Web Services for Management (WS-Management)

October 2004

Authors

Akhil Arora, Sun
Alan Geller, Microsoft (editor)
Jackson He, Intel
Chris Kaler, Microsoft
Ray McCollum, Microsoft
Milan Milenkovic, Intel
Paul Montgomery, AMD
Junaid Saiyed, Sun
Enoch Suen, Dell

Copyright Notice

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

Published specification.

Table of Contents

1. Introduction

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

A crucial application for these services is in the area of **systems management**. To promote interoperability between management applications and managed resources, this specification identifies a core set of Web service specifications and usage requirements 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, 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 compliant 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 agents 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, such as WS-ReliableMessaging and WS-AtomicTransactions.
- Minimize additional mechanism beyond the current Web service architecture.

2. Notations and Terminology

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

2.1 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.

2.2 XML Namespaces

The XML namespace URI that MUST be used by implementations of this specification is:

```
http://schemas.xmlsoap.org/ws/2004/10/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/2004/10/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]
mex	http://schemas.xmlsoap.org/ws/2004/09/mex	WS-MetadataExchange [WS-MetadataExchange]
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]

2.3 Terminology

Agent

An application that provides management services for a System by exposing a set of Resource Services. The Agent provides management operations within its local scope.

Manager

A Web service that is used to manage one or more Systems by sending messages to and/or receiving messages from an Agent for that System.

Resource Instance

A single manageable item, such as a disk drive or a running process. Also called a Resource or an Instance.

Resource Service

A Web service that provides access to a single category of manageable items, such as disk drives or running processes, that share the same operations and representation schema.

System

A top-level managed entity composed of one or more Resource Instances. For instance, a PC is a System that contains Resources such as disk drives and running processes.

2.4 Compliance

An implementation is not compliant with this specification if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined herein. A SOAP Node MUST NOT use the XML namespace identifier for this specification (listed in Section 2.2) within SOAP Envelopes unless it is compliant with this specification.

3. Addressing

WS-Management Resources are identified by the following information:

- The transport address (URL) of the Agent that provides the Resource Services.
- The unique identifier (URI) of the System that the Resource is part of. If the System is uniquely identified by the Agent address, this component may be omitted.
- The unique identifier (URI) of the Resource Service that provides access to the Resource.
- Zero or more keys (string name/value pairs) that identify the Resource

These components are represented in a WS-Addressing Endpoint Reference as follows:

- The Agent’s transport address is mapped to the [address] property.
- The System identifier is mapped to a [reference property] named wsman:System.
- The Resource Service identifier is mapped to a [reference property] named wsman:ResourceURI.
- Each key is mapped to a [reference parameter] named wsman:Key with an element attribute named Name that contains the key name; the contents of the element is the key value.

When serialized into a SOAP message, these components are represented as SOAP headers, as follows:

```
<s:Envelope
    xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.xmlsoap.org/ws/2004/10/management">
  <s:Header>
    ...
    <wsa:To>Agent transport address</wsa:To>
    <wsman:System>System identifier</wsman:System> ?
    <wsman:ResourceURI>Resource Type identifier</wsman:ResourceURI>
    <wsman:Key Name="key name">key value</wsman:Key> *
    ...
  </s:Header>
  ...
</s:Envelope>
```

There may be different representations of the identifying information of a Resource Instance used for other purposes, such as command line entry.

4. General Messaging

4.1 Operation time out

Most management operations are time-critical due to quality of service constraints and obligations. If they cannot be completed in a specified time, usually an alternate approach is required to resolve an issue. The Agent should be aware of any such constraints a Manager may have.

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

All request messages MAY contain a wsman:OperationTimeout header element that indicates the maximum amount of time the Manager is willing to wait for the Agent to issue a response. The Agent SHOULD issue a wsman:OperationTimeout fault as follows if this time is exceeded and the operation is not yet complete:

[Code]	s12:Recipient
[Subcode]	wsman:OperationTimeout
[Reason]	"the operation could not be completed in the time requested"

If the Agent faults a request for an operation time out, it SHOULD undo any effects of the operation that were accomplished before the time out.

A correctly formatted 30-second timeout appears as follows in the SOAP header:

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

4.2 Locale

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

```
<wsman:Locale lang="xs:language" s:mustUnderstand="false" />
```

All request messages MAY contain a wsman:Locale header element whose "lang" attribute indicates the locale of the client using an RFC 1766 (ISO 639) language code. The Agent SHOULD utilize this value when composing the response message and adjust any localizable values accordingly.

The following example indicates the manager prefers a response localized to U.S. English:

```
<wsman:Locale lang="en-us"/>
```

4.3 Data freshness

Many implementations cache expensive values, but managers need a way to signal that recomputed, up-to-date values are required.

