the requested update from the result.
- R4.7-9:** If the success of an operation cannot be reported as described in this section and cannot be reversed, a wsman:EncodingLimit fault with a detail code of wsman:faultDetail/UnreportableSuccess should be returned.

4.8 WS-Management:Rename

Renaming a resource is a common operation.

The wxf:Put operation should not be used to indirectly effect a rename for two reasons: (a) renaming is a serious operation and users may not be aware that changing some values in a wxf:Put actually constitutes a rename, (b) the name or identity of a resource may not be part of its representation, so there is no way to specify the new name in the wxf:Put body.

In WS-Management, renaming is limited to reassigning the wsman:SelectorSet values for the resource.

- R4.8-1:** A conformant service is NOT REQUIRED to support wsman:Rename.
- R4.8-2:** If a service implements wsman:Rename, it MUST implement the following message and its associated response and the rename SHOULD NOT have any side-effect on the representation other than the effects of the rename itself. The rename may logically constitute a "move" as well as a rename, depending on the requirements of the service.

Neither the ResourceURI nor the wsman:System values can be changed. Likewise, singleton resources with no wsman:SelectorSet cannot be renamed.

Note that the rename operation references the 'old' identity of the object in the same

manner as wxf:Get, but the new identity of the resource is in the s:Body of the message, using a wsman:Rename block:

```
(1) <s:Envelope ...>
(2)   <s:Header>
(3)     <wsa:To> networkAddress?ResourceURI=(uri) </wsa:To>
(4)     <wsa:Action s:mustUnderstand="true">
(5)       http://schemas.xmlsoap.org/ws/2005/02/management/Rename
(6)     </wsa:Action>
(7)     <wsman:SelectorSet> ... </wsman:SelectorSet>
(8)   </s:Header>
(9)   <s:Body>
(10)     <wsman:Rename>
(11)       new endpoint-reference
(12)     </wsman:Rename>
(13)   </s:Body>
(14) </s:Envelope>
```

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

wsa:Action

MUST be **http://schemas.xmlsoap.org/ws/2005/02/management/Rename**.

wsman:Rename

This s:Body element wraps the new identity of the resource and is of type wsa:EndpointReference. This contains the original wsa:To and wsman:System values used to address the item, and the *new* wsman:SelectorSet which applies.

R4.8-3 A conformant service which implements wsman:Rename MUST accept an EPR in the wsman:Rename block which uses a wsa:To address with the anonymous endpoint address:

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

...suffixed with the ResourceURI.

The service MUST alternately accept an address which matches the wsa:To address of the request itself, but should reject any wsa:To which is not one of these two possibilities with a wsman:RenameFailure fault with a detail code of wsman:faultDetail/InvalidAddress.

R4.8-4 A conformant service which implements wsman:Rename MAY allow any or all of the wsman:Selector values to change during the rename. The other EPR values MUST not be changed, i.e., the wsman:System and wsa:To values.

R4.8-5 In addition to faults related to EPR usage in finding and locating the resource to be renamed, the service SHOULD return a wsman:RenameFailure fault using one of the following detail codes if possible:

- a) wsman:faultDetail/InvalidSelectorAssignment if the new Selectors cannot be applied to rename the resource.
- b) wsman:faultDetail/InvalidResourceURI if the ResourceURI was not correct.
- c) wsman:faultDetail/InvalidSystem if the System was not correct.

- d) `wsman:faultDetail/AlreadyExists` if the resource under the requested EPR naming already exists.

This mechanism is designed to be forward-compatible with allowing other aspects of the EPR to be renamed in a future version of this specification. For this version, however, only the Selector values may be legally changed.

The response to a `wsman:Rename` is a `wsman:RenameResponse`, which contains the new EPR of the resource.

The format of the response is as follows:

```
(15) <s:Envelope>
(16)   <s:Header>
(17)     ...
(18)   <wsa:Action s:mustUnderstand="true">
(19)     http://schemas.xmlsoap.org/ws/2005/02/management/RenameResponse
(20)   </wsa:Action>
(21)     ...
(22)   <s:Body>
(23)     <wsman:RenamedTo>
(24)       endpoint reference
(25)     </wsman:RenamedTo>
(26)   </s:Body>
(27) </s:Envelope>
```

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

`wsa:Action`

MUST be "`http://schemas.xmlsoap.org/ws/2005/02/management/RenameResponse`".

`wsman:RenamedTo`

Must contain the resultant endpoint reference for the resource: the `wsa:To` with the ResourceURI suffix, and any applicable `wsman:System` and `wsman:SelectorSet` values. In practice, only the Selector values change, although it is convenient to have the full EPR represented for easy composition with subsequent operations.

R4.8-6 A conformant service MUST return the full, new endpoint reference in all cases, regardless of which Selector elements of the EPR changed. The `wsa:To` address SHOULD be the same as the `wsa:To` address which would be used to subsequently execute a `wxf:Get` against the resource, but MAY consist of the anonymous address **`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`** suffixed by the ResourceURI.

There is no implication that the rename takes immediate effect. Services typically will hold off the response until the rename is completed, but this is not a strict requirement.

The following are hypothetical examples of a `Rename` and `RenameResponse` in which a disk drive is renamed from C: to D: :

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
```

```

(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5) <s:Header>
(6)   <wsa:To>
(7)     http://1.2.3.4/wsman?ResourceURI=(wsman:samples.org/2005/02/LogicalDisk)
(8)   </wsa:To>
(9)   <wsa:ReplyTo>
(10)    <wsa:Address> http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(11)    </wsa:Address>
(12)  </wsa:ReplyTo>
(13)  <wsa:Action>
(14)    http://schemas.xmlsoap.org/ws/2005/02/management/Rename
(15)  </wsa:Action>
(16)  <wsa:MessageID>
(17)    uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(18)  </wsa:MessageID>
(19)  <wsman:SelectorSet>
(20)    <wsman:Selector Name="Drive"> C: </wsman:Selector>
(21)  </wsman:SelectorSet>
(22) </s:Header>
(23) <s:Body>
(24)   <wsman:Rename>
(25)     <wsa:EndpointReference>
(26)       <wsa:Address>
(27)         http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous?
(28)         ResourceURI=(wsman:samples.org/2005/02/LogicalDisk)
(29)       </wsa:Address>
(30)     <wsman:SelectorSet>
(31)       <wsman:Selector Name="Drive"> D: </wsman:Selector>
(32)     </wsman:SelectorSet>
(33)   </wsa:EndpointReference>
(34) </wsman:Rename>
(35) </s:Body>
(36) </s:Envelope>

```

Note that the address of the item to be renamed follows the normal pattern in the header. Only the <Body> is different in that the new name is specified. Only the parts of the EPR that need to be changed are present, the wsman:SelectorSet in this case.

Note the use of the special wsa:Address value of **http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous**

...to act as a placeholder for the real address, suffixed by the ResourceURI. The actual transport address could also legally be used.

In this example, only the Selector indicating the drive letter is changed from the "C:" indicated by the the new value on line 31.

The response indicates the new EPR:

```

(37) <s:Envelope
(38)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(39)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"

```

```

(40)     xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(41)   <s:Header>
(42)     <wsa:To>
(43)       http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(44)     </wsa:To>
(45)     <wsa:Action s:mustUnderstand="true">
(46)       http://schemas.xmlsoap.org/ws/2005/02/management/RenameResponse
(47)     </wsa:Action>
(48)     <wsa:MessageID s:mustUnderstand="true">
(49)       uuid:d9726315-bc91-430b-9ed8-ce5ffb858a88
(50)     </wsa:MessageID>
(51)     <wsa:RelatesTo>
(52)       uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(53)     </wsa:RelatesTo>
(54)   </s:Header>
(55)   <s:Body>
(56)     <wsman:RenamedTo>
(57)       <wsa:EndpointReference>
(58)         <wsa:Address>
(59)           http://1.2.3.4/wsman?ResourceURI=(wsman:samples.org/2005/02/LogicalDisk)
(60)         </wsa:Address>
(61)         <wsman:SelectorSet>
(62)           <wsman:Selector Name="Drive"> D: </wsman:Selector>
(63)         </wsman:SelectorSet>
(64)       </wsa:EndpointReference>
(65)     </wsman:RenamedTo>
(66)   </s:Body>
(67) </s:Envelope>

```

Note that the response contains the new EPR in its entirety, ready to use in a new `wxf:Get` (after the required transformation discussed in 2.2). Note specifically that the `ResourceURI` is added back into the new EPR (line 59) even though it was not part of the rename, and the `wsa:Address` is the actual address that the client would use to retrieve the resource, not the "anonymous" role URI.

4.9 Fragment-Level WS-Transfer

Because WS-Transfer works with entire instances and it may be inconvenient to specify hundreds or thousands of EPRs just to model property-level access with full EPRs, WS-Management supports the concept of fragment-level (property) access of resources that are normally accessed via WS-Transfer operations. This is done using special usage of WS-Transfer.

Because of the XML schema limitations discussed in 4.7, it is often incorrect to simply return a subset of the XML defined for the object being accessed, as a subset may violate the XML schema for that fragment. In order to support transfer of fragments or individual elements of a representation object, several modifications to the basic WS-Transfer operations are made.

R4.9-1 A conformant service is NOT REQUIRED to support Property-Level WS-Transfer. The service MUST NOT behave as if normal WS-Transfer operations were in place, but MUST operate exclusively on the fragments specified. If the service does not support fragment-level access, it MUST return a wsman:UnsupportedFeature fault with a detail code of wsman:FragmentLevelAccess.

R4.9-2 A conformant service which supports fragment-level WS-Transfer MUST accept the following SOAP header in all requests and include it in all responses which transport the fragments:

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

This header may only appear once and mustUnderstand MUST be set to true, as a special XML usage is in progress. In this manner, both the service and the client can be certain that a special mode of transfer is in progress. The default value of this header is the XPath 1.0 Selector of the fragment being transferred (using <http://www.w3.org/TR/1999/REC-xpath-19991116>), with relation to the full representation of the object. If a value other than XPath 1.0 is being used, a Dialect attribute may be added to indicate this:

```
(4) <wsman:FragmentTransfer mustUnderstand="true"
(5)   Dialect="URIToNewSelectorDialect">
(6)   dialect text
(7) </wsman:FragmentTransfer>
```

Note that XPath is special-cased due to its importance, but it is not mandated. Any other type of language to describe fragment-level access is permitted as long as the Dialect value is set to indicate to the service what dialect is being used.

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

R4.9-3 All transfer in either direction of the XML fragments must be wrapped with a <wsman:XmlFragment> wrapper which contains a definition which suppresses validation and allows any content to pass. A service MUST reject any attempt to use wsman:FragmentTransfer unless the s:Body wraps the content using a wsman:XmlFragment wrapper. If any other usage is encountered, the service MUST fault the request using a wxf:InvalidRepresentation fault with a detail code of wsman:faultDetail/InvalidFragment.

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

R4.9-4 If fragment-level WS-Transfer is supported, a conformant service SHOULD at least support leaf-node value-level access using an XPath with a /Text()

specifier. In this case, the value is not wrapped with XML, but is transferred directly as text within the wsman:XmlFragment wrapper.

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

R4.9-5 For all fragment-level operations, partial successes are NOT permitted. The entire meaning of the XPath or other dialect MUST be fully observed by the service in all operations and the entire fragment that is specified MUST be successfully transferred in either direction. Otherwise, faults occur as if none of the operation had succeeded.

All faults are as for normal, "full" WS-Transfer operations.

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

4.10 Fragment-Level WS-Transfer:Get

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

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

In this case, the service will execute the specified XPath against the representation that

would normally have been retrieved, and then return a fragment instead.

Note that the **wsman:FragmentTransfer** MUST be repeated in the wxf:GetResponse (line 47-49) by the service to reference the fragment and to signal that a fragment has been transferred, and that the response is wrapped in a wsman:XmlFragment wrapper, which suppresses the schema validation which would otherwise apply:

```
(30) <s:Envelope
(31)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(32)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(33)     xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(34)   <s:Header>
(35)     <wsa:To>
(36)       http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(37)     </wsa:To>
(38)     <wsa:Action s:mustUnderstand="true">
(39)       http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(40)     </wsa:Action>
(41)     <wsa:MessageID s:mustUnderstand="true">
(42)       uuid:d9726315-bc91-430b-9ed8-ce5ffb858a88
(43)     </wsa:MessageID>
(44)     <wsa:RelatesTo>
(45)       uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(46)     </wsa:RelatesTo>
(47)     <wsman:FragmentTransfer mustUnderstand="true">
(48)       PhysicalDisk/Manufacturer
(49)     </wsman:FragmentTransfer>
(50)   </s:Header>
(51)   <s:Body>
(52)     <wsman:XmlFragment>
(53)       <Manufacturer> Acme, Inc. </Manufacturer>
(54)     </wsman:XmlFragment>
(55)   </s:Body>
(56) </s:Envelope>
```

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

If the client wishes to receive the value in isolation without an XML wrapper, XPath techniques, such as using the Text() operator can be used to retrieve just the values. The following request

```
(57) <wsman:FragmentTransfer mustUnderstand="true">
(58)   PhysicalDisk/Manufacturer/Text()
(59) </wsman:FragmentTransfer>
```

...which will yield this XML:

```
(60) <wsman:XmlFragment>
(61)   Acme, Inc.
(62) </wsman:XmlFragment>
```

4.11 Fragment-Level WS-Transfer:Put

Fragment-level WS-Transfer:Put works just like regular wxf:Put except that only the part that is being updated is transferred. Continuing from the example in 4.2 and 4.10, if the client wanted to update the <BootPartition> value from 0 to 1, then the following wxf:Put fragment could be sent to the service:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5) <s:Header>
(6)   <wsa:To>
(7)     http://1.2.3.4/wsman?ResourceURI=(wsman:samples.org/2005/02/physicalDisk)
(8)   </wsa:To>
(9)   <wsa:ReplyTo>
(10)    <wsa:Address>
(11)      http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(12)    </wsa:Address>
(13)  </wsa:ReplyTo>
(14)  <wsa:Action>
(15)    http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
(16)  </wsa:Action>
(17)  <wsa:MessageID>
(18)    uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(19)  </wsa:MessageID>
(20)  <wsman:SelectorSet>
(21)    <wsman:Selector Name="LUN"> 2 </wsman:Selector>
(22)  </wsman:SelectorSet>
(23)  <wsman:OperationTimeout> PT30S </wsman:OperationTimeout>
(24)  <wsman:FragmentTransfer mustUnderstand="true">
(25)    PhysicalDisk/BootPartition
(26)  </wsman:FragmentTransfer>
(27) </s:Header>
(28) <s:Body>
(29)   <wsman:XmlFragment>
(30)     <BootPartition> 1 </BootPartition>
(31)   </wsman:XmlFragment>
(32) </s:Body>
(33) </s:Envelope>
```

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

```
(34) ...
(35) <wsman:FragmentTransfer mustUnderstand="true">
(36)   PhysicalDisk/BootPartition/Text()
(37) </wsman:FragmentTransfer>
(38) </s:Header>
(39) <s:Body>
(40)   <wsman:XmlFragment>
```

```
(41)      1
(42)      </wsman:XmlFragment>
(43)      </s:Body>
```

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

A sample reply:

```
(44)      <s:Envelope
(45)          xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(46)          xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(47)          xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(48)      <s:Header>
(49)          <wsa:To>
(50)              http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(51)          </wsa:To>
(52)          <wsa:Action s:mustUnderstand="true">
(53)              http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse
(54)          </wsa:Action>
(55)          <wsa:MessageID s:mustUnderstand="true">
(56)              uuid:d9726315-bc91-430b-9ed8-ce5ffb858a88
(57)          </wsa:MessageID>
(58)          <wsa:RelatesTo>
(59)              uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(60)          </wsa:RelatesTo>
(61)          <wsman:FragmentTransfer> PhysicalDisk/BootPartition/Text() </wsman:FragmentTransfer>
(62)      </s:Header>
(63)      <s:Body>
(64)          <wsman:XmlFragment>
(65)              1
(66)          </wsman:XmlFragment>
(67)      </s:Body>
(68)      </s:Envelope>
```

R4.11-1 As for normal wxf:Put, the service MAY ignore any read-only values supplied as part of the fragment for update.

