

# Remote Shell Web Services Protocol

**July 2006, Version 1.0, Beta**

## Authors

Alexander Nosov, Microsoft  
Brian Reistad, Microsoft  
Johannes Helander, Microsoft  
Raymond McCollum, Microsoft (Editor)  
Steve Menzies, Microsoft  
Vishwa Kumbalimutt, Microsoft

## Copyright Notice

© 2004-2006 [Microsoft Corporation](#). All rights reserved.

Microsoft Corporation ("Microsoft") hereby grants you permission to copy and display the Remote Shell Web Services Profile, which includes its associated WSDL and Schema files and any other associated metadata (the "Specification"), in any medium without fee or royalty, provided that you agree to the below terms and conditions and that you include the following on ALL copies of the Specification, or portions thereof, that you make:

1. A link or URL to the Specification posted at Microsoft's websites
2. All copyright notices as shown on the Specification.

EXCEPT FOR THE COPYRIGHT LICENSE GRANTED ABOVE, MICROSOFT DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY, INCLUDING PATENTS, IT OWNS OR CONTROLS.

THE SPECIFICATION IS PROVIDED "AS IS," AND MICROSOFT MAKES 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.

YOU AGREE THAT MICROSOFT 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 SPECIFICATION.

The name and trademarks of Microsoft may NOT be used in any manner, including advertising or publicity pertaining to the Specification or its contents without the specific, written prior permission of Microsoft.

## Abstract

This specification describes a set of extensions to the standard WS-Management protocol for accessing common command shell processors.

## Status

This edition corresponds to the beta implementation available in Windows *Vista*. It should be considered preliminary and subject to change.

## Table Of Contents

<b>1.0</b>	<b>Introduction</b>	<b>4</b>
<b>1.1</b>	<b>WS-Management</b>	<b>4</b>
<b>1.2</b>	<b>Message Model</b>	<b>4</b>
<b>1.3</b>	<b>Requirements</b>	<b>5</b>
<b>2.0</b>	<b>Notations, Terminology, and General Notes</b>	<b>6</b>
<b>2.1</b>	<b>Notational Conventions</b>	<b>6</b>
<b>2.2</b>	<b>XML Namespaces</b>	<b>6</b>
<b>2.3</b>	<b>Compliance</b>	<b>7</b>
<b>2.4</b>	<b>Character Encoding</b>	<b>7</b>
<b>2.5</b>	<b>SOAP Header <i>mustUnderstand</i> Usage</b>	<b>7</b>
<b>3.0</b>	<b>Shell Messages</b>	<b>7</b>
<b>3.1</b>	<b>wxf:Create</b>	<b>7</b>
3.1.1	Additional Restrictions	11
3.1.2	Faults	11
3.1.3	wxf:ResourceCreated	11
<b>3.2</b>	<b>wxf:Get</b>	<b>12</b>
<b>3.3</b>	<b>wsen:Enumerate</b>	<b>14</b>
<b>3.4</b>	<b>wxf:Put</b>	<b>14</b>
<b>3.5</b>	<b>wxf&gt;Delete</b>	<b>15</b>
3.2.1	DeleteResponse	15
3.2.2	Example	15
<b>3.6</b>	<b>rsp:Command</b>	<b>17</b>
3.6.1	Rules For Use	18
3.6.2	Faults	18
3.6.3	rsp:CommandResponse	19

3.6.4	Examples	20
<b>3.7</b>	<b>Stream Encoding</b>	<b>21</b>
3.7.1	Encoded Data	23
3.7.2	Stream Naming	23
3.7.3	Logical Records in Streams	23
3.7.4	Stream Binding	23
3.7.5	Stream Ordering And Size	24
<b>3.8</b>	<b>rsp:Send</b>	<b>25</b>
3.8.1	Faults	26
3.8.2	Rules for Use	26
3.8.3	rsp:SendResponse	27
3.8.4	Examples	28
<b>3.9</b>	<b>rsp:Receive</b>	<b>30</b>
3.9.5	Faults	31
3.9.6	rsp:ReceiveResponse	31
3.9.7	Rules For Use	33
3.9.8	Examples	34
<b>3.10</b>	<b>rsp:Signal</b>	<b>36</b>
3.10.1	Faults	37
3.10.2	rsp:SignalResponse	38
3.10.3	Examples	38
<b>3.11</b>	<b>WS-Eventing</b>	<b>40</b>
<b>4.0</b>	<b><i>Security Considerations</i></b>	<b>40</b>
<b>4.1</b>	<b>WS-Management Security</b>	<b>40</b>
<b>4.2</b>	<b>Authorization</b>	<b>40</b>
<b>5.0</b>	<b><i>Faults</i></b>	<b>40</b>
<b>5.1</b>	<b>wxf:InvalidRepresentation</b>	<b>40</b>
<b>5.2</b>	<b>wsman:QuotaLimit</b>	<b>41</b>
<b>5.3</b>	<b>rsp:CommandFault</b>	<b>42</b>
<b>5.4</b>	<b>rsp:SendFault</b>	<b>43</b>
<b>5.5</b>	<b>rsp:ReceiveFault</b>	<b>44</b>
<b>5.6</b>	<b>rsp:SignalFault</b>	<b>45</b>
<b>6.0</b>	<b><i>Standard Shell Processor in Microsoft Windows®</i></b>	<b>45</b>
<b>6.1</b>	<b>Introduction</b>	<b>45</b>
<b>6.2</b>	<b>ResourceURI and ShellId</b>	<b>46</b>
<b>6.3</b>	<b>wxf:Create Usage</b>	<b>46</b>
<b>6.4</b>	<b>Reliability Issues</b>	<b>47</b>
<b>7.0</b>	<b><i>The Schema and WSDL Files</i></b>	<b>47</b>
<b>8.0</b>	<b><i>References</i></b>	<b>47</b>