```
<wsman:NoCache s:mustUnderstand="false" />
```

All request messages MAY contain a wsman:NoCache header element that indicates that the Agent SHOULD NOT used cached values for the content of the response.

5. Resource Access

If a Resource provides a machine-readable representation of its state, and exposes read, update, create, and delete operations that operate on that state, it MUST do so by implementing WS-Transfer. Similarly, if a Resource exposes enumerable items such as tables, logs, or containers, the Resource MUST implement WS-Enumeration to support that enumeration.

5.1 WS-Transfer

WS-Management defines the following header flags (empty elements) that MAY be used with the indicated WS-Transfer operations:

```
<wsman:SummaryPermitted s:mustUnderstand="false" />
```

This header MAY be included on Get requests. It indicates that the Agent SHOULD return an abbreviated representation, if available.

```
<wsman:ReturnResource s:mustUnderstand="true" />
```

This header MAY be included on Put and Create requests. It indicates that the Agent MUST return the new representation of the updated or created resource.

Note that while Agents SHOULD support these headers, they MAY ignore wsman:SummaryPermitted. All WS-Management Agents MUST support the wsman:ReturnResource header; for this reason, it is appropriate to attach the SOAP mustUnderstand attribute to this header with a true value.

The following header MAY be included by a resource instance with its response to the indicated WS-Transfer operations:

```
<wsman:NewKeys s:mustUnderstand="true">
  <wsman:Key Name="key name">key value</wsman:Key> +
</wsman:NewKeys>
```

If the Put operation caused one or more instance keys for the resource to change, this header will contain the complete set of wsman:Key elements that identify the updated resource instance. A Manager MUST recognize and appropriately process the contents of a wsman:NewKeys header.

5.2 WS-Enumeration

```
<wsman:SummaryPermitted s:mustUnderstand="false" />
```

The wsman:SummaryPermitted header flag defined above in section 5.1 may also be used with Pull requests.

6. Eventing

If a Resource can emit events and allows Managers to subscribe to and receive event messages, it MUST do so by implementing WS-Eventing.

6.1 General

6.1.1 Subscription managers and identifiers

WS-Eventing introduces the concept of a subscription manager, which is a Web service that an event source delegates the management of a subscription to. While WS-Eventing places no restrictions on the EPR for the subscription manager, WS-Management constrains this EPR in order to define a consistent mechanism and to allow the subscription manager EPR to be known in cases where the subscription occurs as a result of configuration. All Resources that are event sources MUST create subscription manager EPRs by extending the Resource's

EPR with a reference parameter named wse:Identifier. This reference parameter element MUST have no attributes and simple content, of type xs:anyURI.

Subscribers MAY include a wsman:ProposedID header in a Subscribe message. If present, the contents of this header is a URI whose value is a proposed subscription ID, as follows:

```
<wsman:ProposedID> xs:anyURI </wsman:ProposedID>
```

The Resource that received the Subscribe message MUST use this ID to form the subscription manager EPR by setting wse:Identifier to the value of the wsman:ProposedID header element. If the Resource cannot do this, it MUST raise a wsman:InvalidProposedID fault:

[Code]	s12:Sender
[Subcode]	wsman:InvalidProposedID
[Reason]	"the proposed subscription ID could not be used"

6.1.2 Expiration

If a Subscribe or Renew request contains a requested Expiration of type xs:dateTime, the Resource MAY include an Expiration of type xs:duration in the response message. Systems are required to have an internal clock, but there is no requirement that the clock be synchronized with other Systems, or indeed that the internal clock provide absolute time at all (as opposed to relative time). Therefore, Systems are not required to express subscription expiration as an absolute time.

6.1.3 Event message format

All event messages sent using Push, Batched, or Trap mode MUST include a wse:Identifier header that contains the URI that identifies the related subscription.

6.2 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, Batched, that allows an event source to bundle multiple outgoing event messages into a single SOAP envelope. For this delivery mode, the wse:Delivery element has the following format:

```
<wse:Delivery
      Mode="http://schemas.xmlsoap.org/ws/2004/10/management/Batched">
  <wse:NotifyTo>
    wsa:EndpointReferenceType
  </wse:NotifyTo>
  <wsman:MaxItems> xs:positiveInteger </wsman:MaxItems> ?
  <wsman:MaxTime> xs:duration </wsman:MaxTime> ?
  <wsman:MaxCharacters> xs:positiveInteger </wsman:MaxCharacters> ?
</wse:Delivery>
```

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

wse:Delivery/@Mode

MUST be "http://schemas.xmlsoap.org/ws/2004/10/management/Batched".

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:MaxItems

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:MaxCharacters

This optional element MAY contain a positiveInteger that indicates the maximum number of characters in the SOAP body for the event batch. The Resource MUST NOT deliver a batch of event items whose total character count exceeds this value. Because of the variable size of SOAP Body declarations and the unpredictable size of the SOAP Header, this does not refer to the entire maximum message size.

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.

If Batched mode is requested in a Subscribe message, and none of MaxItems, MaxCharacters, and MaxTime are present, the Resource MUST issue a wsman:InvalidBatchParameter fault.

[Code]	s12:Sender
[Subcode]	wsman:InvalidBatchParameter
[Reason]	"at least one batching parameter must appear"

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

```
<s:Envelope ...>
  <s:Header>
    ...
    <wsa:Action>
        http://schemas.xmlsoap.org/ws/2004/10/management/Events
    </wsa:Action>
    ...
  </s:Header>
  <s:Body>
    <wsman:Events>
      <wsman:Event Action="event action URI">
        ...
      </wsman:Event> +
    </wsman:Events>
  </s:Body>
</s:Envelope>
```

s:Envelope/s:Header/wsa:Action

MUST be http://schemas.xmlsoap.org/ws/2004/10/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.

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 characters in the SOAP body:

```
<wse:Delivery
    Mode="http://schemas.xmlsoap.org/ws/2004/10/management/Batched">
  <wse:NotifyTo>
    <wsa:Address>http://2.3.4.5/client</wsa:Address>
  </wse:NotifyTo>
  <wsman:MaxItems>10</wsman:MaxItems>
  <wsman:MaxTime>PT20S</wsman:MaxTime>
  <wsman:MaxCharacters>8192</wsman:MaxCharacters>
</wse:Delivery>
```

The following example shows an example of batched delivery that conforms to this specification. The salient features are the present of a wse:Identifier header as described above in section 6.1.3, the ws:Action specific to batched delivery, and the actual wsman:Event items juxtaposed in the env:Body acting as wrappers for the real events:

```
<s:Envelope
     xmlns:s="http://www.w3.org/2003/05/soap-envelope"
     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing
     xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing">
  <s:Header>
    <wsa:To env:mustUnderstand="true">http://2.3.4.5/client</wsa:To>
    <wse:Identifier>
      uuid:d795621f-a01d-4542-85f9-bdf50c00cb2e
    </wse:Identifier>
    <wsa:Action>
        http://schemas.xmlsoap.org/ws/2004/10/management/Events
    </wsa:Action>
  </s:Header>
  <s:Body>
    <wsman:Events>
      <wsman:Event
          Action="http://schemas.xmlsoap.org/2004/10/diskspacechange">
        <DiskChange
            xmlns="http://schemas.xmlsoap.org/2004/10/diskspacechange">
          <Drive> C: </Drive>
          <FreeSpace> 802012911 </FreeSpace>
        </DiskChange>
      </wsman:Event>
      <wsman:Event
          Action="http://schemas.xmlsoap.org/2004/10/diskspacechange">
        <DiskChange
            xmlns="http://schemas.xmlsoap.org/2004/10/diskspacechange">
          <Drive> D: </Drive>
          <FreeSpace> 1402012913 </FreeSpace>
        </DiskChange>
      </wsman:Event>
    </wsman:Events>
  </s:Body>
</s:Envelope>
```

6.3 Pull delivery mode

In some circumstances, polling for events is an effective way of controlling data flow and balancing timeliness against processing ability.

WS-Management defines a custom event delivery mode, Pull, 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:

```
<wse:Delivery
      Mode="http://schemas.xmlsoap.org/ws/2004/10/management/Pull" />
```

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

wse:Delivery/@Mode
MUST be "http://schemas.xmlsoap.org/ws/2004/10/management/Pull".

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 wsman:EnumerationContext element that contains an enumeration context (that is, the wsman:EnumerationContext element is of type wsen:EnumerationContextType) as per [WS-Enumeration] that the subscriber may use to poll for event messages by sending a Pull request to the event source with that enumeration context.

That is, the body of the SubscribeResponse message must have the following format:

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

If the subscriber issues a Pull request using the enumeration context from the SubscriptionResponse, and one or more event messages are returned, the wsen:Items element in the PullResponse MUST contain a list of wsman:Event elements, as defined above in section 5.2.

6.4 Trap delivery mode

UDP multicast is a very efficient way to distribute small event messages to many subscribers.

WS-Management defines a custom event delivery mode, Trap, which allows an event source to send event messages using UDP multicast. For this delivery mode, the wse:Delivery element has the following format:

```
<wse:Delivery
      Mode="http://schemas.xmlsoap.org/ws/2004/10/management/Trap" />
```

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

wse:Delivery/@Mode
MUST be "http://schemas.xmlsoap.org/ws/2004/10/management/Trap".

If Trap 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 wsman:MulticastAddress element that contains an EPR specifying the UDP multicast group address on which events may be received.

That is, the body of the SubscribeResponse message must have the following format:

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

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

s:Body/wse:SubscriptionResponse/wsman:MulticastAddress

> This required element MUST contain an endpoint reference that identifies the multicast address on which the event source will multicast event messages. Generally, this EPR will contain only a wsa:Address element whose value will be a soap.udp address.

Event messages sent using UDP multicast MUST be sent according to the [SOAP-UDP] specification.

6.5 Resumable subscriptions

In many cases, management event sources store historical events in a log as well as sending them out to subscribers. In such cases, it is often desirable for a subscriber whose subscription has ended to resubscribe at the point where they left off. In order to achieve this, the subscriber needs to receive any events that were logged since the last event received by the subscriber.

WS-Management defines a set of extensions to WS-Eventing to support this feature. In particular, it introduces the notion of a resumption context, which is an XML element containing arbitrary data that is interpreted by the event source to define a specific point where a subscription may start.

Since not all event sources will understand these extensions, this feature is designed so that a subscriber can recognize whether or not the event source actually processed the extensions.

In the Subscribe request, the subscriber MAY add the following element as a child of the wse:Subscribe element to indicate that they want to start a subscription at a known point and/or that they want the new subscription to be resumable:

```
<wsman:ResumeAt From="earliest|next|context" AllowResumption="xs:boolean" ? >
  ...
</wsman:ResumeAt>
```

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

wsman:ResumeAt/@From

> This required attribute MUST be either "earliest", "next", or "context". If it is "earliest", then the event source MUST start the subscription from the earliest event message available if it supports resumption. If it is "next", then the event source MUST start the subscription from the next event that gets generated. If it is "context", then the contents of the wsman:ResumeAt element MUST be a resumption context that had earlier been received by the subscriber from the event source, and the event source MUST start the new subscription from the next event after that designated by the resumption context, if the event source supports resumption.

If the event source supports resumption but does not support the requested resumption type, it MUST issue a wsman:ResumptionTypeNotSupported fault as follows:

[Code]	s12:Sender
[Subcode]	wsman:ResumptionTypeNotSupported
[Reason]	"The requested resumption type is not supported"

The "next" resumption type MUST be supported by all event sources that support resumption.

If the resumption context is invalid and the event source supports resumption, then the event source MUST issue a wsman:InvalidResumptionContext fault as follows:

[Code]	s12:Sender
[Subcode]	wsman:InvalidResumptionContext
[Reason]	"The resumption context is invalid"

wsman:ResumeAt/@AllowResumption

This optional attribute has an implicit value of "false" if it is not specified. If it is specified and has a value of "true", then the event source MUST attach a resumption context to every event message, as specified below, if it supports resumption. If the event source does not support providing a resumption context but does recognize this extension, it MUST issue a wsman:ResumptionNotSupported fault as follows:

[Code]	s12:Sender
[Subcode]	wsman:ResumptionNotSupported
[Reason]	"This event source does not support resumable subscriptions"

If the event source receives and accepts a Subscribe request containing the wsman:ResumeAt extension element and the event source supports this extension, then the event source MUST include a wsman:Resumed element in the SubscribeResponse. The absence of this element allows the subscriber to determine that the subscription was not resumed as requested because the event source does not support this extension.

If wsman:ResumeAt/@AllowResumption is requested in a Subscribe message and the event source accepts the subscription request, the SubscribeResponse element in the reply message MUST contain a wsman:ResumptionContext element that contains the current resumption context if the event source supports resumption.

Thus, the body of the SubscribeResponse message MUST have the following format if the event source supports this extension:

```
<s:Body ...>
  <wse:SubscribeResponse ...>
    ...
    <wsman:Resumed />
    <wsman:ResumptionContext> ... </wsman:ResumptionContext> ?
    ...
  </wse:SubscribeResponse>
</s:Body>
```

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

s:Body/wse:SubscriptionResponse/wsman:Resumed

This required element MUST appear if the event source recognized and processed the wsman:ResumeAt extension element.

s:Body/wse:SubscriptionResponse/wsman:ResumptionContext

This required element MUST contain XML data that the event source can interpret in a potential future Subscribe request, as described above.

If wsman:Resume/@AllowResumption is requested in a Subscribe message and the event source accepts the subscription request, all event messages sent by the event source on that subscription MUST contain a wsman:ResumptionContext SOAP header element that contains the updated resumption context.

7. Security Considerations

7.1 Message security

In general, management operations and responses should be protected against attacks such as snooping, interception, replay, and in-flight modification. 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.

There are three primary approaches to addressing the requirements for confidentiality, message integrity, and user authentication:

No message security at all

This approach is appropriate in some device scenarios, but is NOT RECOMMENDED.

Transport-based message security

This approach leverages the ubiquitous support for HTTPS to provide confidentiality and message integrity. User authentication may be layered on top by using WS-Security username/password tokens, or by relying on TLS- or HTTP-based authentication mechanisms.

SOAP message security

This approach provides the most flexibility and control. Using the mechanisms in WS-Security and related specifications allows the use of a variety of authentication tokens and federation topologies. This is the RECOMMENDED approach.

Each implementation of this specification will select the option(s) that best apply to its unique circumstances. The specific security requirements placed on clients should be expressed in metadata. Because of the advantages of end-to-end security, the use of SOAP message security and the WS-* security specifications is RECOMMENDED.

The general security model for management operations follows the model outlined for Web services in general. That is, messages provide security tokens and prove them appropriately, which establishes a set of claims for the Manager. Resources have a set of requirement claims necessary to perform a specific action on a specific resource instance. If the requisite claims are provided, and trusted based on the issuer, then the request is authorized. Otherwise the request is not authorized.

The following sections provide additional details on message protection (confidentiality and integrity), authentication, and authorization.

7.1.1 Confidentiality

Most management operations involve passing information that should not be available to unauthorized users. For that reason, all messages SHOULD be protected. Specifically, at least the SOAP body SHOULD be encrypted.

It is RECOMMENDED that a SOAP message encryption protocol such as that specified in WS-Security be used to ensure end-to-end protection of the message content. Alternatively, a secure transport protocol such as HTTPS MAY be used to ensure point-to-point protection of the message content.

7.1.2 Integrity

Protecting message integrity against man-in-the-middle attacks is critical both to prevent the exposure of confidential information and to ensure that commands are not modified or misdirected. To ensure message integrity, all messages SHOULD be signed. Specifically, at least the SOAP body, the wsa:Action header, the wsa:To header, the wsman:ResourceURI header, the wsa:ReplyTo header, and any wsman:Key headers SHOULD be signed.

It is RECOMMENDED that a SOAP message integrity protocol such as WS-Security be used to ensure end-to-end integrity of the message content. Alternatively, a secure transport protocol such as HTTPS MAY be used to ensure point-to-point integrity of the message content.

7.1.3 Authentication

Allowing unauthorized access to management operations allows an attacker to take full control of a System. For this reason, before accepting any requests from a Manager, an Agent SHOULD require the Manager to authenticate itself.

It is RECOMMENDED that mutual authentication be established prior to issuing management commands.