R4.11-2 If the service encounters an attempt to update a read-only value, a wsman:ActionNotSupported fault is returned with a detail code of wsman:faultDetail/ReadOnly

Note that fragment-level Put implies replacement or update and does not insert new values into the representation object. WS-Transfer states that if the new representation differs from the input, then it should be returned in the response. With fragment-level wxf:Put, this only applies to the portion of the representation object being written, not the entire

object. If a single value is written and it accepted, but has side-effects on other values in the representation, the entire object is *not* returned.

To set a value to NULL without removing it as an element, use a value of `xsi:nil`:

```
(69) <s:Body>
(70)   <wsman:XmlFragment xmlns:xsi="www.w3.org/2001/XMLSchema-instance">
(71)     xsi:nil
(72)   </wsman:XmlFragment>
(73) </s:Body>
```

4.12 Fragment-Level WS-Transfer:Create

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

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

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

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

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

To create the target using value alone, the XPath `Text()` operator can be applied to the path:

```
(10) <wsman:FragmentTransfer mustUnderstand="true">
(11)   LogicalDisk/VolumeLabel/Text()
(12) </wsman:FragmentTransfer>
```

The body of the `wxf:Create` contains the value to be insert and is the same as for fragment-level `wxf:Put`:

```
(13) <s:Body>
(14)   <wsman:XmlFragment>
```

```

(15)         MyDisk
(16)         </wsman:XmlFragment>
(17)    </s:Body>
(18)    </s:Envelope>

```

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

```

(19)    <wsman:FragmentTransfer mustUnderstand="true">
(20)         InternetServer/BlockedIPAddress[3]
(21)    </wsman:FragmentTransfer>

```

The body of the wxf:Create contains the value to be insert and is the same as for fragment-level wxf:Put:

```

(22)    <s:Body>
(23)         <wsman:XmlFragment>
(24)             <BlockedIPAddress> 123.12.188.44 </BlockedIPAddress>
(25)         </wsman:XmlFragment>
(26)    </s:Body>
(27)    </s:Envelope>

```

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

R4.12-1: A service MUST not use wxf:Create to perform an update on an existing representation. The targeted object must not already exist, or else the service SHOULD return a wsman:AlreadyExists fault.

R4.12-2: If the wxf:Create fails because the result would not conform to the schema in some way, a wxf:InvalidRepresentation fault is returned.

4.13 Fragment-Level WS-Transfer:Delete

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

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

```

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

```

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

To delete an array element, the XPath [] operators may be used. The following example deletes the third <User> element in the representation:

```
(4) <wsman:FragmentTransfer mustUnderstand="true">  
(5)   LogicalDisk/User[3]  
(6) </wsman:FragmentTransfer>
```

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

R4.13-1: If a value cannot be deleted, due to locking conditions or similar phenomena, a wsman:AccessDenied fault SHOULD be returned.

5.0 WS-Enumeration

5.1 Introduction

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

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

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

R5.1-3: If filtered enumeration (queries) of the instances of a resource is exposed via Web services, a conformant service SHOULD support WS-Enumeration to expose this. The service MAY also support other techniques for enumerating the instances.

The WS-Enumeration specification indicates that enumeration is a three-part operation: An initial wse:Enumerate is issued to establish the enumeration context and wse:Pull operations are used to iterate over the result set. When the enumeration iterator is no longer required and not yet exhausted, a wse:Release is issued to release the enumerator and associated resources. As with other WS-Management methods, the enumeration may make use of wsman:OptionSet, and may make use of wsman:SelectorSet to scope the operation, although typically the ResourceURI is used alone without Selectors.

R5.1-4: A service is NOT REQUIRED to implement any of the following messages from WS-Enumeration and implementing them is NOT RECOMMENDED: Renew, GetStatus, or EnumerationEnd or any associated responses. Since these messages are OPTIONAL, it is RECOMMENDED that the

service fault both Renew and GetStatus requests with a wsa:ActionNotSupported fault.

- R5.1-5:** If a service is exposing enumeration, it MUST at least support the following messages: wsen:Enumerate, wsen:Pull, and wsen:Release, and their associated responses.

If the service does not support stateful enumerators, the Release may be a simple no-op in reality so it is trivial to implement (it always succeeds when a valid operation). But it must be supported in any case to allow for the uniform construction of clients.

5.2 WS-Enumeration:Enumerate

- R5.2-1:** A conformant service is NOT REQUIRED to accept a wsen:Enumerate message with an EndTo address, as implied by R5.1-4 and may issue a wsman:UnsupportedFeature fault with a detail code of wsman:faultDetail/AddressingMode.
- R5.2-2:** A conformant service MUST accept a wsen:Enumerate message with an Expires timeout, but MAY always fault with wsman:UnsupportedFeature and a detail code of wsman:faultDetail/ExpirationTime.
- R5.2-3:** The Filter element in the wsen:Enumerate body MUST be either simple text or a single complex XML element. A conformant service MUST NOT accept mixed content of both text and elements, or multiple peer XML elements under the Filter element.

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

A common filter dialect is <http://www.w3.org/TR/1999/REC-xpath-19991116>, which is XPath 1.0. Resource constrained implementations may find it difficult to export full XPath processing and yet still wish to use a subset of XPath syntax. As long as the filter expression is a proper subset of the specified dialect, it is legal and may be described using that Dialect value

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

- R5.2-4:** A conformant service is NOT REQUIRED to support the entire syntax and processing power of the specified Filter Dialect. The only requirement is that the specified Filter is syntactically correct within the definition of the Dialect. Subsets are therefore legal. If the specified filter exceeds the capability of the service, a wsen:CannotProcessFilter fault should be returned with some text indication as to what went wrong.

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

R5.2-4: A conformant service MUST fault any request without a `wsen:Filter` if a `wsen:Filter` is actually required, using a `wsman:UnsupportedFeature` fault, with a detail code of `wsman:FaultDetail/FilteringRequired`.

5.3 Filter Intepretation

Filters are generally intended to select entire XML infosets or "object" representations. However, most query languages have both filtering and compositional capabilities in that they can return subsets of the original representation, or perform complex operations on the original representation and return something entirely new.

This specification places no restriction on the capabilities of the service, but services may elect to only provide simple filtering capability and no compositional capabilities. In general, filtering dialects fall into the following simple hierarchy:

- 1) Simple enumeration with no filtering
- 2) Filtered enumeration with no representation change (within the capabilities of XPath, for example)
- 3) Filtered enumeration in which a subset is selected (within the capabilities of XPath, for example)
- 4) Composition of new output (XQuery)

Most services will fall into (1) or (2). However, if a service wishes to support fragment-level enumeration to complement fragment-level WS-Transfer (4.9), then the service should implement (3) as well. Only rarely will services implement (4).

Note that XPath 1.0 can be used simply for filtering, or may be used to send back subsets of the representation (or even the values without XML wrappers). In cases where the result is not just filtered but being "altered", the technique in 5.6 applies.

5.4 WS-Enumeration:Pull

The `wsen:Pull` message is used to continue an enumeration, i.e., retrieve batches of results from the initial `wsen:Enumerate`.

Since `wsen:Pull` allows the client to specify a wide range of batching and timing parameters, it is often advisable for the client to know ahead of time what the valid ranges are. This should be exported from the service in the form of metadata, which is beyond the scope of this specification. There is no message-based negotiation for discovering the valid ranges of the parameters.

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

R5.4-1: If a service is exposing enumeration and supports WS-Enumeration:Pull with the `MaxCharacters` element, the SERVICE SHOULD implement this as a general guideline or hint but MAY ignore it if `wsman:MaxEnvelopeSize` is

present, since that takes precedence. The service SHOULD NOT fault in the case of a conflict but SHOULD observe the wsman:MaxEnvelopeSize value.

R5.4-2: If a service is exposing enumeration and supports WS-Enumeration:Pull with the MaxCharacters element and a single response element would cause the limit to be exceeded, the service MAY return the single element in violation of the hint. However, the service MUST NOT violate wsman:MaxEnvelopeSize in any case.

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

- a) wsman:faultDetail/MaxEnvelopeSize if the client's requested maximum would have been exceeded
- b) wsman:faultDetail/ServiceEnvelopeLimit if the service's internal limit was exceeded

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

R5.4-4: If any fault occurs during a wsen:Pull, a compliant service SHOULD allow the client to retry wsen:Pull with other parameters, such as a larger limit or with no limit and attempt to retrieve the items. The service SHOULD not cancel the enumeration as a whole, but retain enough context to be able to retry if the client so wishes. However, the service MAY cancel the enumeration outright if an error occurs with a wsen:InvalidEnumerationContext fault.

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

However, EnumerationContexts from previous wsen:PullResponse messages must not be reused and are not expected to be considered valid by the service. The EnumerationContext from only the latest response is considered to be valid. While the service may return the same EnumerationContext values with each wsen:Pull, it is not required to do so and may in fact change the EnumerationContext unpredictably.

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

It is recommended that clients omit wsen:MaxTime and use only wsman:OperationTimeout and that wman:MaxEnvelopeSize be used in preference to wsen:MaxCharacters.

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

R5.4-6: The service **MUST** accept a wsen:Pull message with an endpoint reference identical to that specified for the original wsen:Enumerate. A wsa:MessageInformationHeaderRequired fault should be returned if the EPR is missing or different.

If there is no content available, the enumerator is still considered active and the Pull may be retried:

R5.4-7: If a service cannot populate the wsen:PullResult with any items before the timeout, it **SHOULD** return a wsman:TimedOut fault to indicate that true timeout conditions occur and that the client is not likely to succeed by simply issuing another wsen:Pull. If the service is simply waiting for results at the point of the timeout, it **SHOULD** return a response with no items and an updated wsen:EnumerationContext, which **MAY** have changed, even though no items were returned:

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

An empty wsen:Items block is essentially a directive from the service to try again. If the service faults with a wsman:TimedOut fault, the implication is that a retry is not likely to succeed. Typically, the service will know which one to return based on its internal state. For example, on the very first wsen:Pull if the service is waiting for another component, then a wsman:TimedOut fault may be likely. If the enumeration is continuing with no problem and after 50 requests a particular wsen:Pull times out, the service may simply send back zero items in the expectation that the client should continue with another wsen:Pull.

R5.4-8: The service **MAY** terminate the entire enumeration early at any time, in which case a wsen:InvalidEnumerationContext fault is returned. No further operations are possible, including wsen:Release. In specific cases, such as internal errors or responses which are too large, other faults may also be returned. In all such cases, the service **SHOULD** invalidate the enumeration context as well.

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

Normally, the end of an enumeration in all cases is reported by the wsen:EndOfSequence

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

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

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

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

5.5 WS-Enumeration:Release

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

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

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

R5.5-2: The client may fail to deliver the wsen:Release in a timely fashion or may never send it. A conformant service MAY terminate the enumeration after a suitable idle time has expired and any attempt to reuse the enumeration context MUST result in a wsen:InvalidEnumerationContext fault.

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

5.6 Ad-Hoc Queries and Fragment-Level Enumerations

As discussed in 4.9, it is desirable that clients should be able to access subsets of a representation. This is especially important in the area of query processing, where users routinely wish to execute XPath or XQuery operations over the representation to receive ad-hoc results.

Since SOAP messages must conform to known schemas and since ad-hoc queries return results which are dynamically generated and may conform to no schema, the wsman:XmlFragment wrapper from 4.9 must be used to wrap the responses.

R5.6-1: The service MAY support ad-hoc compositional queries or enumerations of fragments of the representation objects by supplying a suitable dialect in the `wsen:Filter`. The resulting set of Items in the `wsen:Pull` should be wrapped with `wsman:XmlFragment` wrappers:

```
(1) ...
(2) <s:Body>
(3)   <wsen:PullResponse>
(4)     <wsen:EnumerationContext> ...possibly updated... </wsen:EnumerationContext>
(5)     <wsen:Items>
(6)       <wsman:XmlFragment>
(7)         XML content
(8)       </wsman:XmlFragment>
(9)       <wsman:XmlFragment>
(10)        XML content
(11)      </wsman:XmlFragment>
(12)      ...etc.
(13)    </wsen:Items>
(14)  </wsen:PullResponse>
(15) </s:Body>
```

The schema for `wsman:XmlFragment` contains a directive to suppress schema validation, allowing a validating parser to accept ad-hoc content produced by the query processor acting behind the enumeration.

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

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

6.0 Custom Actions (Methods)

6.1 General

Custom actions or 'methods' are nothing more than ordinary SOAP messages with unique Actions. An implementation may support resource-specific methods in any form, subject to the addressing model and restrictions described in section 2.0 of this specification.

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

R6.1-2: If custom methods are exported, the System, ResourceURI, Selectors and other header usages defined in this specification MUST be observed in the addressing model as specified in section 2.0, and each custom method MUST have a unique `wsa:Action`.

Thus, a custom method must be directed to a specific Resource (using the System, ResourceURI and Selectors if required) and may not rely entirely on a wsa:To address, or else the EPR translation mechanism in 2.14 can be used.

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

This specification places no restrictions on Options being used with custom WSDL-based operations, however.

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

It is not an error to utilize the wsa:Action URI for the ResourceURI of a custom method, but both will still be required in the message for uniform processing on both clients and servers. For example, the following action to reset a network card may have the following EPR usage:

```
(1) <s:Header>
(2)   <wsa:To>
(3)     http://1.2.3.4/wsman?ResourceURI=(http://acme.com/2005/02/networkcards/reset)
(4)   </wsa:To>
(5)   <wsa:Action>
(6)     http://acme.com/2005/02/networkcards/reset
(7)   </wsa:Action>
(8)   ...
```

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

```
(9) <s:Header>
(10)  <wsa:To>
(11)    http://1.2.3.4/wsman?ResourceURI=(http://acme.com/2005/02/networkcards)
(12)  </wsa:To>
(13)  <wsa:Action>
(14)    http://acme.com/2005/02/networkcards/reset
(15)  </wsa:Action>
(16)  ...
```

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

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

All of these are legal usage.

7.0 Eventing

7.1 General

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

- R7.1-1:** If a Resource can emit events and allows clients to subscribe to and receive event messages, it **MUST** do so by implementing WS-Eventing as specified in this specification.
- R7.1-2:** If WS-Eventing is supported, the Subscribe, Renew, and Unsubscribe messages **MUST** be supported. SubscriptionEnd is **OPTIONAL**, and GetStatus is **NOT RECOMMENDED**.

7.2 Subscribe

7.2.1 General

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

- R7.2.1-1:** The identity of the event source **MUST** be based on the ResourceURI suffix, wsman:System (optional), and wsman:SelectorSet.
- R7.2.1-2:** A service is **NOT REQUIRED** to support distinct addresses and distinct security settings for NotifyTo and EndTo, and **MAY** require that these be the same network address, although they **MAY** have separate reference properties and reference parameters in all cases. If the service cannot support the requested addressing, it **SHOULD** return a wsman:UnsupportedFeature fault with a detail code of wsman:faultDetail/AddressingMode.
- R7.2.1-3:** Because many delivery modes require a separate connection to deliver the event, the service **SHOULD** comply with the security profiles defined in section 9 of this specification if HTTP or HTTPS is used to deliver events. If no security is specified, the service **MAY** attempt to use default security mechanisms, or return a wse:UnsupportedFeature fault with a detail code of wsman:faultDetail/InsecureAddress.

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

- R7.2.1-4:** A service is **REQUIRED** to deliver any reference properties and reference parameters in the wse:NotifyTo address with each event delivery as specified in 2.2 of this specification. If EndTo is supported, this behavior applies as well.

As with other WS-Management operations, the endpoint reference described by the ResourceURI, wsman:System, and any applicable Selectors identify the event source to which the subscription is directed. In many cases, the ResourceURI identifies a real or virtual event log and the subscription is intended to provide real-time notifications of any new entries added to the log. In many cases, the wsman:SelectorSet element may not be used as part of the endpoint reference.