# 1.0 Introduction

The use of interactive command shell processors by all kinds of computer users is ubiquitous. Many operating systems support more than one such shell, and there is a broad need for a general-purpose Web Service paradigm which can expose command shell behavior in a platform-neutral manner. The supported range should include everything from complex interactive shells such as BASH down to simple dedicated C applications which only accept one or two switches.

This specification defines a simple SOAP-based protocol for execution of command lines in arbitrary shell processors and thus models a general purpose shell processor. Because almost all shells are based on precisely the same real-world paradigm, it is fairly easy to define a simple protocol which can act as an effective carrier for most of the shell processors in heavy use today on the most popular operating systems.

It is important to note that the scripting language or syntax expected by the shell processor is independent of the paradigm itself. Most shells accept a command terminated by a newline character, the shell processes that command, regardless of the language or syntax, and the output of that command is redirected to the shell console itself for further processing by more commands.

The protocol mimics the operations an interactive user would experience: opening the shell, running one or more commands in the form of simple text-based command-lines, possibly feeding input streams, examining the output streams, running new commands, and finally closing the shell. The protocol acts more as a common I/O mechanism but does not get involved in the processing of the commands or interpreting the I/O streams, which are handled entirely by the shell processor which is serviced by the protocol implementation.

## 1.1 WS-Management

This protocol layers over the WS-Management protocol [[WS-Management](#)].

Any requirements established for that protocol apply to this protocol as well, and the standard control headers for WS-Management apply also to this specification.

This specification defines several new messages and faults, which are considered to be direct extensions to the WS-Management specification.

## 1.2 Message Model

This protocol is an extension to WS-Management, and uses some of the messaging patterns from that specification. (In the following discussion, the XML namespace prefixes are those defined in the table in 2.2.)

The **wxf:Create** operation is used to "open" or "create" a shell instance. The wsman:ResourceURI in the WS-Management addressing model identifies the type of shell to create. A new shell instance is then created and a wsa:EndpointReference to the processor is returned to the client in a **wxf:ResourceCreated** message.

To delete or close the shell later on, **wxf:Delete** is used with this wsa:EndpointReference. Executing a **wxf:Get** on this same EPR returns a resource instance which represents the shell instance that is open and acts as its instrumentation.

To execute a command, a **rsp:Command** message is sent to the shell instance (via its specific wsa:EndpointReference), and the output from the command is read by issuing **rsp:Receive** messages and examining the responses, which will contain the output. After the last batch of outputs has been received, the client issues a **rsp:Signal** to finalize the command execution and prepare the shell for the next command.

There is no asynchronous transmission of the output from the service to the client. Rather, the client must explicitly retrieve output by synchronously "pulling" the output in a manner similar to that defined for wsen:Pull using **rsp:Receive** message.

In cases where the command needs one or more input streams, **rsp:Send** can be used to "pipe" input to the command. The **rsp:Send** and **rsp:Receive** messages may be executed concurrently relative to each other.

The **rsp:Signal** message may be sent asynchronously at any time to interrupt or otherwise modify any processing that is currently in progress and is used to terminate the most recently executed command. This final termination step occurs consistently whether the command is terminated early, or whether it is run to completion and all output has been received.

Most implementations will support **wsen:Enumerate** to list outstanding shell instances, although the EPR of the collection may or may not be the same one used for the **wxf:Create** message. The list may be scoped depending on access rights (i.e., list the shells created by you as a user as opposed to all shells created by anyone).

This version of the specification places simple restrictions on concurrency. Only one command may execute at a time and must run to completion before a new command can be submitted. There are restrictions on the concurrency of the various messages, although it is typical for the **rsp:Send** and **rsp:Receive** to execute concurrently, as already described. These restrictions simplify the implementation and make it simple to build a client-side stack which will work consistently with a variety of shell processors.

Because commands may require one or more input streams and may emit one or more output streams, this specification contains an explicit model for naming and encoding the streams.

## 1.3 Requirements

This specification intends to meet the following requirements:

- Support any of the widely used shell processors on most operating systems, as well as simple dedicated processors such as single C applications which take only one command line.
- Support multiple streams of input and output in a consistent manner.
- To allow users to asynchronously halt or modify processing of the currently running command, or to send other control codes to a running shell, such as pause and resume.
- To allow users to asynchronously deliver input stream data (such as the response to a prompt).
- To allow users to read output streams asynchronously to any possible input streams.