It is RECOMMENDED that Agents allow Managers to authenticate themselves by using the mechanisms defined in WS-Security and related profiles and specifications. If a secure transport protocol such as HTTPS is being used for message confidentiality and integrity, an Agent MAY allow Managers to authenticate themselves by passing an unencrypted WS-Security UsernameToken containing a username and password verifiable by the Agent over the HTTPS connection.

Alternatively, an Agent may allow Managers to authenticate themselves using transport-level mechanisms, including:

- Using HTTP Basic Authentication to pass a username and password verifiable by the SERVICE over an HTTPS connection. The HTTPS connection is itself encrypted, so a plaintext user name and password is secure.
- Using HTTP Digest Authentication to perform a challenge-response authentication sequence over an HTTPS connection.
- Using a client certificate over an HTTPS connection.
- Using Kerberos over an HTTP connection or other transport.

7.1.4 Authorization

Access control within the WMX security model is based on the following principles:

1. Management operations are sent to a Resource of a specific Resource Type within a System.
2. Access control is determined based on the following parameters:
    a. The authenticated identity of the user making the request
    b. The System
    c. The Resource Type
    d. The Resource
    e. The wsa:Action; that is, the operation type

3. An implementation may determine access control based on a subset of these parameters.

Once the requestor’s identity has been established it can be used to determine additional claims; alternatively, those claims could be provided as part of the request. Note that claims can be delegated, subject to policy restrictions in the System. When multiple exchanges occur, the Manager MAY establish a security context using HTTPS or WS-SecureConversation.

If non-repudiation is a requirement then SOAP message security with client signatures (using appropriate tokens) is RECOMMENDED.

7.2 Event Delivery Security

Asynchronous event delivery has special security requirements because the event message from the Agent is not sent in reply to a request from a Manager. For synchronous event delivery (Pull mode; see section 6.3 above), no special processing beyond that for any other management request and response is necessary.

It is RECOMMENDED that the Agent sign each event message that it delivers asynchronously, using a key that may be verified by the recipient of each event message. For example, the Agent may sign each event using a certificate issued by a CA trusted by all recipients of the event.

When using either the Push or Batched delivery modes, it is RECOMMENDED that the Agent establish a security context using HTTPS or WS-SecureConversation to ensure the authenticity, integrity, and confidentiality of event messages.

The multicast nature of the Trap delivery mode prevents the establishment of a point-to-point or end-to-end security context; however, the Agent SHOULD still sign each Trap event message to ensure integrity and authenticity. If a globally shared secret is available, the Agent SHOULD encrypt each Trap event message using that secret.

8. Acknowledgements

This specification has been developed as a result of joint work with many individuals and teams, including:

Don Box, Microsoft
Josh Cohen, Microsoft
David Filani, Intel
Omri Gazitt, Microsoft
Frank Gorishek, AMD
Arvind Kumar, Intel
Brad Lovering, Microsoft
Sasha Nosov, Microsoft
Jeffrey Schlimmer, Microsoft
Tom Slaight, Intel
Marvin Theimer, Microsoft
Dave Tobias, AMD
John Tollefsrud, Sun
Anders Vinberg, Microsoft
Doug Walter, Microsoft

9. 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.

Appendix I. Profile

This appendix contains the profile of the base Web services specifications used by WS-Management.

Appendix I.1 URI

R0001: An Agent MAY fail to process any URI with more than MAX_URI_SIZE octets. In this case, the SERVICE MUST return a wsman:UriLimit fault.

R0002: An Agent SHOULD NOT generate a URI with more than MAX_URI_SIZE octets.

The constant MAX_URI_SIZE is 2,048 octets.

Appendix I.2 UDP

R0003: An Agent SHOULD NOT send a SOAP ENVELOPE that has more octets than the MTU over UDP.

To improve reliability, a SERVICE should minimize the size of SOAP ENVELOPEs sent over UDP. If a SOAP ENVELOPE is larger than an MTU, the underlying IP network stacks may fragment and reassemble the UDP packet.

Appendix I.3 HTTP/HTTPS

A common binding for SOAP is HTTP. While WS-Management does not require the use of this binding, it is recommended that SERVICES support SOAP over HTTP as a baseline for interoperability.

R0004: An Agent that supports SOAP over HTTP MUST support transfer-coding = "chunked".

R0005: An Agent that supports SOAP over HTTP MUST at least support the SOAP HTTP Binding as specified in the Basic Profile 1.0.