If a client needs to have events delivered to more than one destination, more than one subscription is required.

7.2.2 Filtering

The observations on the interpretation of the filter described in 5.3 also apply here.

The standard wse:Filter dialect is <http://www.w3.org/TR/1999/REC-xpath-19991116>, which is XPath 1.0. Resource constrained implementations may find it difficult to export full XPath processing and yet still wish to use a subset of XPath syntax. This does not require the addition of a new dialect, as long as the expression specified in the filter is a true XPath expression. The use of the filter dialect URI does not imply that the service supports the entire specification for that dialect, only that the expression conforms to the rules of that dialect. Most services will use XPath only for filtering, but will not support the composition of new XML or removing portions of XML which would result in the XML fragment violating the schema of the event.

- R7.2.2-1:** The wse:Filter element MUST contain either simple text or a single XML element of a single or complex type. A service SHOULD reject any Filter with mixed content or multiple peer XML elements using a wse:EventSourceUnableToProcess fault.
- R7.2.2-2:** A conformant service is NOT REQUIRED to support the entire syntax and processing power of the specified Filter Dialect. The only requirement is that the specified Filter is syntactically correct within the definition of the Dialect. Subsets are therefore legal. If the specified filter exceeds the capability of the service, a wse:EventSourceUnableToProcess fault should be returned with text explaining why the filter was problematic.
- R7.2.2-3:** If a service requires complex initialization parameters in addition to the filter, these SHOULD be part of the wse:Filter block, as they logically form part of the filter initialization, even if some of the parameters are not strictly used in the filtering process. A unique Dialect URI MUST be devised for the event source in this case and the schema and usage published.
- R7.2.2-4:** If the service supports composition of new XML or filtering to the point where the resultant event would not conform to the original schema for that event, the event delivery SHOULD be wrapped in the same way as content for fragment-level transfer (4.9 of this specification).

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

R7.2.2-5: If the service requires specific initialization XML in addition to the filter in order to formulate a subscription, this initialization XML MUST form part of the filter body and be documented as part of the filter dialect.

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

7.2.3 Connection Retries

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

R7.2.3-1: A service MAY observe any connection retry policy, or allow the subscriber to define it by including the following `wsman:ConnectionRetry` instruction in a subscription. The service is NOT REQUIRED to accept `wsman:ConnectionRetry` and may return a `wsman:UnsupportedFeature` fault with a detail code of `wsman:faultDetail/DeliveryRetries`. This only applies to failures to *connect* and does not include replay of actual SOAP deliveries:

```
(1) <wse:Subscribe>
(2)   <wse:Delivery>
(3)     <wse:NotifyTo> ... </wse:NotifyTo>
(4)     <wsman:ConnectionRetry Total="count"> xs:duration </wsman:ConnectionRetry>
(5)   </wse:Delivery>
(6) </wse:Subscribe>
```

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

`wsman:ConnectionRetry`

An `xs:interval` on how long to wait between retries while trying to connect.

`wsman:ConnectionRetry/@Total`

How many retries to attempt, observing the above interval between the attempts.

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

Note that the retry mechanism only applies to attempts to connect. Failures to deliver on an established connection should result in the termination of the connection according to the rules of the transport in use, and termination of the subscription. Other Web services mechanisms can be used to synthesize reliable delivery or safe replay of the actual deliveries.

7.2.4 `wse:SubscribeResponse`

The service may of course return any service-specific reference properties or reference

parameters in the wse:SubscriptionManager, and these must be used by the subscriber (client) later when issuing an Unsubscribe and Renew messages.

R7.2.4-1: In the wse:SubscribeResponse, the service MAY specify any EPR for the wse:SubscriptionManager. However, the address is typically expected to be the same as the wsa:To address of the original wse:Subscribe request, or the client may not be able to use the wse:SubscriptionManager content or access the specified address. It is RECOMMENDED that the wsa:Address be the same as the address used in the wsa:Subscribe message.

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

7.2.5 Heartbeats

A typical problem with event subscriptions is a situation in which no event traffic occurs. It is difficult for clients to know whether no events matching the subscription have occurred or whether the subscription has simply failed and the client was not able to receive any notification.

Because of this, WS-Management defines a "heartbeat" pseudo-event which can be periodically sent for any subscription. This event is sent if no regular events occur, and the client then knows that the subscription is still active. Should the heartbeat event not arrive at the client, then the client knows that connectivity is bad or that the subscription has expired and it may take corrective action.

The heartbeat event is sent *in place of* the events that would have occurred and is *never* intermixed with "real" events. In all modes, including batched, it occurs alone.

To request heartbeat events as part of a subscription, the wse:Subscribe request has an additional field in the wse:Delivery section:

```
(1) <wse:Delivery>
(2)   ...
(3)   <wsman:Heartbeats> xs:duration </wsman:Heartbeats>
(4)   ...
(5) </wse:Delivery>
```

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

wsman:Heartbeats

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

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

Heartbeats apply to "pull" mode deliveries as well, in that they are a hint to the publisher about how often to expect a wsen:Pull request. The service may refuse to deliver events if the client does not regularly call back at the heartbeat interval. If no events are available at

the heartbeat interval, the service simply includes a heartbeat event as the result of the `wsen:Pull`.

R7.2.5-2: While a subscription with heartbeats is active, the service **MUST** ensure that either real events or heartbeats are sent out within the specified `wsm:Heartbeat` interval. The service **MAY** send out heartbeats at this interval in addition to the events due to colliding time windows and race conditions, as long as the heartbeat events are sent separately (not batched with other events). The goal is to ensure that some kind of event traffic always occurs within the heartbeat interval.

R7.2.5-3: A conformant service **MAY** send out heartbeats at earlier intervals than specified in the subscription. However, the events should **NOT** be intermixed with other events when batching delivery modes are used. Typically, heartbeats are sent out *only when no real events occur*. It is **NOT** a **REQUIREMENT** for a service to fail to produce heartbeats at the specified interval if real events have been delivered.

R7.2.5-4: A conformant service **MUST NOT** send out heartbeats asynchronously to any event deliveries already in progress. They must be delivered in sequence like any other events, although they are delivered alone as single events or the only event in a batch.

In practice, heartbeat events are based on a countdown timer. If no events occur, the heartbeat is sent out alone. However, every time a real event is delivered, the heartbeat countdown timer is reset. If a steady stream of events occurs, heartbeats may never be delivered.

Heartbeats need to be acknowledged like any other event.

The client will assume that the subscription is no longer active if no heartbeats are received within the specified interval, so the service should proceed to cancel the subscription and send any requested `SubscriptionEnd` messages, as the client will likely resubscribe shortly. Used in combination with bookmarks, heartbeats can be used to achieve highly reliable delivery with known latency behavior.

The heartbeat event itself is simply an event message with no body and is identified by its `wsa:Action` URI:

```
(1) s:Envelope ...>
(2)   <s:Header>
(3)     <wsa:To> .... </wsa:To>
(4)     <wsa:Action s:mustUnderstand="true">
(5)       http://schemas.xmlsoap.org/ws/2005/02/management/Heartbeat
(6)     </wsa:Action>
(7)     ...
(8)
```

7.2.6 Bookmarks

Reliable delivery of events is difficult to achieve. It is highly desirable that management subscribers have a mechanism where they can be certain that all events from a source have been received. When subscriptions expire or when deliveries fail, windows of time can occur in which the client cannot be certain that critical events have not occurred. Rather than using a highly complex transacted delivery model, WS-Management defines a simple

mechanism for ensuring that either all events are either delivered or it can be detected that events have been dropped.

For this to succeed, event sources must be backed by logs, whether short-term or long-term. The client subscribes as normal for WS-Eventing, specifying that Bookmarks should be used. The service then sends a new bookmark along with each event delivery, which the client is responsible for persisting. This bookmark is essentially a context or a pointer to the location in the logical event stream that matches the subscription filter. As each new delivery occurs, the client updates the bookmark in its own space. If the subscription expires or is terminated unexpectedly, the client may subscribe again, using the last known bookmark. In essence, the subscription filter identifies the set of events that are desired, and the bookmark tells the service where to start in the log. The client may then pick up where it left off.

Note that this mechanism is immune to transaction problems, because the client can simply start from any of several recent bookmarks. The only requirement is that the service have some type of persistent log to which to apply the bookmark. If the submitted bookmark is too old (temporally or positionally within the log), the service can fault the request, and the client at least reliably knows that events have been dropped.

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

To request bookmark services, the client includes the following element in the `wse:Subscribe` request in the `Delivery` element:

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

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

`wsman:SendBookmarks`

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

The bookmark is a token which represents an abstract pointer in the event stream, but it is not material whether it points to the last delivered event or the last event plus one (the upcoming event), since the token is supplied to the same implementation during a subsequent `wse:Subscribe` operation. The service may thus attach any service-specific meaning and structure to it with no change to the client.

If bookmarks are requested, each event delivery contains a new bookmark value as a SOAP

header, and the format of the bookmark is entirely determined by the service and should be treated as an opaque value:

```
(9)  <s:Envelope
(10)      xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(11)      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing
(12)      xmlns:wsmn="http://schemas.xmlsoap.org/ws/2005/02/management">
(13)  <s:Header>
(14)      <wsa:To s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
(15)      ...
(16)      <wsmn:Bookmark> xs:any </wsmn:Bookmark>
(17)      ...
(18)  </s:Header>
(19)  <s:Body>
(20)      ...event content...
(21)  </s:Body>
(22) </s:Envelope>
```

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

wsman:Bookmark

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

R7.2.6-2: If bookmarks are supported, they MUST consist of XML content defined by the specific service, but may not be simple text. [That is, wsman:Bookmark does not support mixed content].

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

Once the subscription has terminated, for whatever reason, a subsequent wse:Subscribe on the part of the client may include the bookmark in the subscribe request. The service then 'knows' where to start. The last-known bookmark received by the client is added to the wse:Subscribe message as a new block, positioned after the wse:Filter element:

```
(23)  ...
(24)  <s:Body>
(25)      <wse:Subscribe>
(26)          <wse:Delivery> ... </wse:Delivery>
(27)          <wse:Expires> ... </wse:Expires>
(28)          <wse:Filter> ... </wse:Filter>
(29)          <wsman:Bookmark>
(30)              ...last known bookmark from a previous delivery...
(31)          </wsman:Bookmark>
(32)          <wsman:SendBookmarks/>
(33)      </wse:Subscribe>
(34)  </s:Body>
```

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

wsman:Bookmark

Arbitrary XML content previously supplied by the service as a wsman:Bookmark during event deliveries from a previous subscription.

wsman:SendBookmarks

An instruction to continue delivering updated bookmarks with each event delivery.

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

R7.2.6-5: The service **MAY** support a short queue of previous bookmarks, allowing the subscriber to start using any of several previous bookmarks. If bookmarks are supported, the service is **REQUIRED** only to support the most recent bookmark for which delivery had apparently succeeded.

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

(a) wsman:faultDetail/Expired : The bookmark has expired (the source is not able to back up and replay from that point).

(b) wsman:faultDetail/InvalidFormat : The format is unknown

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

There is a predefined, reserved bookmark value for indicating that the subscription should start at the earliest possible point in the event stream backed by the publisher:

<http://schemas.xmlsoap.org/ws/2005/02/management/bookmark/earliest>. If a subscription is received with this bookmark, the event source should replay all possible events which match the filter and of course any events which subsequently occur for that event source. Absence of any bookmark means "begin at the next available event".

R7.2.6-7: A conformant service **MAY** support the reserved bookmark **<http://schemas.xmlsoap.org/ws/2005/02/management/bookmark/earliest>** and not support any other type of bookmark.

7.2.7 Delivery Modes

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

In essence, delivery consists of

- a) A delivery mode (how events are packaged)
- b) An address (the transport and network location)
- c) An authentication profile to use when connecting or delivering the events (security)

The standard security profiles are discussed in Section 9 and may be required for subscriptions if the service needs hints or other indications as to which security model to

use at event-time.

R7.2.7-1: For any given transport, a conformant service SHOULD support at least one of the delivery modes listed below in order to interoperate with standard clients:

- a) <http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push>
- b) <http://schemas.xmlsoap.org/ws/2005/02/management/PushWithAck>
- c) <http://schemas.xmlsoap.org/ws/2005/02/management/Events>
- d) <http://schemas.xmlsoap.org/ws/2005/02/management/Pull>

Note that the delivery mode does *not imply* any specific transport.

Modes describe SOAP message behavior and are unrelated to the transport that is in use. Note that a delivery mode implies a specific SOAP message format, so a messages which deviates from that format will require a new delivery mode.

7.2.8 Event Action URI

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

```
http://schemas.xmlsoap.org/ws/2005/02/management/Event
```

This URI may have to be used in cases where event types are inferred in real-time from other sources and were not published as Web service events, and thus do not have a designated Action URI.

7.2.9 Delivery Sequencing and Acknowledgement

For some event types, ordered and acknowledged delivery is important, and with some types of events the order of arrival is not significant. WS-Management defines four standard delivery modes:

- (a) **<http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push>**
With this mode, there is only one event per SOAP message, and there is no acknowledgment or SOAP response with this delivery mode. The service MAY deliver events for the subscription asynchronously without regarding to any events already in transit. This is primarily useful when the order of events does not matter, such as with events containing running totals in which each new event can replace the previous one completely and the timestamp is sufficient for identifying the most recent event.
- (b) **<http://schemas.xmlsoap.org/ws/2005/02/management/PushWithAck>**
With this mode, there is only one event per SOAP message, but each event is acknowledged before another may be sent. The service MUST queue all undelivered

events for the subscription only deliver each new event after the previous one has been acknowledged.

(c) <http://schemas.xmlsoap.org/ws/2005/02/management/Events>

With this mode, there can be many events per SOAP message, but each batch is acknowledged before another may be sent. The service MUST queue all events for the subscription and deliver them in that order, maintaining the order in the batches.

(d) <http://schemas.xmlsoap.org/ws/2005/02/management/Pull>

With this mode, there can be many events per SOAP message, but each batch is acknowledged. Since the receiver uses wsen:Pull to synchronously retrieve the events, acknowledgment is implicit. The order of delivery must be maintained.

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

The acknowledgement model is discussed in 7.7.

7.2.10 Push Mode

The standard mode from WS-Eventing is

<http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push> in which each delivery consists of a single event. There is no acknowledgement, so it is not possible to fault the delivery in order to cancel the subscription.

Therefore, subscriptions made with this delivery mode should have short durations to prevent a situation in which deliveries cannot be stopped if the wse:SubscriptionManager content from the wse:SubscribeResponse information is corrupted or lost.

To promote fast routing of events, the required wsa:Action URI in each event message should be distinct for each event type, regardless of how strongly typed the event Body is.

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

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

```
(1) <wse:Delivery>
(2)   <wsa:Address> ... </wsa:Address>
(3)   ...
(4)   <wsman:MaxEnvelopeSize Policy="enumConstant">
(5)     xs:positiveInteger
(6) </wsman:MaxEnvelopeSize>
(7)   ...
```

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

wsman:MaxEnvelopeSize

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

An OPTIONAL value with one of the following enumeration values:

- (a) **CancelSubscription**, meaning to cancel on the first oversized event.
- (b) **Skip**, meaning to silently skip oversized events.

(c) **Notify**, meaning to notify the subscriber that events were dropped as specified in 7.9.

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

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

7.2.11 PushWithAck Mode

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

The delivery mode URI is:

`http://schemas.xmlsoap.org/ws/2005/02/management/PushWithAck`

In every other respect except the delivery mode URI, this mode is identical to Push mode as described in 7.2.10.

R7.2.11-1: A service **SHOULD** support the `http://schemas.xmlsoap.org/ws/2005/02/management/PushWithAck` delivery mode. If the delivery mode is not supported, a fault of `wse:DeliveryModeRequestedUnavailable` **SHOULD** be returned.

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

7.2.12 Batched Delivery Mode

Batching of events is an effective way of minimizing event traffic from a high-volume event source without sacrificing event timeliness. WS-Management defines a custom event delivery mode that allows an event source to bundle multiple outgoing event messages into a single SOAP envelope. Delivery is always acknowledged, using the model defined in 7.7.