- To allow users to open more than one instance of a shell, where supported by the underlying system
- To establish a simple instrumentation model for active shell instances

## 2.0 Notations, Terminology, and General Notes

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.microsoft.com/wbem/wsman/1/windows/shell
```

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)
rsp	<a href="http://schemas.microsoft.com/wbem/wsman/1/windows/shell">http://schemas.microsoft.com/wbem/wsman/1/windows/shell</a>	This specification
wsman	<a href="http://schemas.dmtf.org/wbem/wsman/1/wsman">http://schemas.dmtf.org/wbem/wsman/1/wsman</a>	WS-Management [ <a href="#">WS-Management</a> ]

s	<a href="http://www.w3.org/2003/05/soap-envelope">http://www.w3.org/2003/05/soap-envelope</a>	SOAP 1.2 [ <a href="#">SOAP 1.2</a> ]
s11	<a href="http://schemas.xmlsoap.org/soap/envelope/">http://schemas.xmlsoap.org/soap/envelope/</a>	SOAP 1.1 [ <a href="#">SOAP 1.1</a> ]
wsa	<a href="http://schemas.xmlsoap.org/ws/2004/08/addressing">http://schemas.xmlsoap.org/ws/2004/08/addressing</a>	WS-Addressing [ <a href="#">WS-Addressing</a> ]
xs	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>	XML Schema [ <a href="#">Part 1, 2</a> ]
wsdl	<a href="http://schemas.xmlsoap.org/wsdl/">http://schemas.xmlsoap.org/wsdl/</a>	WSDL/1.1 [ <a href="#">WSDL 1.1</a> ]
wxf	<a href="http://schemas.xmlsoap.org/ws/2004/09/transfer">http://schemas.xmlsoap.org/ws/2004/09/transfer</a>	WS-Transfer [ <a href="#">WS-Transfer</a> ]

Note that the URI value

<http://schemas.microsoft.com/wbem/wsmn/1/windows/shell/cmd> refers to a specific ResourceURI within Microsoft Windows which acts as the default shell processor in Windows Vista. It is not an XML namespace URI, however.

## 2.3 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.

## 2.4 Character Encoding

This specification does not address character encoding requirements for specific shell processors. Some processors may allow either ANSI or UNICODE input and may emit output in either or both forms, depending on the underlying engine. This protocol passes the character encodings literally to the shell processor, both with regard to launching commands via `rsp:Command` and with regard to I/O streaming via `rsp:Send` and `rsp:Receive`. The client must be aware of the requirements of the shell and any commands which it may execute. This is in fact the expected requirement when using shells locally outside of any remoting protocols.

## 2.5 SOAP Header *mustUnderstand* Usage

In practice, all headers from other specifications referenced in this specification should follow the rules established in their owning specifications.

# 3.0 Shell Messages

## 3.1 wxf:Create

To open a new shell, a WS-Transfer Create message MUST be sent using the model and restrictions established by WS-Management.

No new requirements are established by this specification, other than the fact that the EPR containing the wsman:ResourceURI contains the URI of a resource which can create new instances of the specific type of shell referenced by the URI. In essence, this EPR (with its wsman:ResourceURI) is a "factory" for instances of the shell. An implementation may expose more than one such EPR, one for each "type" of shell. There is no requirement that the resource support more than one instance of a shell for a given type, although this is typical.

The s:Body of the message contains startup parameters for the shell. There is a standardized s:Body which is extensible so that different shells can expose their own initialization parameters as required, and this is illustrated below:

```
(1)  <s:Envelope ...>
(2)    <s:Header ...>
(3)      <wsa:Action>
(4)        http://schemas.xmlsoap.org/ws/2004/09/transfer/Create
(5)      </wsa:Action>
(6)      ...other WS-Addressing & WS-Management headers...
(7)      <wsman:ResourceURI> Shell Resource URI </wsman:ResourceURI>
(8)
(9)    </s:Header>
(10)   <s:Body ...>
(11)     <rsp:Shell>
(12)       <rsp:WorkingDirectory> xs:string </rsp:WorkingDirectory> ?
(13)       <rsp:Environment>
(14)         <rsp:Variable Name="xs:string">
(15)           xs:string
(16)         </rsp:Variable> *
(17)       </rsp:Environment> ?
(18)       <rsp:Lifetime>
(19)         xs:duration
(20)       </rsp:Lifetime> ?
(21)
(22)       <rsp:InputStreams> xs:token list </rsp:InputStreams>
(23)       <rsp:OutputStreams> xs:token list </rsp:OutputStreams> ?
(24)
(25)       ...additional shell-specific initialization XML...
(26)
(27)     </rsp:Shell> ?
(28)   </s:Body>
(29) </s:Envelope>
```

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

/s:Envelope/s:Header/wsa:Action (line 4)

This required element MUST contain the value `http://schemas.xmlsoap.org/ws/2004/09/transfer/Create`, as required by WS-Management.

`/s:Envelope/s:Header/wsman:ResourceURI` (line 7)

This is part of the WS-Management default EPR address, and must contain the `wsman:ResourceURI` which represents the shell factory and which can create an instance of the shell for subsequent use. The `wsman:SelectorSet` may also be required as part of the EPR, depending on the shell processor.

`/s:Envelope/s:Body/rsp:Shell` (lines 11-26)

This optional element contains any information required to properly initialize the targeted shell. This is typically used to set up the initial execution environment for the shell. The element has several predefined elements and ends with an open content model so additional resource-specific initialization can be provided using a single XML element of arbitrary structure from another XML namespace.

`/s:Envelope/s:Body/rsp:Shell/rsp:WorkingDirectory` (line 12)

An `xs:string` value containing the starting directory that the shell should use for initialization. The syntax is shell-specific. If there is no directory system established for the shell processor, this may be omitted or set to an empty string value, and a default directory will be used by the shell processor.

`/s:Envelope/s:Body/rsp:Shell/rsp:Environment` (lines 13-15)

The starting set of environment variables that the shell will use. If omitted, then any default environment is indicated. Each environment variable is individually defined.

`/s:Envelope/s:Body/rsp:Shell/rsp:Environment/rsp:Variable` (line 14)

One or more environment variables which constitute the starting environment for the shell instance when it is created. The content of this element is shell-specific and may contain the new literal value of the specified variable or may contain instructions or formulae for computing the effective value. In some systems, for example, the variable may be defined in terms of other environment variables or predefined values using system-specific string operators. Note that XML reserved characters, such as `<`, `>`, or quotes may be required for the literal values of some variables, and these must be escaped using normal XML escaping rules.

`/s:Envelope/s:Body/rsp:Shell/rsp:Environment/rsp:Variable/@Name` (line 14)

The environment variable name.

`/s:Envelope/s:Body/rsp:Shell/rsp:Lifetime`(lines 16-18)

An OPTIONAL quota setting to control the total shell lifetime. The shell MAY become invalid once this time limit is reached and no other commands may be issued.

`/s:Envelope/s:Body/rsp:Shell/rsp:IdleTimeout`(line 19)

This represents an optional idle timeout for the shell. The service is permitted to close and terminate the shell instance if it is idle for this much time. If the shell is reused within this time limit, then the countdown timer is reset once the command sequence is completed.

`/s:Envelope/s:Body/rsp:Shell/rsp:InputStreams` (line 21)

This is a simple token list of all input streams the client will be using during execution. The client must know what input streams names are supported by the shell processor and assert the ones of interest. There is no requirement that the client in fact make use of all the available streams supported by the shell processor or that they are all identified. The client must not later attempt to deliver a named stream which is not

listed, however. For example, if a shell processor supports a standard input stream named "stdin", but the client knows that this stream will not be used during the session because none of the commands that will be used require input streams, then the client can specify an empty `rsp:InputStreams` element or omit it altogether. However, if the client anticipates that the `stdin` stream may be used, it **MUST** include the stream name in this list. If it is specified, there is no requirement that the client actually use the stream during the session, only that it contractually obligates the service to recognize that stream if it is used. The service can thus detect initially whether it can intelligently interoperate with the client or not. Some shell processors may not support input streaming at all, in which case this element is left empty or omitted.

/s:Envelope/s:Body/rsp:Shell/rsp:OutputStreams (line 22)

This is a simple token list of all output streams expected by the client. The client must know what output streams names are supported by the shell processor and assert the ones of interest. There is no requirement that the client in fact make use of all the available output streams, but only that it will not try to read a stream which is not listed. For example, if a shell processor supports a standard streams named "stdout" and "stderr", but the client only needs "stdout" during the session, then the client can simply list "stdout" as the sole stream of interest. If a stream is specified, there is no requirement that the client actually use the stream during the session, only that it contractually obligates the service to output that stream during execution if the command writes to it. The service can thus detect initially whether it can intelligently interoperate with the client or not. The service may require that certain output streams be used in every case, and if these are missing a fault is issued with one of the fault detail codes listed in 5.1.

In addition to the above standardized initialization parameters, the open content model for the `rsp:Shell` body allows individual shells to have any additional XML content required for initialization, as shown on lines 40-43:

```
(30)   <s:Body ...>
(31)     <rsp:Shell>
(32)       <rsp:WorkingDirectory> xs:string </rsp:WorkingDirectory>
(33)       <rsp:Environment> ...
(34)     </rsp:Environment>
(35)       <rsp:IdleTimeout> xs:duration </rsp:IdleTimeout>
(36)       <rsp:InputStreams> ...
(37)     </rsp:InputStreams>
(38)
(39)       <rsp:OutputStreams> ...
(40)     </rsp:OutputStreams>
(41)       <ShellInit xmlns="http://schemas.example.com/SampleShellInit">
(42)         <InitValue> xyz </InitValue>
(43)         <MaxWidgets> 123 </MaxWidgets>
(44)       </ShellInit>
(45)     </rsp:Shell>
(46)   </s:Body>
```

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

If shell-specific initialization XML follows the `rsp:OutputStreams` element, the service implementation **MUST** comply with any instructions or content in that XML, treating it as if a `mustUnderstand="true"` were applied to it (Note that the SOAP `mustUnderstand` attribute is not usable for `s:Body` content, only for `s:Header` content.). The service **MUST NOT** ignore this additional initialization content and **SHOULD** return a `wxf:InvalidRepresentation` fault if it is not recognized or supported, using the following detail code:

```
http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidExtension.
```

Of course, the additional shell-specific XML must be properly qualified by an XML namespace.

### 3.1.1 Additional Restrictions

While WS-Management allows the `wsman:Locale` message to appear on any message, in general it only makes sense to establish the `wsman:Locale` when the shell is created using `wxf:Create`, as changing locales during execution would be highly unusual and complex. In general, `wsman:Locale` is ignored for all messages except `wxf:Create`.

### 3.1.2 Faults

The faults are as defined for WS-Management, but additional detail codes may be returned. See 5.0.

### 3.1.3 wxf:ResourceCreated

Upon successful processing of an `wxf:Create` message, the service is expected to create a shell instance and return a reference to it as a `wsa:EndpointReference`, as specified for `wxf:ResourceCreated` and WS-Management:

```
(1)  ...
(2)  <s:Body>
(3)    <wxf:ResourceCreated
(4)      <wsa:Address>
(5)        Transport level address of shell processor
(6)      </wsa:Address>
(7)      <wsa:ReferenceParameters>
(8)        <wsman:ResourceURI> URI to shell processor </w:ResourceURI>
(9)        <wsman:SelectorSet>
(10)          <wsman:Selector Name="selector"></wsman:Selector>
(11)        </wsman:SelectorSet>
(12)      </wsa:ReferenceParameters>
(13)    </wxf:ResourceCreated>
(14) </s:Body>
```

As shown, the `wsa:EndpointReference` encapsulated within the `wxf:ResourceCreated` contains a reference to the newly created shell instance. This address is used in all subsequent messages to the shell instance, i.e., `wxf:Delete`, `rsp:Command`, `rsp:Signal`, `rsp:Send`, and `rsp:Receive`.

The returned EPR must conform to WS-Management, and thus MUST have at least a `wsman:ResourceURI`, and optionally a `wsman:SelectorSet` containing one or more `wsman:Selector` values which will reference the newly created shell instance.

The client must extract this new EPR and use it in all subsequent messages, as it refers to the shell instance that was just created.

An example of a `wxf:ResourceCreated` with such an EPR follows:

```
(15) ...
(16) <s:Body>
(17)   <wxf:ResourceCreated>
(18)     <wsa:Address>http:
(19) http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
(20)   </wsa:Address>
(21)     <wsa:ReferenceParameters>
(22)       <wsman:ResourceURI>
(23) http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd
(24)     </wsman:ResourceURI>
(25)     <wsman:SelectorSet>
(26)       <wsman:Selector Name="ShellId">
(27)         uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6
(28)       </wsman:Selector>
(29)     </wsman:SelectorSet>
(30)   </wsa:ReferenceParameters>
(31) </wxf:ResourceCreated>
(32) </s:Body>
```

### 3.2 wxf:Get

The `wxf:Get` operation in WS-Management is optionally defined for this specification. Its purpose would be to return resource information about the shell instance that was created by the `wxf:Create` operation. This is typically used to expose instrumentation. The EPR used in the `wxf:Get` is the same one returned by the `wxf:ResourceCreated` message described in 3.1 and the same one which is compatible with `wxf:Delete` to close the shell instance.

This specification makes no statements as to the authorization model for the result of the enumeration. Depending on the access rights of the client, the `wxf:Get` may be able to return instances (for administrators) of any shells created by anyone, or (for lower-level users) just the instance of a shell created by the user who is doing the `wxf:Get`.

An example `wxf:Get` follows:

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

```

(2)   <s:Header ...>
(3)     <wsa:Action>
(4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
(5)     </wsa:Action>
(6)   <wsa:MessageID>
(7)     uuid:e7c5726b-de29-4313-b4d4-b3425b200453
(8)   </wsa:MessageID>
(9)   <wsa:To>http://www.example.org/wsman</wsa:To>
(10)  <wsman:ResourceURI>
(11)  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd
(12)  </wsman:ResourceURI>
(13)  <wsman:SelectorSet>
(14)    <wsman:Selector Name="ShellId">
(15)      uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6
(16)    </wsman:Selector>
(17)  </wsman:SelectorSet>
(18)  <wsa:MessageId>
(19) </s:Header>
(20) <s:Body />
(21) </s:Envelope>

```

By default, the wxf:GetResponse message should return an XML infoset which corresponds to the current state of the shell as regards its working directory, environment variables, and so on:

```

(1)   <s:Envelope ...>
(2)   <s:Header ...>
(3)     <wsa:Action>
(4)     <wsa:Action>
(5)       http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(6)     </wsa:Action>
(7)   </s:Header>
(8)   <s:Body>
(9)     <rsp:Shell
(10)
xmlns:rsp="http://schemas.microsoft.com/wbem/wsman/1/windows/shell">
(11)       <rsp:ShellId>
(12)         uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6
(13)       <rsp:ShellId>
(14)       <rsp:Environment>
(15)         <rsp:Variable Name="foo">bar</rsp:Variable>
(16)         <rsp:Variable Name="Zappa">Rock'n'Roll</rsp:Variable>
(17)       </rsp:Environment>

```

```

(18)     <rsp:WorkingDirectory>
(19)         %windir%
(20)     </rsp:WorkingDirectory>
(21)     <rsp:InputStreams>stdin</rsp:InputStreams>
(22)     <rsp:OutputStreams>stdout stderr</rsp:OutputStreams>
(23)         ...shell-specific extensions...
(24)     </rs:Shell>
(25) </s:Body>
(26) </s:Envelope>

```

Note that the service adds the ShellId element to the schema which is otherwise the same as the one used in the wxf:CreateMessage. The ShellId is optional but can be used to identify the shell instance. The client can construct a valid EPR to the shell instance by extracting this value and using it as described in 3.1.3. Implementations which do not support a ShellId may omit this value.

Note also that shell-specific content may appear after the rsp:OutputStreams element.

### 3.3 wsen:Enumerate

The wsen:Enumerate message is defined by default to return any "live" instances of the shell processor. The wsman:ResourceURI is typically the same one used to create new shell instances using wxf:Create, but may be different. Executing a wsen:Enumerate (and subsequent wsen:Pull messages) against this URI will return a series of instances which represent open shell instances, using the same format and schema as recommended for wxf:GetResponse above in section 3.2 of this specification.

This specification makes no statements as to the authorization model for the result of the enumeration. Depending on the access rights of the client, the enumeration may return all instances (for administrators) of all shells created by anyone, or (for lower-level users) just the instances of shells created by the user doing the enumeration.

As described in 3.2, that the service adds the ShellId element to each instance of the rsp:Shell schema in the numeration. The ShellId is optional, but can be used to identify the shell instance. The client can build a valid EPR to the shell instance by extracting this value and using it as described in 3.1.3. Implementations which do not support a ShellId may omit this value.

### 3.4 wxf:Put

The use of wxf:Put is not defined for this specification.

Typically, the state of a shell is updated by executing commands within it, as opposed to trying to update it via a wxf:Put operation. Typical examples are setting environment variables. Rather than attempting to use a wxf:Put with the new values, the rsp:Command message could be executed with a shell-specific command to set the environment variables.

## 3.5 wxf:Delete

To close a shell, a wxf:Delete message is sent using the wsa:EndpointReference returned in the wxf:ResourceCreated when the shell was first created.

This may be sent asynchronously to any outstanding messages in progress to the specified shell, allowing the shell to be forcibly closed. Any commands in progress should be immediately terminated and all resources for the shell freed. The final result of any operations in progress are undefined and similar to forcibly terminating a shell processor in any other context outside of SOAP.

The Delete request message is of the form specified in WS-Management:

```
(1) <s:Envelope ...>
(2)   <s:Header ...>
(3)     <wsa:Action>
(4)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
(5)     </wsa:Action>
(6)     <wsa:To> Network address URL </wsa:To>
(7)     <wsman:ResourceURI> URI returned in wxf:Create </wsman:ResourceURI>
(8)     <wsman:SelectorSet>
(9)       Selectors returned in wxf:Create
(10)    </wsman:SelectorSet>
(11)    ...
(12)  </s:Header>
(13)  <s:Body />
(14) </s:Envelope>
```

The only requirement is that the EPR to the shell (lines 7-11) be the one returned in the original wxf:ResourceCreated. There are no specific additional requirements imposed by this specification.

All faults defined for wxf:Delete in WS-Management apply, and no new faults are defined.

### 3.2.1 DeleteResponse

Upon successful processing of a Delete request message, a processor is expected to return a DeleteResponse message, which MUST adhere to the form described in WS-Management.

This specification places no additional restrictions or requirements on the response.

### 3.2.2 Example

The following sequence illustrates a request to close a shell instance using wxf:Delete:

```
(1) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope '
(2)   xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(3)   xmlns:wsman='http://schemas.dmtf.org/wbem/wsman/1/wsman '
(4)   <wsa:Action>
(5)     http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
(6)   </wsa:Action>
(7)   <wsa:To> Network address URL </wsa:To>
(8)   <wsman:ResourceURI> URI returned in wxf:Create </wsman:ResourceURI>
(9)   <wsman:SelectorSet>
(10)    Selectors returned in wxf:Create
(11)  </wsman:SelectorSet>
(12) </s:Envelope>
```