R0006: An Agent that supports SOAP over HTTP 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/).

R0007: An Agent that supports SOAP over HTTP 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).

R0008: An Agent that supports SOAP over HTTP MUST at least support Request Message SOAP Envelopes and one-way SOAP Envelopes that are delivered using HTTP POST.

R0009: An Agent that supports SOAP over HTTP SHOULD at least support the SOAP HTTP Binding in R0013 using HTTPS as specified in [HTTPS].

Appendix I.4 XML Encoding

R0010: Any REQUEST MESSAGE MAY be encoded using either UNICODE 3.0 (UTF-16) or UTF-8 encoding. An Agent 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 interoperation, it is trivial for a server to support both encodings, since R0011 places limits on the required character set.

R0011: An Agent 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:UnsupportedEncoding fault.

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.

R0012: 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.

The BOM indicates whether little-endian or big-endian encoding is in force. It is trivial for an implementation to simply swap adjacent octets in each character to the native form before processing the message.

Appendix I.5 SOAP Envelope

R0013: An Agent MUST at least receive and send SOAP 1.2 SOAP Envelopes.

R0014: If a SERVICE does not support attachments, it MAY reject a SOAP Envelope with more than MAX_ENVELOPE_SIZE octets. Similarly, it MAY fault any operation that would require a single reply exceeding MAX_ENVELOPE_SIZE octets. In this case, the SERVICE MUST return a wsman:EnvelopeLimit fault.

The constant MAX_ENVELOPE_SIZE is 32,767 octets.

Appendix I.6 Attachments

R0015: If a SERVICE supports attachments, the SERVICE MUST support the Abstract Transmission Optimization Feature.

R0016: If a SERVICE supports attachments, the SERVICE MUST support the Optimized MIME Multipart Serialization Feature.

R0017: If a SERVICE that supports SOAP over HTTP supports attachments, the SERVICE MUST support the HTTP Transmission Optimization Feature.

Appendix I.7 WS-Addressing

R0018: All messages MUST include a wsa:MessageID header element. The value of this element MUST be a valid URI using the uuid: scheme.

WS-Addressing allows the use of any URI as the message identifier. In practice, however, there is no need to support any format other than uuid because message identifiers carry no semantic information beyond uniqueness.

R0019: An Agent SHOULD reject all messages that do not have a Message Information Header representing the [action] property.

R0020: An Agent MUST include a Message Information Header representing the [action] property in each message the SERVICE generates.

WS-Addressing requires messages to contain a Message Information Header representing the [action] property; a SERVICE is not required to support processing messages that do not contain such a Message Information Header.

R0021: An Agent MUST allow requests to use the anonymous URI, http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous, as the wsa:ReplyTo address. In this case the reply message MUST be sent on the same transport channel as the request was received on.

R0022: An Agent MAY reject an HTTP Request Message SOAP ENVELOPE if the [address] of the [reply endpoint] is not "http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous".

The SOAP HTTP Binding requires the response message to be transmitted as the HTTP Response for the corresponding request message.

R0023: For all resource endpoints, a SERVER MUST return a wsa:ActionNotSupported fault (defined in WS-Addressing) if a requested operation is not supported, or is not applicable to the addressed resource.

R0024: If a request is received with an unknown reference property value, a SERVER MUST return a wsa:DestinationUnreachable fault (defined in WS-Addressing).

R0025: If a request is received with a valid reference property value but the named resource is not available at that time, a SERVER MUST issue a wsa:EndpointUnavailable fault (defined in WS-Addressing).

Fault delivery as a form of error notification is useful in the dynamic, data-driven situations that are common in systems management. Most specifications leave fault delivery as optional (MAY), but some faults provide enough value that they should be mandatory for management scenarios.

Appendix II: Faults

The following faults are defined by this specification:

Subcode	***Description***	***Encoding***
EnvelopeLimit	A message that is larger than MAX_ENVELOPE_SIZE was received.	**[Code]** s:Sender **[Reason]** Envelope too large
InvalidBody	The request that was received had an invalid body.	**[Code]** s:Sender **[Reason]** Invalid SOAP message