R7.2.12-1: A service **MAY** support the `http://schemas.xmlsoap.org/ws/2005/02/management/Events` delivery mode. If the delivery mode is not supported, a fault of `wse:DeliveryModeRequestedUnavailable` **SHOULD** be returned.

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

```
(1) <wse:Delivery Mode="http://schemas.xmlsoap.org/ws/2005/02/management/Events">
(2)   <wse:NotifyTo>
(3)     wsa:EndpointReferenceType
(4)   </wse:NotifyTo>
(5)   <wsman:MaxElements> xs:positiveInteger </wsman:MaxElements> ?
(6)   <wsman:MaxTime> xs:duration </wsman:MaxTime> ?
(7)   <wsman:MaxEnvelopeSize Policy="enumConstant"> xs:positiveInteger </wsman:MaxEnvelopeSize> ?
(8) </wse:Delivery>
```

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

wse:Delivery/@Mode

MUST be "http://schemas.xmlsoap.org/ws/2005/02/management/Events".

wse:Delivery/wse:NotifyTo

This required element MUST contain the endpoint reference to which event messages should be sent for this subscription.

wse:Delivery/wsman:MaxElements

This optional element MAY contain a positiveInteger that indicates the maximum number of event bodies to batch into a single SOAP envelope. The Resource MUST NOT deliver more than this number of items in a single delivery, although it MAY deliver fewer.

wse:Delivery/wsman:MaxEnvelopeSize

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

wsman:MaxEnvelopeSize/@Policy

An OPTIONAL attribute with one of the following enumeration values:

- (a) **CancelSubscription**, meaning to cancel on the first oversized events.
- (b) **Skip**, meaning to silently skip oversized events.
- (c) **Notify**, meaning to notify the subscriber that events were dropped as specified in 7.9.

wse:Delivery/wsman:MaxTime

This optional element MAY contain a duration that indicates the maximum amount of time the SERVICE should allow to elapse while batching EVENT bodies. That is, this time may not be exceeded between the encoding of the first event in the batch and the dispatching of the batch for delivery. Some publisher implementations may choose more complex schemes in which different events included in the subscription are delivered at different latencies or at different priorities. In such cases, a specific filter dialect should be designed for the purpose and used to describe the instructions to the publisher. In such cases, wsman:MaxTime can be omitted if it is not applicable, but if present, serves as an override on anything defined within the filter.

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

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

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

- a) `wman:Unsupported` with a fault detail code of `wsman:faultDetail/MaxElements` if `MaxElements` is not supported or is excessive.
- b) `wman:Unsupported` with a fault detail code of `wsman:faultDetail/MaxEnvelopeSize` if it is not supported or is excessive.
- c) `wman:Unsupported` with a fault detail code of `wsman:faultDetail/MaxTime` if `MaxTime` is not supported or is excessive.
- d) `wman:Unsupported` with a fault detail code of `wsman:faultDetail/MaxEnvelopePolicy` if `MaxEnvelopeSize/@Policy` is not supported.

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

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

```
(9) <s:Envelope ...>
(10)   <s:Header>
(11)     ...
(12)     <wsa:Action>
(13)       http://schemas.xmlsoap.org/ws/2005/02/management/Events
(14)     </wsa:Action>
(15)     ...
(16)   </s:Header>
(17)   <s:Body>
(18)     <wsman:Events>
(19)       <wsman:Event Action="event action URI"> +
(20)         ...event body...
(21)     </wsman:Event>
(22)   </wsman:Events>
(23) </s:Body>
(24) </s:Envelope>
```

s:Envelope/s:Header/wsa:Action

MUST be `http://schemas.xmlsoap.org/ws/2005/02/management/Events`.

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

Each of these required elements MUST contain the body of the corresponding event message, as if `wsman:Event` were the `s:Body` element.

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

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

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

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

The following example shows batching parameters supplied to a `wse:Subscribe` operation. The service is instructed to send no more than 10 items per batch, to wait no more than 20 seconds between the time the first event is encoded until the entire batch is dispatched, and to include no more than 8192 octets in the SOAP message:

```
(25) ...
(26) <wse:Delivery
(27)     Mode="http://schemas.xmlsoap.org/ws/2005/02/management/Events">
(28)   <wse:NotifyTo>
(29)     <wsa:Address>http://2.3.4.5/client</wsa:Address>
(30)   </wse:NotifyTo>
(31)   <wsman:MaxElements>10</wsman:MaxElements>
(32)   <wsman:MaxTime>PT20S</wsman:MaxTime>
(33)   <wsman:MaxEnvelopeSize>8192</wsman:MaxEnvelopeSize>
(34) </wse:Delivery>
(35)
```

The following example shows an example of batched delivery that conforms to this specification:

```
(36) <s:Envelope
(37)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(38)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(39)   xmlns:wse="http://schemas.xmlsoap.org/ws/2004/09/eventing">
(40) <s:Header>
(41)   <wsa:To s:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
(42)   <wsa:Action>
(43)     http://schemas.xmlsoap.org/ws/2005/02/management/Events
(44)   </wsa:Action>
(45)   ...
(46) </s:Header>
(47) <s:Body>
(48)   <wsman:Events>
(49)     <wsman:Event
```



```

(50)         Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(51)         <DiskChange
(52)             xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(53)             <Drive> C: </Drive>
(54)             <FreeSpace> 802012911 </FreeSpace>
(55)         </DiskChange>
(56)     </wsman:Event>
(57) <wsman:Event
(58)     Action="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(59)     <DiskChange
(60)         xmlns="http://schemas.xmlsoap.org/2005/02/diskspacechange">
(61)         <Drive> D: </Drive>
(62)         <FreeSpace> 1402012913 </FreeSpace>
(63)     </DiskChange>
(64)     </wsman:Event>
(65) </wsman:Events>
(66) </s:Body>
(67) </s:Envelope>

```

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

7.2.13 Pull Delivery Mode

In some circumstances, polling for events is an effective way of controlling data flow and balancing timeliness against processing ability. And in some cases, network restrictions prevent "push" modes from being used; the service cannot initiate a connection to the subscriber.

WS-Management defines a custom event delivery mode, "pull mode", which allows an event source to maintain a logical queue of event messages that are received by enumeration. For this delivery mode, the wse:Delivery element has the following format:

```

(1)
(2) <wse:Delivery Mode="http://schemas.xmlsoap.org/ws/2005/02/management/Pull">
(3) ...
(4) </wse:Delivery>
(5)

```

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

wse:Delivery/@Mode

MUST be "http://schemas.xmlsoap.org/ws/2005/02/management/Pull".

R7.2.13-1: A service is NOT REQUIRED to support the <http://schemas.xmlsoap.org/ws/2005/02/management/Pull> delivery mode. If requested and not supported, the service MUST return a fault of wse:DeliveryModeRequestedUnavailable.

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

R7.2.13-2: If a subscription incorrectly specifies parameters that are not compatible with "Pull Mode", then the service SHOULD issue a `wsman:UnsupportedFeature` fault with a detail code of `wsman:faultDetail/FormatMismatch`.

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

```
(6) <s:Body ...>
(7)   <wse:SubscribeResponse ...>
(8)     <wse:SubscriptionManager>
(9)       wsa:EndpointReferenceType
(10)    </wse:SubscriptionManager>
(11)    <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires>
(12)    <wsen:EnumerationContext>...</wsen:EnumerationContext>
(13)    ...
(14)  </wse:SubscribeResponse>
(15) </s:Body>
```

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

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

R7.2.13-5: If Pull mode is active, and `wsen:Pull` request returns no events (because none have occurred since the last 'pull'), the service SHOULD return a `wsman:TimedOut` fault. The `wsen:EnumerationContext` is still considered active and the subscriber may continue to issue `wsen:Pull` requests with the most recent `wsen:EnumerationContext` for which event deliveries actually occurred.

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

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

Note that in pull mode, the `wsen:Pull` request controls the batching. If no defaults are specified, the batch size is 1 and the maximum envelope size and timeouts are service-defined.

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

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

Note that the batching properties used in 'batched' mode do not apply to "pull" mode. The client controls the maximum event size using the normal mechanisms in `wsen:Pull`.

7.3 GetStatus

This message is optional for WS-Management.

R7.3-1: A conformant service is **NOT REQUIRED** to implement the `GetStatus` message or its response. It is **NOT RECOMMENDED** that services implement this for future compatibility.

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

7.4 Unsubscribe

Unsubscribe cancels a subscription.

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

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

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

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

7.5 Renew

According to WS-Eventing, the `wse:Renew` message is not optional in terms of processing,

but there is no requirement that it actually must succeed.

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

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

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

7.6 SubscriptionEnd

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

- R7.6-1:** A conformant service is **NOT REQUIRED** to implement the `SubscriptionEnd` message. If implemented, the service **MAY** fail to accept a subscription with any address differing from the `NotifyTo` address.
- R7.6-2:** A conformant service **MUST NOT** implement the `SubscriptionEnd` when event delivery is done using Pull mode as defined in 7.2.12.
- R7.6-3:** If `SubscriptionEnd` is supported, the message **MUST** contain any reference properties or parameters specified by the subscriber in the `EndTo` address in the original subscription.
- R7.6-4:** If `SubscriptionEnd` is supported, it is **RECOMMENDED** that it be sent to the subscriber prior to sending the `UnsubscribeResponse`.

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

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

7.7 Acknowledgement of Delivery

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

In other words, the transport may have accepted delivery of the events but not forwarded them to the actual subscriber process, and the service would move on to the next set of events. System failures might result in dropped events. Therefore, there needs to be a mechanism in which a message-level acknowledgement can occur. This allows acknowledgement to be pushed up to the application level, increasing the reliability of event deliveries.

The client selects acknowledged delivery by selecting a delivery mode in which each event has a response. In this specification, the two acknowledged delivery modes are:

- <http://schemas.xmlsoap.org/ws/2005/02/management/PushWithAck>
- <http://schemas.xmlsoap.org/ws/2005/02/management/Events>

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

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

```
(1) <s:Header>
(2)   <wsa:ReplyTo> where to send the acknowledgement </wsa:ReplyTo>
(3)   <wsman:AckRequested/>
```

wsa:ReplyTo

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

wsman:AckRequested

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

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

R7.7-3: If a service requests acknowledgement of receipt by using the wsman:AckRequested block, the receiver MUST acknowledge the receipt by replying with an **<http://schemas.xmlsoap.org/ws/2005/02/management/Ack>** message. If this message is not received as a reply, the service MAY terminate the subscription immediately. The acknowledgment message format is identical for all delivery modes. It contains a unique wsa:Action, and MUST contain the event the wsa:RelatesTo field set to the MessageID of the event delivery to which it applies:

```

(4)
(5)  <s:Envelope ...>
(6)  <s:Headers>
(7)    ...
(8)    <wsa:To> endpoint reference from the event delivery ReplyTo field </wsa:To>
(9)    <wsa:Action> http://schemas.xmlsoap.org/ws/2005/02/management/Ack </wsa:Action>
(10)   <wsa:RelatesTo> message ID of original event delivery </wsa:RelatesTo>
(11)   ...
(12)  </s:Headers>
(13)  <s:Body/>
(14) </s:Envelope>

```

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

s:Envelope/s:Header/wsa:Action

MUST be `http://schemas.xmlsoap.org/ws/2005/02/management/Ack`.

s:Envelope/s:Header/wsa:RelatesTo

This MUST contain the `wsa:MessageID` of the event delivery to which it refers.

s:Envelope/s:Header/wsa:To

The endpoint reference address extracted from the `ReplyTo` field in the event delivery. All reference properties and reference parameters must be extracted and added to the SOAP header as well.

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

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

If the client does not support acknowledgement, it may respond with a `wsman:UnsupportedFeature` fault with a detail code `wsman:faultDetail/Ack`.

However, this is as difficult as acknowledging the delivery, so most clients should scan for the `wsman:AckRequested` field and be prepared to acknowledge delivery or fault it.

Note that with simple "Push" mode, there is no way for the client to fault a delivery or acknowledge it.

7.8 Refusal of Delivery

With all acknowledged delivery modes as described in 7.7, a subscriber may refuse to take delivery of events, either for security reasons or a policy change. It then responds with a fault rather than an acknowledgement.

In this case, the service must be prepared to end the subscription even though a `wse:Unsubscribe` message is not issued by the subscriber.

R7.8-1: During event delivery, if the receiver faults the delivery with a `wsman:DeliveryRefused` fault, the service MUST immediately cancel the subscription and MAY also issue a `wse:SubscriptionEnd` message to the `wse:EndTo` endpoint in the original subscription if supported.

Thus, the receiver MAY issue the fault in as a technique for canceling the subscription when it does not have the wse:SubscriptionManager information.

7.9 Dropped Events

Events which cannot be delivered should not be silently dropped from the event stream, or the subscriber gets a false picture of the event history. WS-Management defines three behaviors for events which cannot be delivered via "Push" modes or which are too large to fit within the delivery constraints requested by the subscriber:

- a) Terminate the subscription
- b) Silently skip such events
- c) Send a special event in place of the dropped event(s)

These options are discussed in 7.2.10 and 7.2.11

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

R7.9-1: If a service drops events, it SHOULD issue an **<http://schemas.xmlsoap.org/ws/2005/02/management/DroppedEvents>** event which indicates this to the client. Any reference properties or reference parameters which were specified in the wsa:NotifyTo address in the subscription MUST also be copied into this message. This is a normal event and implicitly considered part of any subscription.

R7.9-2: If an **<http://schemas.xmlsoap.org/ws/2005/02/management/DroppedEvents>** event is issued, it MUST take the ordinal position of the original dropped event in the delivery stream. If "batched" delivery mode is in use, the event takes the place in the batch of the event it represents.

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

```
(1) <s:Envelope ...>
(2)   <s:Header>
(3)     ...subscriber endpoint-reference...
(4)
(5)   <wsa:Action>
(6)     http://schemas.xmlsoap.org/ws/2005/02/management/DroppedEvents
(7)   </wsa:Action>
(8) </s:Header>
(9) <s:Body>
```

```

(10)     <wsman:DroppedEvents Action="wsa:Action URI of dropped event">
(11)         xs:positiveInteger
(12)     </wsman:DroppedEvents>
(13)     ...
(14) </wsman:DroppedEvent>
(15) </s:Body>
(16) </s:Envelope>

```

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

s:Envelope/s:Header/wsa: Action

MUST be <http://schemas.xmlsoap.org/ws/2005/02/management/DroppedEvents>.

s:Body/wsman: DroppedEvents/@Action

The Action URI of the event which was dropped.

s:Body/wsman: DroppedEvents

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

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

Here is an example of how a dropped event would appear in the middle of a batched event delivery:

```

(17)
(18) <wsman:Events>
(19)   <wsman:Event Action="https://foo.com/someEvent">
(20)     ...event body
(21)   </wsman:Event>
(22)   <wsman:Event Action="http://schemas.xmlsoap.org/ws/2005/02/management/DroppedEvents">
(23)     <wsman:DroppedEvents Action="https://foo.com/someEvent"> 1 </wsman:DroppedEvents>
(24)   </wsman:Event>
(25)   <wsman:Event Action="https://foo.com/someEvent">
(26)     ...event body
(27)   </wsman:Event>

```

Note that the dropped event is an event in itself.

R7.9-3: If a service cannot deliver an event and does not support the <http://schemas.xmlsoap.org/ws/2005/02/management/DroppedEvents> event, it SHOULD terminate the subscription rather than silently skipping events.

Since this cannot be enforced and some dropped events are irrelevant when replaced by a subsequent event (running totals, for example), it is not a firm requirement that dropped events are signaled or that they result in a termination of the subscription.

8.0 Standard System Resources

A WS-Management service must support a standard system endpoint for the purposes of identification and versioning of the implementation.

8.1 wsman:system/2005/02/this

This resource identifies the protocol implementation.

R8.1.1-1 A conformant service **MUST** support a wxf:Get against a ResourceURI of **wsman:system/2005/02/this**. No Selectors are required. If the service does not have the metadata, but knows where it can be found, it **SHOULD** return a wsman:MetadataRedirect fault, which indicates the correct network HTTP address or SOAP EPR. If the service does not have the metadata, then a normal wsa:DestinationUnreachable fault **SHOULD BE** returned.

As an example, the wxf:Get would appear as follows:

```
(1) <s:Envelope
(2)   xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
(4)   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/02/management">
(5) <s:Header>
(6)   <wsa:Action>
(7)     http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
(8)   </wsa:Action>
(9)   <wsman:To> http://1.2.3.4/wsman?ResourceURI=(wsman:system/2005/02/this)
(10)  ...
```

The value returned via the wxf:GetResponse documents the current WS-Management implementation. This is the format:

```
(11) <s:Body>
(12)   <This xmlns="http://schemas.xmlsoap.org/ws/2005/02/management">
(13)     <Vendor> Vendor identification </Vendor>
(14)     <Version> Vendor implementation version </Version>
(15)   </This>
(16) </s:Body>
```

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

s:Body/This

The resource describing the WS-Management implementation

s:Body/This/Vendor

A string identifying the vendor. This may be a URL or a text representation.

s:Body/This/Version

A string which indicates the implementation version. Whenever the implementation for that vendor is updated in any way via a patch or major upgrade, this string should be altered to reflect it. It does *not* refer to the protocol version, which is already implied by the resource URI.

Note that there is a strong binding between the **wsman:system/2005/02/this** URI and the above schema namespace. If future metadata documents are required, they are simply assigned a different URI.

A full example of the document follows:

```
(17)   <s:Body>
(18)       <This xmlns="http://schemas.xmlsoap.org/ws/2005/02/management">
(19)           <Vendor> Acme Inc. http://www9.acme.com/default </Vendor>
(20)           <Version> 2.02.4a </Version>
(21)       </This>
(22)   </s:Body>
```

The above example indicates the vendor and implementation version.

8.2 Other Discovery

WS-Management allows implementations to support a wide variety of options and parameterization. The discovery of the service capabilities and any negotiation between the client and the service about what is legal are not a matter of protocol operations.

Rather, the service should export metadata which describes the valid operations and ranges of values. The format of this metadata is beyond the scope of this particular specification.

9.0 Security

9.1 Introduction

In general, management operations and responses should be protected against attacks such as snooping, interception, replay, and modification during transmission. Generally, it is also necessary to authenticate the user who has sent a request in order to apply access control rules to determine whether or not to process a request.

This specification establishes the minimum interoperation standards and predefined profiles using transport-level security.

This approach provides the best balance of simplicity of implementation (HTTP and HTTPS stacks are readily available, even for hardware) and the security mechanisms sit in front of any SOAP message processing, limiting the attack surface.

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

Implementations which expect to interoperate should adopt one or more of the transport and security models defined in this chapter and are free to define any additional profiles under different URI-based designators.

9.2 Security Profiles

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

describes the message exchange involved.

Discovery of which profiles are supported by the service should be done through metadata retrieval and is beyond the scope of this particular specification.

For all of the predefined profiles, the transport is responsible for all message integrity, protection, authentication and security.

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

9.3 Interoperation Conformance

This specification does not mandate that conformant services must provide HTTP or HTTPS based access. However, it does mandate that if HTTP or HTTPS is used, at least one of the predefined profiles for that transport must be supported so that clients can reliably access the service.

R9.3-1: A conformant service which supports HTTP MUST support one of the predefined HTTP-based profiles.

R9.3-2: A conformant service which supports HTTPS MUST support one of the predefined HTTPS-based profiles.

R9.3-3: A conformant service MUST NOT expose WS-Management over a completely unauthenticated HTTP channel.

There is no requirement that the service only export a single HTTP or HTTPS address. The service may export multiple addresses, each of which supports a specific security profile or multiple profiles.

If clients support all of the predefined profiles, they are assured of access to a WS-Management implementation which supports HTTP and/or HTTPS.

9.4 `wsman:secprofile/http/standard`

Typically, HTTP-based implementations will support the normal HTTP model of access in which the original request fails and an HTTP 401 error is returned with a list of supported authentication modes. While Basic Authentication is not recommended, it may be suitable for less secure environments, especially when the service is only exporting read-only information which is not intrinsically a security risk.

In practice, this profile means nothing more than ensuring that the HTTP Authorization header is in the POST request.

The typical sequence is:

	Client		Service
1	Client connects with no auth header	➔	Service sees no header
2		➔	Service sends 401, listing available auth modes (Digest, Basic, etc.)
3	Client selects appropriate auth mode & HTTP authorization header	➔	Service authenticates the client

This is the normal behavior for HTTP. If the client connects with an authorization header initially and it is valid, then of course the request immediately succeeds.

This does not preclude Basic Authentication or other types of authentication and does not mandate Basic or Digest.

9.5 wsman:secprofile/https/standard

This mode is the same as wsman:secprofile/http/standard, except that it occurs over HTTPS. The server-side cert is used to protect the network connection (via TLS 1.0), but the client does not supply a certificate and is expecting to use an authorization header or be challenged for one:

	Client		Service
1	Client connects with no auth header	➔	Server-side cert is used to encrypt connection and service sees no auth header
2		➔	Service sends 401, listing available auth modes (Digest, Basic, etc.)
3	Client selects appropriate auth mode & HTTP authorization header	➔	Service authenticates the client

This is the normal behavior for HTTP, but using HTTPS to protect the network traffic. If the client connects with an authorization header initially and it is valid, then of course the request immediately succeeds.

Note that supporting Digest authentication in this profile does not present any greater protection than support Basic authentication from the point of view of securing network traffic, since all traffic is encrypted anyway.

To maximize the probability of interoperation, Basic authentication is recommended for this profile.

9.6 wsman:secprofile/https/mutual

In this security mode, the client supplies an X.509 certificate and it is used to authenticate the client. No HTTP(S) authorization header is required in the HTTP POST request.

However, as a hint to the service, the following HTTP(S) authorization header may be present:

```
Authorization: wsman:auth/https/tls/1.0
```

However, since the service can be configured to always look for the certificate, this is not required.

The sequence is simple:

	Client		Service
1	Client connects with no auth header but supplies an X.509 cert	➔	Service ignores the authorization header and retrieves the client-side cert used in the TLS 1.0 handshake
2		←	Service accept access, or denies access with 403.7 or 403.16, etc.

9.7 wsman:secprofile/https/dual

In this profile, when the client connects the service looks for an HTTP(S) authorization header initially. If one is found, it is used:

	Client		Service
1	Client connects with auth header, but no cert	➔	Service looks for authorization header and finds it, and uses that mechanism
2		←	If the supplied authorization header is not of the correct type, 401 is returned.

Alternately, the client may connect with an X.509 certificate, but no HTTP(S) authorization header:

	Client		Service
1	Client connects with cert, but no authorization header	➔	Service looks for authorization header and does not find it, so it assumes a certificate is in use.
2		←	If certificate is missing, 403.7 is returned. If it is not valid, a specific

			error such as 403.16 is returned.
--	--	--	-----------------------------------

In this way, the same HTTPS address can be used to support both certificates or HTTP(S) authorization headers.

The main requirement here is that if the client wishes to use an authorization header and not supply a client-side certificate, it **MUST** initially supply an HTTP(S) authorization header at the outset, because if no header is present, the service will assume a certificate is in use and never issue the 401 response, allowing the client to select a different authentication mechanism. Instead, it will return a 403.7 or 403.16. The client will have to reconnect with the authorization header in the initial request.

Also, note that if the client supplies a certificate and uses an authorization header, the certificate is ignored (unless it is the `wsman:auth/https/tls/1.0`). To force recognition of the certificate using an authorization header, see 9.6.

9.8 `wsman:secprofile/https/selected`

In this profile, the client connects with HTTPS and the service is capable of accepting more than one authentication model over the same connection. The client **MUST** supply an HTTP(S) authorization header at some point, whether or not client certificates are used. This has the advantage that the client does not need to supply an authorization header initially and can select from among the available mechanisms after the first request.

The additional requirement is that the client must be aware of a new HTTP(S) authorization header and be able to use it.

There is a strict sequence of events:

	Client		Service
1	Client connects with or without cert, but no auth header	➔	Service cannot safely check for cert, because if it is not there, it cannot return a 401 allowing other authorization options to the client
2		←	Service sends 401, listing available auth modes, including the special authorization header for client certs header (<code>wsman:auth/https/tls/1.0</code>).
3	If client wants to use the cert it originally sent, it uses the HTTP authorization header <code>wsman:auth/https/tls/1.0</code>	➔	Service sees the authorization header and now knows to check for the certificate, returning a 403.7 or 403.16 as appropriate.
OR			
3	If the client wants to use a different authentication model, it selects among the	➔	Service uses whatever authorization header is selected by the client

	possibilities in the 401 response (step 2)		
--	--	--	--

The important aspect of this sequence is step 1. The goal is to allow the connection to allow the client to use one of several authentication models without knowing which one the client will choose right from the start.

The client may not have supplied the certificate in the initial connection, counting on being challenged with a 401 so that it can select Basic Authentication or some other authentication. In this case, the server-side cert is the only one used, and it simply protects the connection by encrypting the network traffic.

But the client may have used a certificate when connecting and wanted to use the certificate. The service cannot simply query for it, because if it does and the certificate is in fact not present, then there is no possibility of returning an HTTP 401 error and letting the client select Basic Authentication or some other model.

Therefore, if no authentication header is present initially, the service will always return a 401, and make the client choose an HTTP(S) authorization header. This happens whether or not the client initially supplied a certificate. In addition to the standard authorization headers (Basic, Digest), WS-Management defines the following additional authorization header:

```
Authorization: wsman:auth/https/tls/1.0
```

This merely means that the certificate attached to the HTTPS TLS 1.0 connection is to be used in authentication.

If the HTTP(S) header is already present in the initial request, then the negotiation sequence beginning with a 401 challenge is avoided.

If the client specifies an authorization header right from the start, the service uses the one that is specified.

9.9 wsman:secprofile/https/mutual/rechallenge

This mode is the same as wsman:secprofile/https/mutual except that an authorization header is used *in addition* to mutual authentication using the TLS 1.0 handshake.

This is used in cases where the mutual authentication is used at one level to ensure the two endpoints have authorization to communicate (such as machine-level certificates), but the resources may be secured at different levels, so an additional level of authorization has to occur using the HTTP authorization header:

	Client		Service
1	Client connects with cert and special auth header	→	Service queries for client cert and authenticates. If cert is missing or invalid, the sequence stops here with 403.7 or 403.16 return codes.

2		←	After authenticating the certificate, the service sends 401, listing available auth modes (Digest, Basic, etc.)
3	Client selects new auth mode to use	→	Service authenticates the client again

In the initial request, the HTTP authorization header MUST be

```
Authorization: wsman:auth/https/tls/1.0/rechallenge
```

This indicates to the service that this special mode is in use and it can query for the client certificate. The service then returns a 401, listing the authorization headers to use in subsequent requests.

If the initial certificate is not valid or missing, then 403.7 or 403.16 is returned as appropriate.

9.10 Subscriptions

When specifying the wse:NotifyTo address in subscriptions, it is often important to give hints to the service as to which authentication model to use when delivering the event.

If no hints are present, then it is assumed that the service can simply infer from the wsa:To address what needs to be done. However, if the service can support multiple modes and has a certificate or password store, it may not know which authentication model to choose or which credentials to use without being told in the subscription.

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

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

```
(1) <s:Body>
(2)   <wse:Subscribe>
(3)     <wse:Delivery>
(4)       <wse:NotifyTo> address </wse:NotifyTo>
(5)       <wsman:Auth Profile="">
(6)         ...
(7)       </wsman:Auth>
(8)
```

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

wsman:Auth

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

wsman:Auth/@Profile

A URI which indicates which security profile to use when authenticating and how to interpret the content of the wsman:Auth block.

If the wsman:Auth block is not present, then the service must infer what to do by using the wse:NotifyTo address using any preconfigured policy or settings it has available.

The available predefined profiles for event delivery are discussed in the following sections.

9.11 Standard profiles

The wsman:Auth block may have no content and the Profile may contain any of the predefined profiles already defined. In this case, the service must know what credentials to use by its own internal configuration.

For example, if the service knows which certificate to use when delivering events, the subscriber can request standard mutual authentication:

```
(1) <s:Body>
(2)   <wse:Subscribe>
(3)     <wse:Delivery>
(4)       <wse:NotifyTo> HTTPS address </wse:NotifyTo>
(5)     <wsman:Auth Profile="wsman:secprofile/https/mutual"/>
```

Similarly, if the service knows how to retrieve a proper username and password for event delivery, simple HTTP standard authorization can be used:

```
(6) <s:Body>
(7)   <wse:Subscribe>
(8)     <wse:Delivery>
(9)       <wse:NotifyTo> HTTP address </wse:NotifyTo>
(10)    <wsman:Auth Profile="wsman:secprofile/http/standard"/>
```

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

9.12 wsman:secprofile/https/standard/userRef

In this profile, the service connects using HTTPS to deliver the event. The receiver's certificate is used to encrypt the connection, but not to authenticate the sender (the service in this case). Then, the service (acting as the event publisher) uses a standard HTTP Authorization header to authenticate. In order to know which user name, password, etc. to use, the subscription contains a reference to it, in which the wsman:Auth block contains a username with no password:

```
(1)
(2) <s:Body>
```

```

(3) <wse:Subscribe>
(4)   <wse:Delivery>
(5)     <wse:NotifyTo> address </wse:NotifyTo>
(6)     <wsman:Auth Profile="wsman:secprofile/https/standard/userRef">
(7)       username="username_literal" realm="realm"
(8)     </wsman:Auth>
(9)     ...

```

The implication is that at event-time, the service will connect with HTTPS and use `wsman:secprofile/https/standard` to connect. The Authorization header may already be present, in which case the event delivery will succeed, or it may be absent and the receiving end will challenge with a 401, as described under `wsman:secprofile/https/standard`.

The service must have access to a password store and look up the password that goes with the username that was specified and carry out the authentication sequence after properly hashing the password (Basic) or processing the nonce (Digest), etc.

The `wsman:Auth` content block may contain additional information, depending on the authentication model in use:

```

(10) <wsman:Auth Profile="wsman:secprofile/https/standard/userRef">
(11)   username="user" realm="realm" ...and any optional components...
(12) </wsman:Auth>

```

While it is possible to also supply the password in the subscription so that no store is required on the service side, this is a weak security practice, even though the subscription was delivered over an encrypted connection, since the subscriber is distributing a password to an unknown entity which might not treat them in a secure fashion.

9.13 `wsman:secprofile/https/mutual/certhash`

In this profile, the service (acting as publisher) connects via HTTPS to the receiver when delivering events, but the certificate to use for authenticating the sender to the receiver is preselected by the subscriber by referencing its hash value or "thumbprint":

```

(1) <s:Body>
(2) <wse:Subscribe>
(3)   <wse:Delivery>
(4)     <wse:NotifyTo> address </wse:NotifyTo>
(5)     <wsman:Auth Profile="wsman:secprofile/https/mutual/certhash">
(6)       ef 8f 11 63 97 b8 de 09 a5 d4 73 ed f2 92 35 8e 91 b4 6f 91
(7)     </wsman:Auth>
(8)     ...

```

The format of the block must be a series of hexadecimal digits (upper or lower case), with each octet (two characters) optionally separated by spaces.

The service must be able to locate and use the specified certificate when connecting for event delivery.

The service uses `wsman:secprofile/https/mutual` when actually delivering the events.

9.14 Correlation of Events with Subscription

In many cases, the subscriber will want to ensure that the event delivery corresponds to a valid subscription that was issued by an authorized party. In this case, it is recommended that reference parameters be introduced into the `wse:NotifyTo` definition.