InvalidResumptionContext	The subscription request contained an invalid resumption context. For instance, the context could be sufficiently stale that the event source cannot resume at the specified point.	**[Code]** s:Sender **[Reason]** The resumption context is invalid

Subcode	*Description*	*Encoding*
OperationTimeout	The requested operation could not be completed in the time given, and so was cancelled.	**[Code]** s:Receiver **[Reason]** Operation processing timeout exceeded
ResumptionNotSupported	The subscription request specified that the subscription be resumable, and the event source does not support subscription resumption.	**[Code]** s:Sender **[Reason]** This event source does not support resumable subscriptions
ResumptionTypeNotSupported	The subscription request contained a resumption type (earliest or context) that is not supported.	**[Code]** s:Sender **[Reason]** The requested resumption type is not supported
UnsupportedEncoding	A message with character encoding other than UTF-8 or UTF-16 was received.	**[Code]** s:Sender **[Reason]** Unsupported character encoding
UriLimit	A message that contained a URI that is larger than MAX_URI_SIZE was received.	**[Code]** s:Sender **[Reason]** URI too large

Appendix III: 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/2004/10/management"
    xmlns:tns="http://schemas.xmlsoap.org/ws/2004/10/management"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsen="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    blockDefault="#all">

  <xs:import
      namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
  <xs:import
      namespace="http://schemas.xmlsoap.org/ws/2004/09/enumeration" />
```

```
<!-- General-use types -->
<xs:complexType name="KeyType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="Name" type="xs:token" />
      <xs:anyAttribute namespace="##other" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="FlagType">
  <!-- Allow for "soap:mustUnderstand" if the flag is a header -->
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>

<!-- Addressing elements -->
<xs:element name="System" type="xs:anyURI" />
<xs:element name="ResourceURI" type="xs:anyURI" />
<xs:element name="Key" type="tns:KeyType" />

<!-- Headers; note that all are defined so that soap:MustUnderstand may be
     added if desired. -->
<xs:element name="OperationTimeout">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="wsen:PositiveDurationType">
        <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="Locale">
  <xs:complexType>
    <xs:attribute name="lang" type="xs:language" />
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>
</xs:element>

<xs:element name="NoCache" type="tns:FlagType" />
<xs:element name="SummaryPermitted" type="tns:FlagType" />
<xs:element name="ReturnResource" type="tns:FlagType" />

<xs:element name="NewKeys">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Key" type="tns:KeyType" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>
</xs:element>

<!-- WS-Eventing: batched delivery mode -->
<xs:element name="MaxItems" type="xs:positiveInteger" />
<xs:element name="MaxTime" type="wsen:PositiveDurationType" />
<xs:element name="MaxCharacters" type="xs:positiveInteger" />
```

```
  <xs:complexType name="BatchedEventType">
    <xs:sequence>
      <xs:any minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="Action" type="xs:anyURI" />
  </xs:complexType>

  <xs:element name="Events">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Event" type="tns:BatchedEventType"
            maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- WS-Eventing: pull delivery mode -->
  <xs:element name="EnumerationContext" type="wsen:EnumerationContextType" />

  <!-- WS-Eventing: trap delivery mode -->
  <xs:element name="MulticastAddress" type="wsa:EndpointReferenceType" />

  <!-- WS-Eventing: resumable subscriptions -->
  <xs:complexType name="ResumptionContextType">
    <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:simpleType name="ResumeFromType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="earliest" />
      <xs:enumeration value="next" />
      <xs:enumeration value="context" />
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="ResumeAt">
    <xs:complexType>
      <xs:complexContent mixed="true">
        <xs:extension base="tns:ResumptionContextType">
          <xs:attribute name="From" type="tns:ResumeFromType" />
          <xs:attribute name="AllowResumption" type="xs:boolean" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="Resumed" type="tns:FlagType" />
  <xs:element name="ResumptionContext" type="tns:ResumptionContextType" />
</xs:schema>
```