For example, at subscription-time, a `uuid` could be supplied as a correlation token:

```
(1) <s:Body>
(2) <wse:Subscribe>
(3)   <wse:Delivery>
(4)   <wse:NotifyTo>
(5)     <wsa:Address> address </wsa:Address>
(6)     <wsa:ReferenceParameters>
(7)       <MyNamespace:uuid> uuid:b0f685ec-e5c9-41b5-b91c-7f580419093e </MyNamespace:uuid>
(8)     </wsa:ReferenceParameters>
(9)   </wse:NotifyTo>
(10) ...
```

This definition requires that the service include the `MyNamespace:uuid` value as a SOAP header with each delivery (see 2.1). The service can use this to correlate the event with any subscription that it issued and to validate its origin.

This is not a transport-level or SOAP-level authentication mechanism per se, but it does help to maintain and synchronize valid lists of subscriptions and determine if the event delivery is authorized or not, even though the connection itself may have been authenticated.

This mechanism still may require the presence of the `wsman:Auth` block to specify which security mechanism to use to actually authenticate the connection at event-time.

It is important that each new subscription receive at least one unique reference parameter which is never reused, such as the illustrate `uuid`, in order for this mechanism to be of value.

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

9.15 Transport-Level Authentication Failure

Since transports typically go through their own authentication mechanisms prior to any SOAP traffic occurring, the first attempt to connection may result in a transport-level authentication failure. In such cases, SOAP faults will not occur, and the means of communicating the denial to the client is implementation- and transport-specific.

10.0 Transports and Message Encoding

10.1 Introduction

While WS-Management is a SOAP protocol and not tied to a specific network transport,

interoperation requires the some common standards be established. This specification centers on establishing common usage over HTTP 1.1 and HTTPS.

For identification and referencing, each transport is identified by a URI, and each authentication mechanism defined in this specification is also identified by a URI.

As new transports are standardized, they should also acquire a URI for referencing purposes, and any new authentication mechanisms that they expose should also be assigned URIs for publication and identification purposes in XML documents.

An example of the usage of such URIs is shown in 8.1.1 in the standard system resource **wsman:system/2005/02/this**. However, they occur in other contexts as well.

For this specification, the standard transports are HTTP 1.1 and HTTPS (using TLS 1.), designated as follows:

- <http://www.ietf.org/rfc/rfc2616.txt> (HTTP 1.1)
- <http://www.ietf.org/rfc/rfc2818.txt> (for HTTPS)

The SOAP and HTTP encoding models specified in the following base specifications are used for WS-Management encoding over HTTP and HTTPS:

1. **SOAP Version 1.2 Part 2: Adjuncts**, SOAP HTTP binding described in section 7 of <http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapinhttp>
2. **WS-I Basic Profile Version 1.1**
<http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html>

10.2 HTTP(S) Encoding

R10.2-1: A service MUST support **Transfer-Encoding : chunked**.

This requires the service to be able to receive incoming SOAP messages in several parts or to be able to deliver them in several parts when they are very large or the size is unknown. The limits are service-specific.

R10.2-2 A service MUST at least support the SOAP HTTP Binding.

R10.2-3: A service MUST at least implement the Responding SOAP Node of the SOAP **Request-Response** Message Exchange Pattern (<http://www.w3.org/2003/05/soap/mep/request-response/>).

R10.2-4 A service MAY choose not to implement the Responding SOAP Node of the SOAP **Response** Message Exchange Pattern (<http://www.w3.org/2003/05/soap/mep/soap-response/>).

R10.2-5: A service MAY choose not to support the SOAP Web Method Feature.

R10.2-6: A service MUST at least implement the Responding SOAP Node of an HTTP one-way Message Exchange Pattern where the SOAP ENVELOPE is carried in the HTTP Request and the HTTP Response has a Status Code of 202 Accepted and an empty Entity Body (no SOAP ENVELOPE).

This is used to carry SOAP messages which require no response.

R10.2-7: A service **MUST** at least support Request Message SOAP ENVELOPEs and one-way SOAP ENVELOPEs that are delivered using HTTP POST.

R10.2-8: The HTTP(S) URL itself **MUST** contain the WS-Management ResourceURI suffixed to the HTTP address in the same manner as used in the wsa:To header as described in section 2.4.

Note this **important** requirement for HTTP(S) based access to WS-Management. While the ResourceURI is also contained in the SOAP, it simplifies the implementation if the ResourceURI can be copied to and from wsa:Address fields without modification or analysis.

R10.2-9: In cases where the service cannot respond with a SOAP message, the HTTP error code 500 (Internal Server Error) **SHOULD** be returned and the client side should close the connection.

10.3 SOAP

R10.3-1: A SERVICE **MUST** at least receive and send SOAP 1.2 [SOAP 1.2] SOAP ENVELOPEs.

R10.3-2: A SERVICE **MAY** reject a TEXT SOAP ENVELOPE with more than 32,767 octets.

R10.3-3: A SERVICE **SHOULD NOT** send a TEXT SOAP ENVELOPE with more than 32,767 octets in length unless the client has specified a wsman:MaxEnvelopeSize header overriding this limit.

Large SOAP ENVELOPEs are expected to be serialized using attachments.

R10.3-4: Any REQUEST MESSAGE **MAY** be encoded using either UNICODE 3.0 (UTF-16) or UTF-8 encoding. An Service **MUST** accept either encoding for all operations and emit RESPONSES using the same encoding as the original request.

Some SOAP-enabled systems only have UNICODE available, and some only have UTF-8. To maximize interoperability, it is trivial for a server to support both encodings, since R10.3-5 places limits on the required character set.

R10.3-5: A service **IS REQUIRED** to support characters from U+0000 to U+007F inclusive with both UTF-8 and UTF-16 encodings, and **MAY** support characters outside this range. If the message contains unsupported characters above U+007F, the service **MUST** return a wsman:EncodingLimit fault.

R10.3-6: For UTF-8 encodings, the service **MAY** fail to process any message beginning with the UTF-8 BOM (0xEF 0xBB 0xBF) at the beginning of the message and **MUST** send UTF-8 responses without the BOM. The presence of BOM in 8-bit characters encodings reduces interoperability. Where extended characters are a requirement UTF-16 should be used.

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

R10.3-7: If UTF-16 is the encoding, the SERVICE MUST support either byte order mark (BOM) U+FFFE or U+FFEF as defined in the UNICODE 3.0 specification as the first character in the message.

R10.3-8: Duplicate headers SHOULD NOT be processed. The service should issue a `wsa:InvalidMessageInformationHeaders` fault if they are detected. However, a conformant service MAY ignore any duplicate headers if it assumes the first occurrence is the valid one.

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

R10.3-9: Services MAY fault requests with leading and trailing whitespace in XML values unless they are part of the literal value. If a service detects bad whitespace usage in a request, then a `wsman:EncodingLimit` fault with a detail code of `wsman:faultDetail/Whitespace` should be returned.

Clients should not send messages with leading or trailing whitespace in the values, since the service is allowed to fault such requests. Services may be coded to be more forgiving, but the standard is to eliminate unneeded whitespace on both sides.

10.4 Lack of Response

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

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

This behavior is preferable to attempting a complex model for sending responses in a delayed fashion. Implementations should generally keep a log of all requests and their results, and allow the client to reconnect later to enumerate the operation log (using `wsen:Enumerate`) if they fail to get a response. The format and behavior of such a log is beyond the scope of this specification. Since the client must be coded to take into account a lack of response in any case, all abnormal message conditions can safely revert to this scenario.

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

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

10.5 Replay Of Messages

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

R10.5-1: A service MUST NOT resend an unacknowledged messages unless they

are part of a higher general-purpose reliable messaging or transactional protocol layer, in which case the retransmission follows the rules for that protocol.

10.6 Encoding Limits

Most of the following limits are in characters. However, the maximum overall SOAP envelope size is defined in octets. Implementations are free to exceed these limits. However, a service is considered conformant if it observes these limits. Any limit violation results in a `wsman:EncodingLimit` fault. In addition to any requirements or limits established by the WS-I Basic Profile, the service should observe the following:

- R10.6-1:** A service MAY fail to process any URI with more than 2048 characters.
- R10.6-2:** A service SHOULD NOT generate a URI with more than 2048 characters.
- R10.6-3:** A service MAY fail to process a Selector or Option Name of more than 2048 characters.
- R10.6.4:** A service MAY fail to process a Selector value or Option value of more than 4096 characters, including any embedded Selectors, and MAY fail to process a message which contains more than 8096 characters of content in the root `<Selectors>` element.
- R10.6-6:** A service MAY reject a SOAP Envelope with more than 32,767 octets. Similarly, it MAY fault any operation that would require a single reply exceeding 32,767 octets.
- R10.6-7:** A service MAY always emit *faults* that are 4096 octets or less in length, regardless of any requests by the client to limit the response size. Clients should always be prepared for this minimum in case of an error.

10.7 Binary Attachments

MTOM is used to support binary attachments to WS-Management. If a service supports attachments, the following rules apply:

- R10.7-1:** A conformant service MAY OPTIONALLY support binary attachments to any operation using the SOAP Message Transmission Optimization Mechanism (MTOM) proposal (<http://www.w3.org/TR/2004/PR-soap12-mtom-20041116>).
- R10.7-2:** If a service supports attachments, the service MUST support the Abstract Transmission Optimization Feature.
- R10.7-3:** If a service supports attachments, the service MUST support the Optimized MIME Multipart Serialization Feature.
- R10.7-4:** If a service supports attachments, the service MUST support the HTTP Transmission Optimization Feature.
- R10.7-5:** If a service cannot process a message with an attachment or unsupported encoding type, it MUST return a `wsman:EncodingLimit` fault with a detail code of `wsman:EncodingType`.

Other attachment types are not prohibited.

11.0 Faults

11.1 Introduction

Faults are returned when the SOAP message is successfully delivered by the transport and processed by the service, but the message cannot be processed properly. If the transport cannot successfully deliver the message to the SOAP processor, a transport error will occur instead.

Only SOAP 1.2 faults [or later] should be supported.

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

11.2 Fault Encoding

This section discusses the encoding of faults in XML.

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

```
(1) <s:Envelope
(2)     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
(3)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(4) <s:Header>
(5)     <wsa:Action>
(6)         http://schemas.xmlsoap.org/ws/2004/08/addressing/fault
(7)     </wsa:Action>
(8)     <wsa:MessageID>
(9)         uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(10)    </wsa:MessageID>
(11)    <wsa:RelatesTo>
(12)        uuid:d9726315-bc91-430b-9ed8-ce5ffb858a85
(13)    </wsa:RelatesTo>
(14) </s:Header>
(15)
(16) <s:Body>
(17)     <s:Fault>
(18)         <s:Code>
(19)             <s:Value> [Code] </s:Value>
(20)             <s:Subcode>
(21)                 <s:Value> [Subcode] </s:Value>
(22)             </s:Subcode>
(23)         </s:Code>
```



```

(24)     <s:Reason>
(25)         <s:Text xml:lang="en">  [Reason] </s:Text>
(26)     </s:Reason>
(27)     <s:Detail>
(28)         [Detail]
(29)     </s:Detail>
(30) </s:Fault>
(31) </s:Body>

```

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

s:Envelope/s:Header/wsa:Action

MUST be `http://schemas.xmlsoap.org/ws/2004/08/addressing/fault` for all faults.

s:Envelope/s:Header/wsa:MessageId

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

s:Envelope/s:Header/wsa:RelatesTo

Like any other reply, this MUST contain the MessageID of the original request which caused the fault.

s:Body/s:Fault/s:Value

MUST be one of s:Sender or s:Receiver, as specified in the Master Fault Table under the "Code" entry.

s:Body/s:Fault/s:Subcode/s:Value

For WS-Management-related messages, MUST be one of the subcode QNames defined in the Master Fault Table. If the service exposes custom methods or other messaging, this of course may be another QName not in the Master Fault Table.

s:Body/s:Fault/s:Reason

This OPTIONAL element SHOULD contain localized text explaining the fault in more detail. This is typically extracted from the "Reason" field of the Master Fault Table. However, the text may be adjusted to reflect a specific circumstance. This element may be repeated for each language. Note that the `xml:lang` attribute MUST be present.

s:Body/s:Fault/s:Detail

This OPTIONAL element SHOULD reflect the RECOMMENDED content from the Master Fault Table.

The above fault template is populated by examining entries from the Master Fault Table in 11.3, which includes all relevant faults from WS-Management and its underlying specifications. Note that s:Reason and s:Detail are always optional, but recommended.

11.3 NotUnderstood Faults

There is a special case for faults relating to mustUnderstand attributes on SOAP headers. SOAP specifications define the fault differently than the encoding in 11.2. In practice, the fault only varies in indicating the SOAP header that was not understood, the QName and namespace (line 3):

```

(1) <s:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope
(2)     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
(3)
(4)   <s:Header>
(5)     <s:NotUnderstood qname="QName of header" xmlns:ns="XML namespace of header"/>
(6)   </s:Header>
(7)
(8)   <wsa:MessageID>
(9)     uuid:d9726315-bc91-430b-9ed8-ce5ffb858a87
(10)  </wsa:MessageID>
(11)  <wsa:RelatesTo>
(12)    uuid:d9726315-bc91-430b-9ed8-ce5ffb858a85
(13)  </wsa:RelatesTo>
(14)
(15)  <s:Body>
(16)    <s:Fault>
(17)      <s:Code>
(18)        <s:Value>s:MustUnderstand</s:Value>
(19)      </s:Code>
(20)      <s:Reason>
(21)        <s:Text xml:lang="en-US">Header not understood</s:Text>
(22)      </s:Reason>
(23)    </s:Fault>
(24)  </s:Body>
(25) </s:Envelope>

```

The fault template shown above may be used in all cases of failure to process mustUnderstand attributes.

It is important that the wsa:RelatesTo be included so that the client can correlate the fault with the original request. Over transports other than HTTP in which may requests may be interlaced, this may be the only way to respond to the correct sender.

If the original wsa:MessageID itself is faulty and the connection is request-response oriented, the service MAY attempt to send back a fault without the wsa:MessageID and wsa:RelatesTo fields, or may simply fail to respond, as discussed in 11.4.

11.4 Degenerate Faults

In rare cases, the SOAP message may not contain enough information for a fault to be generated properly. For example, if the wsa:MessageID is garbled, it will be difficult for the service to return a fault which references the original message. With some transports, it may be impossible to reference the sender in order to return the fault.

If the transport guarantees a simple request-response pattern, then the service MAY send back a fault with no wsa:RelatesTo field. However, in some cases, there is no guarantee that the sender can be reached (the wsa:FaultTo contains an invalid address, so there is no way to deliver the fault).

In all of the cases, the service SHOULD revert to the rules of 10.4, in which no response is

sent. The service SHOULD attempt to log the requests in some way so as to help identify the defective client.

11.5 Master Fault Table

The following table includes all faults from this specification and all underlying specifications and should be taken as the normative fault list for WS-Management.

R11.5-1: A service MUST return faults from the following list when the operation that caused them was a message in this specification for which faults are specified. A conformant service MAY return other faults for messages which are not part of WS-Management.

It is critical to client interoperability that the same fault be used in identical error cases. If each service returns a distinct fault for "Not Found", it will be impossible to construct interoperable clients. In the tables that follow, the source specification of a fault is based on its QName.

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

11.5.1 wsman:AccessDenied

Fault Subcode	wsman:AccessDenied
Code	s:Sender
Reason	The sender was not authorized to access the resource
Detail	None
Comments	This is returned generically for all access denials relating to authentication or authorization failures. This should not be used to indicate locking or concurrency conflicts or other types of denials not related to security per se.
Applicability	Any message
Remedy	Client must acquire the correct credentials and retry the operation.

11.5.2 wsman:NoAck

Fault Subcode	wsman:NoAck
Code	s:Sender
Reason	The receiver did not acknowledge the event delivery.

Detail	None
Comments	This is returned when the client (subscriber) receives an event with a wsman:AckRequested header and does not (or cannot) acknowledge. The service should cease sending events and terminate the subscription.
Applicability	Any event delivery action (including heartbeats, dropped events, etc.) in any delivery mode
Remedy	For subscribers, the subscription must be resubmitted without the acknowledgement option. For services delivering events, the service should cancel the subscription immediately.

11.5.3 **wsa:ActionNotSupported**

Fault Subcode	wsa:ActionNotSupported
Code	s:Sender
Reason	The action is not supported by the service
Detail	<s:Detail> <wsa:Action> <i>Incorrect Action URI</i> </wsa:Action> </s:Detail> <!-- The unsupported Action URI is returned, if possible -->
Comments	This means that the requested action is not supported at all by the implementation. As an example, read-only implementations (supporting only wxf:Get, wsen:Enumerate) will return this for any other operations.
Applicability	All messages
Remedy	Client must consult the standard 'this' system object to see what actions are legal.

11.5.4 **wsman:Concurrency**

Fault Subcode	wsman:Concurrency
---------------	-------------------

Code	s:Sender
Reason	The action could not be completed due to concurrency or locking problems
Detail	
Comments	This means that the requested action could not be carried out due to either internal concurrency or locking problems or because another user is accessing the resource.
Applicability	All messages
Remedy	Client must wait and retry

11.5.5 wsman:AlreadyExists

Fault Subcode	wsman:AlreadyExists
Code	s:Sender
Reason	The sender attempted to create a resource which already exists
Detail	none
Comments	This is returned in cases where the user attempted to create resource which already exists.
Applicability	wxf:Create
Remedy	Client use wxf:Put or else create a resource with a different identity.

11.5.6 wsen:CannotProcessFilter

Fault Subcode	wsen:CannotProcessFilter
Code	s:Sender
Reason	The requested filter could not be processed.
Detail	<pre><s:Detail> <s:Text xml:lang="en"> Explanation of why filter cannot be processed </s:Text> </s:Detail></pre>

Comments This is typically returned for syntax errors or other semantic problems with the filter.

If the filter was valid, but the service cannot execute the filter due to misconfiguration, lack of resources or other service-related problems, more specific faults should be returned, such as wsman:QuotaLimit or wsman:InternalError.

Applicability wsen:Enumerate

Remedy Client fixes the filter problem and tries again.

11.5.7 wse:DeliveryModeRequestedUnavailable

Fault Subcode wse:DeliveryModeRequestedUnavailable

Code s:Sender

Reason The requested delivery mode is not supported.

Detail <s:Detail>
 <wse:SupportedDeliveryMode>... </wse:SupportedDeliveryMode>
 <wse:SupportedDeliveryMode>... </wse:SupportedDeliveryMode>
 ...
</s:Detail>

<!-- This is a simple list of one or more supported delivery mode URIs. This may be left empty. It is optional. -->

Comments This is returned for unsupported delivery modes for the specified resource.

If the stack supports the delivery mode in general, but not for the specific resource, this fault is still returned.

Other resources may support the delivery mode. The fault does not imply that the delivery mode is not supported by the implementation.

Applicability wse:Subscribe

Remedy Client should select one of the supported delivery modes.

11.5.8 wsman:DeliveryRefused

Fault Subcode	wsman:DeliveryRefused
---------------	-----------------------

Code	s:Receiver
------	------------

Reason	The receiver refuses to accept delivery of events and requests that the subscription be canceled.
--------	---

Detail	none
--------	------

Comments	This is returned by event receivers to force a cancellation of a subscription.
----------	--

This can happen when the client tried to Unsubscribe, but failed, or when the client has lost knowledge of active subscriptions and doesn't want to keep receiving events it no longer owns. This can help with cleanup of spurious or leftover subscriptions when clients are reconfigured or reinstalled and their previous subscriptions are still active.

Applicability	Any event delivery message in any mode
---------------	--

Remedy	The service should cease delivering events for the subscription and cancel the subscription, sending any applicable wse:SubscriptionEnd messages.
--------	---

11.5.9 wsa:DestinationUnreachable

Fault Subcode	wsa:DestinationUnreachable
---------------	----------------------------

Code	s:Sender
------	----------

Reason	No route can be determined to reach the destination role defined by the WS-Addressing To.
--------	---

Detail	<s:Detail>
--------	------------

<s:Text xml:lang="en">

Explanation of why endpoint cannot be reached

</s:Text>

<!-- The following elements are optional -->

<wsman:FaultDetail> *one of the URI values below*

```

</wsman:FaultDetail>
  ...any service-specific additional XML content...
</s:Detail>

```

Optionally, the wsman:FaultDetail field may contain one of the following
wsman:faultDetail/InvalidResourceURI
wsman:faultDetail/InvalidSystem

Comments This is returned as the general "Not Found" case for a Resource (or Resource/System combination), in which the ResourceURI and any applicable Selectors were valid, but the actual targeted object could not be found.

This fault is NOT used to merely indicate the resource is temporarily offline, which is indicated by wsa:EndpointUnavailable.

Applicability All request messages

Remedy Client should attempt diagnose the version of the service, query any metadata, and perform other diagnostic operations to determine why the request cannot be routed.

11.5.10 wsman:EncodingLimit

Fault Subcode	wsman:EncodingLimit
Code	s:Sender
Reason	An internal encoding limit was exceeded in a request or would be violated if the message were processed.
Detail	<pre> <s:Detail> <wsman:FaultDetail> <i>Optional; one of the enumeration values from below</i> </wsman:FaultDetail> <s:Text> <i>Optional textual description of the limit violation</i> </s:Text> </pre>

...any service-specific additional XML content...

</s:Detail>

In the <wsman:FaultDetail> element, one of the following enumeration values:

wsman:faultDetail/URILimitExceeded

(URI was too long)

wsman:faultDetail/MaxEnvelopeSize

(The requested maximum was too large)

wsman:faultDetail/MaxEnvelopeSizeExceeded

(The computed response is too large based on the client limit, but operation was read-only or never executed to start with)

wsman:faultDetail/ServiceEnvelopeLimit

(Service reached its own internal limit when computing response)

wsman:faultDetail/SelectorLimit

(Too many Selectors)

wsman:faultDetail/OptionLimit

(Too many Options)

wsman:faultDetail/CharacterSet

(Unsupported character set)

wsman:faultDetail/UnreportableSuccess

(Operation succeeded and cannot be reversed, but result is too large to send)

wsman:faultDetail/Whitespace

(Client-side whitespace usage is not supported)

wsman:faultDetail/EncodingType

(Used for unsupported MTOM or other encoding types)

Comments

This is returned when a system limit was exceeded, whether a published limit or a service-specific limit.

Applicability	All request messages
Remedy	Client should be reconfigured to send messages which fit the encoding limits of the service.

11.5.11 **wsa:EndpointUnavailable**

Fault Subcode	wsa:EndpointUnavailable	
Code	s:Receiver	
Reason	The specified endpoint is currently unavailable	
Detail	<pre><s:Detail> <wsa:RetryAfter> <i>xs:duration</i> </wsa:RetryAfter> <!-- optional --> ...optional service-specific XML content <wsman:FaultDetail> <i>one of the URI values below</i> </wsman:FaultDetail> </s:Detail></pre>	
Comments	<p>This is returned if the message was correct and the EPR was valid (valid ResourceURI and valid Selectors), but the specified resource is offline.</p> <p>In practice , it is difficult for a service to distinguish between "Not Found" cases and "Offline" cases. In general, wse:DestinationUnreachable is preferable.</p>	
Applicability	All request messages	
Remedy	Client can retry later, after the resource is again online.	

11.5.12 **wse:EventSourceUnableToProcess**

Fault Subcode	wse:EventSourceUnableToProcess	
Code	s:Sender	

Reason	The event source cannot process the subscription.
Detail	<pre> <s:Detail> <s:Text> <i>Text description of why subscription cannot be processed</i> </s:Text> ...any service-specific additional XML content... </s:Detail> </pre>
Comments	<p>This should be limited to cases where the event filter contains syntax or semantic errors.</p> <p>It should <i>not</i> be used to report other internal failures, such as resource limits, internal service errors, "Server Busy", "Access Denied", and any other more specific faults which provide more information to the client.</p>
Applicability	wse:Subscribe
Remedy	Client should repair the filter syntax.

11.5.13 wsen:FilterDialectRequestedUnavailable

Fault Subcode	wsen:FilterDialectRequestedUnavailable
Code	s:Sender
Reason	The requested filtering dialect is not supported.
Detail	<pre> <s:Detail> <wsen:SupportedDialect> </wsen:SupportedDialect> + </s:Detail> </pre>
Comments	This is returned when the client requests a filter type or query language that is not supported by the service.

The filter dialect may vary from resource to resource, or may apply to the entire service.

Applicability	wsen:Enumerate
Remedy	Client must switch to a supported dialect or do a simple enumeration with no filter.

11.5.14 wse:FilteringNotSupported

Fault Subcode	wse:FilteringNotSupported
Code	s:Sender
Reason	Filtering over the event source is not supported.
Detail	none
Comments	Returned when the service does not support filtered subscriptions for the specified event source, but only supports simple delivery of all events for the resource.

Note that the service may support filtering over a different event resource, or may not support filtering for *any* resource. The same fault applies.

Applicability	wse:Subscribe
Remedy	Client must subscribe using unfiltered delivery.

11.5.15 wsen:FilteringNotSupported

Fault Subcode	wsen:FilteringNotSupported
Code	s:Sender
Reason	Filtered enumeration is not supported.
Detail	
Comments	Returned when the service does not support filtering of enumerations at all, but only supports simple enumeration. If enumeration as a whole is not supported, then the correct fault is wsa:ActionNotSupported

Note that the service may support filtering over a different enumerable resource, or may not support filtering for *any* resource. The same fault applies.

Applicability	wse:Enumerate
Remedy	Client must switch to a simple enumeration.

11.5.16 wse:FilteringRequestedUnavailable

Fault Subcode	wse:FilteringRequestedUnavailable
Code	s:Sender
Reason	The requested filter dialect is not supported
Detail	<pre><s:Detail> <wse:SupportedDialect>.. </wse:SupportedDialect> + <wsman:FaultDetail> ..the URI below, if applicable </wsman:FaultDetail> </s:Detail></pre> <p>wsman:faultDetail/FilteringRequired</p>
Comments	<p>This is returned when the client requests a filter dialect that is not supported by the service.</p> <p>In some cases, a subscription <i>requires</i> a filter, as the result of an unfiltered subscription may be infinite or extremely large. In these cases, the wsman:faultDetail/FilteringRequired needs to be included in the s:Detail element.</p>
Applicability	wse:Subscribe
Remedy	Client must switch to a supported filter dialect or use no filtering.

11.5.17 wsman:IncompatibleEPR

Fault Subcode	wsman:IncompatibleEPR
Code	s:Sender
Reason	The EPR format used is not supported.
Detail	<s:Detail> <wsa:EndpointReference> ...the correct EPR, if possible </wsa:EndpointReference> </s:Detail>
Comments	<p>This is returned when the EPR is of the wrong format. If the service can translate the format to the required one automatically, the correct EPR should be returned in the Detail field.</p> <p>This is used when the service does <i>not</i> support the WS-Management common EPR model for lookup purposes, as opposed to direct resource access.</p>
Applicability	All messages
Remedy	Client extracts the new EPR from the Detail and retries the operation, or the client uses the wsman:system/2005/02/LookupEPR resource to translate to the correct EPR.

11.5.18 wsman:InternalError

Fault Subcode	wsman:InternalError
Code	s:Receiver
Reason	The service cannot comply with the request due to internal processing errors.
Detail	<s:Detail> <s:Text> <!-- Text description of the internal failure or system-specific error codes & text -->

```

</s:Text>
...service-specific extension XML elements....
<s:Detail>

```

Comments

This is a generic error for capturing internal processing errors within the service. For example, if the service cannot load the necessary executable images, or its configuration is corrupted, or hardware is not operating properly, or any 'unknown' or "unexpected" internal errors, this is the correct fault.

It is expected that the service must be reconfigured, restarted or reinstalled, so merely asking the client to retry will not succeed.

Applicability

All messages

Remedy

Client must repair the service out of band to WS-Management.

11.5.19 wsman:InvalidBookmark

Fault Subcode	wsman:InvalidBookmark
---------------	-----------------------

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>

```

wsman: faultDetail/Expired

wsman: faultDetail/Invalid

Comments

This is returned if a bookmark has expired or is corrupt, or otherwise unknown. If the service cannot detect "Expired" bookmarks, "Invalid" may always be returned.

Applicability wsen:Subscribe

Remedy Client must issue a new subscription without bookmarks at all or locate the correct bookmark.

11.5.20 wsen:InvalidEnumerationContext

Fault Subcode wsen:InvalidEnumerationContext

Code s:Receiver

Reason The supplied enumeration context is invalid.

Detail None

Comments An invalid enumeration context was supplied with the message. Typically, this will happen with a wsen:Pull.

The enumeration context may be invalid due to expiration, an invalid format, or reuse of an old context which is no longer being tracked by the service.

The service also can return this for any case where the enumerator has been terminated unilaterally on the service side, although one of the more descriptive faults is preferable, since this usually happens on out-of-memory (wsman:QuotaLimit), authorization failures (wsman:AccessDenied) or internal errors (wsman:InternalError).

Applicability wsen:Pull, wsen:Release (whether a pull-mode subscription, or a normal enumeration).

Remedy Client must abandon the enumeration and let the service time it out, as wsen:Release will fail as well.

11.5.21 wse:InvalidExpirationTime

Fault Subcode wse:InvalidExpirationTime

Code s:Sender

Reason Invalid expiration time

Detail	none
Comments	Expiration time was not valid at all or within the limits of the service. Used for outright errors (expirations in the past, etc.) or expirations too far into the future. If the service does not support expiration times at all, then a wsman:Unsupported fault should be returned with the correct detail code.
Applicability	wse:Subscribe
Remedy	Client issues a new subscription with a supported expiration time.

11.5.22 wsen:InvalidExpirationTime

Fault Subcode	wsen:InvalidExpirationTime
Code	s:Sender
Reason	The expiration time was not valid.
Detail	none
Comments	Since WS-Management recommends against implementing the wsen:Expiration feature, this fault should not occur with most implementations. Consult the WS-Enumeration specification for more information.
Applicability	wsen:Enumerate
Remedy	N/A

11.5.23 wse:InvalidMessage

Fault Subcode	wse:InvalidMessage
Code	s:Sender
Reason	The request message had unknown or invalid content and could not be processed
Detail	<s:Detail> <s:Text> ...identify the problem either with text or XML fragments

</s:Text>
</s:Detail>

Comments Generally not used in WS-Management, although it MAY be used for cases not covered by other faults.

If the content violates the schema, a wsman:SchemaValidationError fault should be sent. If specific errors occur in the subscription body, one of the more descriptive faults should be used.

This should not be used to indicate unsupported features, only unexpected or unknown content in violation of this specification.

Applicability WS-Eventing request messages

Remedy Client has a defect and should be corrected to issue valid messages which comply with this specification.

11.5.24 wsa:InvalidMessageInformationHeader

Fault Subcode	wsa:InvalidMessageInformationHeader
---------------	-------------------------------------

Code	s:Sender
------	----------

Reason	A message information header is not valid and the message cannot be processed.
--------	--

Detail	<s:Detail> ...the invalid header... </s:Detail>
--------	---

Comments This may occur with any type of SOAP header error. The header may be invalid in terms of schema, value, or may constitute a semantic error.

This should not be used to indicate an invalid resource URI, bad Selector, or other WS-Management-specific concepts, but should be limited to structural problems with the SOAP payload prior to interpretation in the WS-Management context.

Examples are repeated MessageIDs, missing RelatesTo on a response, badly formed addresses, or any other missing header content.

Applicability	All messages
Remedy	Major client defect. The SOAP packets are not correctly formed.

11.5.25 wsman:InvalidOptions

Fault Subcode	wsman:InvalidOptions
Code	s:Sender
Reason	One or more options were not valid.
Detail	<pre><s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail></pre> <p>wsman:faultDetail/NotSupported wsman:faultDetail/InvalidName wsman:faultDetail/InvalidValue</p>
Comments	This generically covers all cases where the option names or values are not valid or they are used in incorrect combinations.
Applicability	All request messages
Remedy	Client should retrieve the catalog entry for the resource and determine how to correct the invalid option values.

11.5.26 wsman:InvalidParameter

Fault Subcode	wsman:InvalidParameter
Code	s:Sender
Reason	An operation parameter was not valid
Detail	<pre><s:Detail> <wsman:FaultDetail> If possible, one of the following URI values</pre>

</wsman:FaultDetail>

</s:Detail>

wsman: faultDetail/TypeMismatch

wsman: faultDetail/InvalidName

Comments Returned when a parameter to a custom action was not valid.

This is a default for new implementations which need to have a generic fault for this case. The method may also return any specific fault of its own.

Applicability All messages with custom actions

Remedy Client should consult the WSDL for the operation and determine how to supply the correct parameter.

11.5.27 wxf:InvalidRepresentation

Fault Subcode wxf: InvalidRepresentation

Code s: Sender

Reason The XML content was invalid.

Detail <s:Detail>

<wsman:FaultDetail>

If possible, one of the following URI values

</wsman:FaultDetail>

</s:Detail>

wsman: faultDetail/InvalidValues

wsman: faultDetail/MissingValues

wsman: faultDetail/InvalidNamespace

wsman: faultDetail/InvalidFragment

Comments This may be returned when the input XML is not valid semantically or uses

the wrong schema for the resource.

However, a `wsman:SchemaValidationError` fault should be returned if the error is related to XML Schema violations per se, as opposed to invalid semantic values.

Note the anomalous case where a schema violation does not occur, but the namespace is simply the wrong one, in which `wsman:faultDetail/WrongNamespace` is returned.

Applicability	<code>wxf:Put</code> , <code>wxf:Create</code>
Remedy	Client defect. The client should correct the input XML.

11.5.28 `wsman:InvalidSelectors`

Fault Subcode	<code>wsman:InvalidSelectors</code>
Code	<code>s:Sender</code>
Reason	The Selectors for the resource were not valid
Detail	<pre><s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail></pre> <code>wsman:faultDetail/InsufficientSelectors</code> <code>wsman:faultDetail/UnexpectedSelectors</code> <code>wsman:faultDetail/TypeMismatch</code> <code>wsman:faultDetail/InvalidValue</code> <code>wsman:faultDetail/AmbiguousSelectors</code> <code>wsman:faultDetail/DuplicateSelectors</code>
Comments	This covers all cases where the specified Selectors were incorrect or unknown for the specified resource.
Applicability	All request messages
Remedy	Client should retrieve documentation or metadata and correct the Selectors.

11.5.29 wsman:InvalidSystem

Fault Subcode	wsman:InvalidSystem
Code	s:Sender
Reason	A valid wsman:System URI is required
Detail	<s:Detail> <wsman:FaultDetail> <i>one of the values below</i> </wsman:FaultDetail> </s:Detail>

wsman:faultDetail/MissingSystem **wsman:faultDetail/InvalidSystem**

Comments	This is returned when the service does not support a default system and requires an explicit wsman:System value, or when the specified system was not valid.
Applicability	All messages
Remedy	Client must discover a valid wsman:System value.

11.5.30 wsa:MessageInformationHeaderRequired

Fault Subcode	wsa:MessageInformationHeaderRequired
Code	s:Sender
Reason	A required header was 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	Major client defect. The SOAP packets are not correctly formed.

11.5.31 wsman:MetadataRedirect

Fault Subcode	wsman:MetadataRedirect
Code	s:Sender
Reason	The requested metadata is not available at the current address.
Detail	<pre><s:Detail> <wsa:EndpointReference> ...<i>SOAP address</i> ... </wsa:EndpointReference> <wsa:EndpointReference> ...<i>SOAP address</i> ... </wsa:EndpointReference> <wsman:URL> <i>http address</i> </wsman:URL> <wsman:URL> <i>https address</i> </wsman:URL> <wsman:URL> <i>ftp address</i> </wsman:URL> ...etc. </s:Detail></pre>
Comments	<p>This is returned in cases where metadata is requested and it is not available, such as a WSDL document, an XML schema, or other metadata, and the service knows the actual location.</p> <p>The address(es) should be encoded in the s:Detail element. If the address is a SOAP endpoint, it is wrapped in wsa:EndpointReference wrappers. If the address is a URL-based address (such as an HTTP web address), then a wsman:URL wrapper is used.</p> <p>More than one address may be used.</p>
Applicability	wxf:Get, wsen:Enumerate
Remedy	Client should extract addresses from the Detail element, if possible, and retry with the new addresses.

11.5.32 wsman:QuotaLimit

Fault Subcode	wsman:QuotaLimit
Code	s:Sender
Reason	The service is busy servicing other requests.
Detail	<s:Detail>

```
<s:Text> reason </s:Text>
</s:Detail>
```

Comments	This is returned when the SOAP message is otherwise correct, but the service has reached a resource or quota limit.
Applicability	All messages
Remedy	Client can retry later

11.5.33 wsman:RenameFailure

Fault Subcode	wsman:RenameFailure
---------------	---------------------

Code	s:Sender
Reason	The Selectors for the resource were not valid
Detail	<s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail>

wsman: faultDetail/InvalidResourceURI
wsman: faultDetail/InvalidSystem
wsman: faultDetail/TargetAlreadyExists
wsman: faultDetail/InvalidAddress
wsman: faultDetail/InvalidSelectorAssignment

Comments	This covers all cases where the specified Selectors were incorrect.
Applicability	All request messages
Remedy	Client should retrieve the catalog entry and correct the Selectors.

11.5.34 wsman:SchemaValidationError

Fault Subcode	wsman:SchemaValidationError
---------------	-----------------------------

Code	s:Sender
Reason	The supplied SOAP violates the corresponding XML Schema definition.
Detail	<pre><s:Detail> <s:Text> Service-specific error messages as to the schema violation. </s:Text> </s:Detail></pre>
Comments	<p>Used for any XML parsing failure or schema violations.</p> <p>Note that full validation of the SOAP against schemas is not expected in real-time, but processors may in fact notice schema violations, such as type mismatches. In all of these cases, this fault applies.</p> <p>In debugging modes where validation is in fact occurring, this should be returned for <i>all</i> errors noted by the validating parser.</p>
Applicability	All messages
Remedy	Client corrects the message

11.5.35 wsen:TimedOut

Fault Subcode	wsen:TimedOut
Code	s:Receiver
Reason	The enumerator has timed out and is no longer valid.
Detail	none
Comments	This should not be used in WS-Management due to overlap with the wsman:TimedOut which covers all the other messages.
Applicability	wsen:Pull
Remedy	The client can retry the wsen:Pull.

11.5.36 wsman:TimedOut

Fault Subcode	wsman:TimedOut
---------------	----------------

Code	s:Receiver
Reason	The operation has timed out
Detail	none
Comments	The operation could not be completed within the wsman:OperationTimeout value or else an internal override timeout was reached by the service while trying to process the request.
	This is also returned in all enumerations when there is no content available for the current wsen:Pull request. Clients may simply retry the wsen:Pull again until a different fault is returned.
Applicability	All requests
Remedy	Client may retry the operation.
	If the operation was a write (delete, create, execute), the client should consult the system operation log before blindly attempting a retry, or attempt a wxf:Get or other read operation to try and discover the result of the previous operation.

11.5.37 wse:UnableToRenew

Fault Subcode	wse:UnableToRenew
Code	s:Sender
Reason	The subscription could not be renewed
Detail	<pre><s:Detail> <s:Text> Optional service-specific error messages as to why the Renew failed </s:Text> </s:Detail></pre>
Comments	This is returned in all cases where the subscription cannot be renewed, but is otherwise valid.
Applicability	wse:Renew
Remedy	Client must issue a new subscription.

11.5.38 wse:UnsupportedExpirationType

Fault Subcode	wse:UnsupportedExpirationType
Code	s:Sender
Reason	The specified expiration type is not supported
Detail	none
Comments	A specific time for expiration is not supported (as opposed to duration). This fault should not be used if the value itself is incorrect, only if the <i>type</i> is not supported.
Applicability	wse:Subscribe
Remedy	Client corrects the expiration to use a duration.

11.5.39 wsen:UnsupportedExpirationType

Fault Subcode	wsen:UnsupportedExpirationType
Code	s:Sender
Reason	The specified expiration type is not supported
Detail	none
Comments	The specified expiration type is not supported. For example, a specific time based expiration type may not be supported (as opposed to a duration based expiration type). This fault should not be used if the value itself is incorrect, only if the <i>type</i> is not supported.
Applicability	wsen:Enumerate
Remedy	Client corrects the expiration time or omits it and retries.

11.5.40 wsman:UnsupportedFeature

Fault Subcode	wsman:UnsupportedFeature
Code	s:Sender

Reason	The specified feature is not supported
Detail	<pre> <s:Detail> <wsman:FaultDetail> If possible, one of the following URI values </wsman:FaultDetail> </s:Detail> </pre> <p>One of the following:</p> <ul style="list-style-type: none"> wsman: faultDetail/AuthorizationMode wsman: faultDetail/AddressingMode wsman: faultDetail/Ack wsman: faultDetail/OperationTimeout wsman: faultDetail/Locale wsman: faultDetail/ExpirationTime wsman: faultDetail/FragmentLevelAccess wsman: faultDetail/Replay wsman: faultDetail/DeliveryRetries wsman: faultDetail/Heartbeats wsman: faultDetail/Bookmarks wsman: faultDetail/MaxElements wsman: faultDetail/MaxTime wsman: faultDetail/MaxEnvelopeSize wsman: faultDetail/MaxEnvelopePolicy wsman: faultDetail/FilteringRequired wsman: faultDetail/InsecureAddress wsman: faultDetail/FormatMismatch wsman: faultDetail/FormatSecurityToken
Comments	Used to indicate than an unsupported feature was attempted.
Applicability	any message
Remedy	Client corrects or removes the unsupported feature request and retries.

12.0 WS-Management XSD

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

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

```
<xs:schema
  targetNamespace="http://schemas.xmlsoap.org/ws/2005/02/management "
  xmlns="http://schemas.xmlsoap.org/ws/2005/02/management "
  xmlns:tns="http://schemas.xmlsoap.org/ws/2005/02/management "
  xmlns:xs="http://www.w3.org/2001/XMLSchema "
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing "
  xmlns:s="http://www.w3.org/2003/05/soap-envelope "
  elementFormDefault="qualified"
  >

  <xs:complexType name="SystemType">
    <xs:sequence>
      <xs:element name="System" type="tns:SystemType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:anyURI"/>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <xs:element name="Replay">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:unsignedInt">
          <xs:anyAttribute namespace="##other" processContents="lax" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="AckRequested">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:anyAttribute namespace="##other" processContents="lax" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="FragmentTransfer">
    <xs:complexType>
```

```

    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Dialect" type="xs:anyURI" use="optional"/>
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="MaxEnvelopeSize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:unsignedLong">
        <xs:attribute name="Policy" type="xs:string" use="optional"/>
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:complexType name="RenameType">
  <xs:sequence maxOccurs="unbounded">
    <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:element name="OperationTimeout">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:duration">
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="Locale">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:complexType name="SelectorType">
  <xs:complexContent mixed="true">
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="Name" type="xs:token" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

```

```

        <xs:anyAttribute namespace="##other" processContents="lax" />
    </xs:restriction>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="OptionType">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="Name" type="xs:token" use="required"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="SelectorSetType">
    <xs:sequence>
        <xs:element name="Selector" type="tns:SelectorType"
            minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="OptionSetType">
    <xs:sequence>
        <xs:element name="Option" type="tns:OptionType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="BookmarkType">
    <xs:complexContent mixed="true">
        <xs:restriction base="xs:anyType">
            <xs:sequence>
                <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:anyAttribute namespace="##other" processContents="lax" />
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="XmlFragmentType">
    <xs:complexContent mixed="true">
        <xs:restriction base="xs:anyType">
            <xs:sequence>
                <xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:anyAttribute namespace="##other" processContents="skip" />
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>

<xs:element name="SelectorSet" type="tns:SelectorSetType"/>
<xs:element name="OptionSet" type="tns:OptionSetType"/>
<xs:element name="Rename" type="tns:RenameType"/>

```

```

<xs:element name="RenamedTo" type="tns:RenameType"/>
<xs:element name="SendBookmarks" type="xs:token"/>
<xs:element name="System" type="tns:SystemType"/>
<xs:element name="Heartbeats" type="xs:duration"/>
<xs:element name="Bookmark" type="tns:BookmarkType"/>
<xs:element name="XmlFragment" type="tns:XmlFragmentType"/>

<xs:element name="MaxElements" type="xs:unsignedLong"/>
<xs:element name="MaxTime" type="xs:duration"/>

<xs:element name="ConnectionRetry">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:duration">
        <xs:attribute name="Total" type="xs:unsignedInt" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="Auth">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Profile" type="xs:anyURI" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:complexType name="EventType">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="Action" type="xs:anyURI" use="required"/>
      <xs:anyAttribute namespace="##other" processContents="skip" />
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="EventBlockType">
  <xs:sequence>
    <xs:element name="Event" type="tns:EventType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="Events" type="tns:EventBlockType"/>
</xs:schema>

```


13.0 Acknowledgements

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

Paul C. Allen, Microsoft
Don Box, Microsoft
Jerry Duke, Intel
David Filani, Intel
Kirill Gavrylyuk, Microsoft
Omri Gazitt, Microsoft
Frank Gorishek, AMD
Arvind Kumar, Intel
Brad Lovering, Microsoft
Pat Maynard, Intel
Sasha Nosov, Microsoft
Brian Reistad, Microsoft
Tom Slight, Intel
Marvin Theimer, Microsoft
Dave Tobias, AMD
John Tollefsrud, Sun
Anders Vinberg, Microsoft

14.0 References

[HTTP]

R. Fielding et al, "[IETF RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1](#)," June 1999

[HTTPS]

E. Rescorla, "[RFC 2818: HTTP over TLS](#)," May 2000

[RFC 2119]

S. Bradner, "[RFC 2119: Key words for use in RFCs to Indicate Requirement Levels](#)," March 1997

[SOAP 1.2]

M. Gudgin, et al, "[SOAP Version 1.2 Part 1: Messaging Framework](#)," June 2003.

[MTOM]

M. Gudgin et al, "[SOAP Message Transmission Optimization Mechanism](#)," August 2004

[BP1]

K. Ballinger et al, "[WS-I Basic Profile Version 1.0a](#)," April 2004

[SOAP-UDP]

H. Combs et al, "[SOAP over UDP](#)," September 2004

[WS-Addressing]

D. Box et al, "[Web Services Addressing \(WS-Addressing\)](#)," August 2004

[WS-Enumeration]

J. Alexander et al, "[Web Services Enumeration \(WS-Enumeration\)](#)," September 2004

[WS-Eventing]

D. Box et al, "[Web Services Eventing \(WS-Eventing\)](#)," August 2004

[WS-MetadataExchange]

K. Ballinger et al, "[Web Services Metadata Exchange \(WS-MetadataExchange\)](#)," September 2004

[WS-SecureConversation]

G. Della-Libera et al, "[Web Services Secure Conversation Language \(WS-SecureConversation\)](#)," May, 2004

[WS-Security]

A. Nadalin et al, "[Web Services Security: SOAP Message Security 1.0](#)," May, 2004

[WS-Transfer]

J. Alexander et al, "[Web Services Transfer \(WS-Transfer\)](#)," September 2004

[WSDL 1.1]

E. Christensen et al, "[Web Services Description Language \(WSDL\) 1.1](#)," March 2001.

[XML Schema, Part 1]

H. Thompson et al, "[XML Schema Part 1: Structures](#)," May 2001.

[XML Schema, Part 2]

P. Biron et al, "[XML Schema Part 2: Datatypes](#)," May 2001.

[RFC 2396] Uniform Resource Identifiers (URI) : Generic Syntax

<http://www.ietf.org/rfc/rfc2396.txt>

[WS-I Basic Profile]

<http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>

[MTOM] SOAP Message Transmission Optimization Mechanism

<http://www.w3.org/TR/2004/PR-soap12-mtom-20041116/>

[RFC 3066] Tags for the Identification of Languages

<http://www.ietf.org/rfc/rfc3066.txt>