```

(4)
(5)   <s:Header>
(6)     <wsa:Action>
(7)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete
(8)     </wsa:Action>
(9)     <wsa:MessageID>
(10)      uuid:e7c5726b-de29-4313-b4d4-b3425b200453
(11)    </wsa:MessageID>
(12)    <wsa:To>http://www.example.org/wsman</wsa:To>
(13)    <wsman:ResourceURI>
(14)      http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd
(15)    </wsman:ResourceURI>
(16)    <wsman:SelectorSet>
(17)      <wsman:Selector Name="ShellId">
(18)        uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6
(19)      </wsman:Selector>
(20)    </wsman:SelectorSet>
(21)  </s:Header>
(22)  <s:Body/>
(23) </s:Envelope>

```

Line 7 indicate this message is a wxf:Delete request and that the shell processor is expected to respond with a wxf:DeleteResponse message.

Lines (12-20) contains the endpoint reference from the original wxf:ResourceCreated message. This is the reference to the instance that will be deleted.

The response is as defined in WS-Management and contains no new information other than a wsa:RelatesTo to correlate the response with the original message.

```

(24) <s:Envelope
(25) xmlns:s='http://www.w3.org/2003/05/soap-envelope'
(26) xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'>
(27) <s:Header>
(28)   <wsa:Action>
(29)     http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse
(30)   </wsa:Action>
(31)   <wsa:MessageID>
(32)     uuid:e7c5726b-de29-4313-b4d4-b3425b243140
(33)   </wsa:MessageID>
(34)   <wsa:RelatesTo>
(35)     uuid:e7c5726b-de29-4313-b4d4-b3425b200839
(36)   </wsa:RelatesTo>
(37) </s:Header>
(38) <s:Body/>
(39) </s:Envelope>

```

Line 27 indicates this message is a DeleteResponse message and lines 32-34 correlate the response with the original request.

### 3.6 rsp:Command

To execute a command within a shell, the rsp:Command message is sent to the EPR of an existing shell instance. This EPR was obtained from a wxf:ResourceCreated message during shell instance creation.

This is a new message not defined within WS-Management, but adheres to all the requirements established by that specification. It is unique only in that it has a unique wsa:Action URI value, and the <s:Body> content is specific to the concept of submitting a command-line to the shell processor.

The rsp:Command operation is initiated by sending an rsp:Command request message to the shell processor. The request message MUST be of the following form:

```
(1)  <s:Envelope ...
(2)    xmlns:rsp="http://schemas.microsoft.com/wbem/wsman/1/windows/shell"
(3)    ...
(4)    >
(5)    <s:Header ...>
(6)      <wsa:Action>
(7)        http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Command
(8)      </wsa:Action>
(9)
(10)   ...include EPR of the targeted shell instance...
(11)   ...any other headers permitted by WS-Management...
(12)
(13)  </s:Header>
(14)  <s:Body ...>
(15)    <rsp:CommandLine ...>
(16)      <rsp:Command> xs:string </rsp:Command>
(17)      <rsp:Arguments> xs:string </rsp:Arguments> +
(18)    </rsp:CommandLine>
(19)  </s:Body>
(20) </s:Envelope>
```

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

/s:Envelope/s:Header/wsa:Action (line 7)

This required element MUST contain the value

http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Command

/s:Envelope/s:Body/rsp:CommandLine (line 15)

This is a REQUIRED wrapper element which contains the command line and other initialization parameters for the command.

/s:Envelope/s:Body/rsp:Command/rsp:Command (line 16)

This is a REQUIRED string which represents the command to be executed, without any arguments, which are separately encoded. The syntax is shell-specific.

/s:Envelope/s:Body/rsp:CommandLine/rsp:Arguments (line 17)

This OPTIONAL element contains a string value of a single command argument. If no arguments are required, this element may be omitted. If more than one argument is required then multiple elements are included. Since many command processors use characters which are XML reserved characters, it is important to observe correct XML reserved character escape sequences. This is particularly important because the < and > characters (reserved in XML) are often used for I/O redirection in shell implementations, so these must be replaced with their "escaped" XML equivalents **&lt;** or **&gt;**. Many shell processors require ANSI newline or carriage return characters to appear at the end of the sequence to properly begin execution. Such requirements are shell specific and not part of this protocol. In general, it is recommended that shell processors observe a lax processing model and accept an rsp:Arguments value, with or without special terminating whitespace characters such as newline characters.

Note that any other appropriate WS-Management headers may be used in conjunction with these definitions, such as wsman:OperationTimeout, and so on. However, wsman:Locale is ignored and only processed during wxf:Create as described in 3.1.1.

### 3.6.1 Rules For Use

In this version of the specification, only one command may be outstanding per instance of the shell processor. To achieve concurrent commands, more than one instance of the shell must be created.

Once the command is created and the CommandId value has been obtained from the rsp:CommandResponse, rsp:Send messages may be sent for that command containing input stream data, if the command requires input beyond the command line arguments. In addition rsp:Receive messages MUST be sent until all output streams end unless the command is to be terminated early using an rsp:Signal.

An rsp:Signal must be sent as the last message in all cases, whether the command is terminated early or run to completion.

An attempt to execute a new command while unread output streams still exist will result in a wsman:Concurrency fault.

### 3.6.2 Faults

Note that shell failure and any associated messages fall into two categories: protocol-level failures in which the command cannot be successfully delivered to the shell processor, and errors during the shell processing itself.

An invalid command message can result in an rsp:Command fault (see 5.3). This is a SOAP-level error, meaning the message cannot be processed well enough to deliver it to the shell.

Any errors which occur within the processing of the command by the shell processor are NOT considered SOAP faults, since the protocol has successfully delivered the instructions and the shell processor begins its work.

Instead, any such errors are considered part of the output stream of the shell, and are non-error conditions as regards the protocol. They are retrieved in the normal course of events by using `rsp:Receive` to get the output from the processor.

In other words, even if the command is invalid, refers to a non-existent executable program, or the command line is ill-formed, the protocol implementation will not know this, only the shell processor will know it, so a SOAP fault does NOT apply.

### 3.6.3 `rsp:CommandResponse`

Upon successful processing of a `Command` request message, a shell processor is expected to return a `rsp:CommandResponse` message, which MUST adhere to the following form and must adhere to other conventions defined in `WS-Management`:

```
(21) <s:Envelope ...>
(22)   <s:Header ...>
(23)     <wsa:Action>
(24)
http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandResponse
(25)     </wsa:Action>
(26)     ...other required headers such as wsa:RelatesTo...
(27)
(28)   </s:Header>
(29)   <s:Body ...>
(30)     <rsp:CommandResponse ...>
(31)       <rsp:CommandId>xs:anyURI</rsp:CommandId>
(32)     </rsp:CommandResponse>
(33)   </s:Body>
(34) </s:Envelope>
```

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

`/s:Envelope/s:Header/wsa:Action` (line 4)

This required element MUST contain the value

`http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandResponse`.

`/s:Envelope/s:Body/rsp:CommandResponse/rsp:CommandId` (line 11)

This is a full or relative URI that uniquely identifies the command that was just created and is used in subsequent messages to reference the input, output, and status for that particular command. This value MUST BE non-repeating within the scope and lifetime of the shell instance. It is not a requirement for this value to be globally unique such as a GUID, but GUIDs are typically used. The only requirement is that the value must not be reused within the lifetime of the current shell instance. A value based on an increasing, non-repeating integer may also be suitable.

### 3.6.4 Examples

The following examples illustrate execution of a command using `rsp:Command`.

```
(1) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope'  
(2)   xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'  
(3)   xmlns:wsman='http://schemas.dmtf.org/wbem/wsman/1/wsman'  
(4)   xmlns:rsp='http://schemas.microsoft.com/wbem/wsman/1/windows/shell'>  
(5) <s:Header>  
(6)   <wsa:Action>  
(7)     http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Command  
(8)   </wsa:Action>  
(9)   <wsa:MessageID>  
(10)    uuid:e7c5726b-de29-4313-b4d4-b3425b201833  
(11)  </wsa:MessageID>  
(12)  <wsa:To>http://www.example.org/wsman</wsa:To>  
(13)  <wsman:ResourceURI>  
(14)    http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd  
(15)  </wsman:ResourceURI>  
(16)  <wsman:SelectorSet>  
(17)    <wsman:Selector Name="ShellId">  
(18)      uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6  
(19)    </wsman:Selector>  
(20)  </wsman:SelectorSet>  
(21)  <wsman:OperationTimeout>PT30S</wsman:OperationTimeout>  
(22)  <wsman:MaxEnvelopeSize>65536</wsman:MaxEnvelopeSize>  
(23) </s:Header>  
(24) <s:Body>  
(25)   <rsp:CommandLine>  
(26)     <rsp:Command> dir </rsp:Command>  
(27)     <rsp:Arguments> /s </rsp:Arguments>  
(28)     <rsp:Arguments> c:\temp </rsp:Arguments>  
(29)   </rsp:CommandLine>  
(30) </s:Body>  
(31) </s:Envelope>
```

Lines (6-8) indicate this message is a `rsp:Command` request and that the shell processor is expected to respond with a `rsp:CommandResponse` message.

Lines (12-20) contains the endpoint reference from the original `wxf:ResourceCreated` message which targets the specific shell instance.

Line (21) indicates the output should be generated no more than 30 seconds after receipt of the `Command` request.

Line (22) indicates that the response message should contain no more than 64K octets.

Lines (25-29) represent the simple command line to be executed by the processor. Specifically, line (26) says that the **dir** command will be executed, and the arguments are `"/s c:\temp"`, indicating a recursive directory listing of the `c:\temp` directory.

This message is the hypothetical response:

```
(32) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope '  
(33)   xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing '  
(34)   xmlns:wsman='http://schemas.dmtf.org/wbem/wsman/1/wsman '  
(35)   xmlns:rsp='http://schemas.microsoft.com/wbem/wsman/1/windows/shell '>  
(36) <s:Header>  
(37)   <wsa:Action>  
(38)     http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandResponse  
(39)   </wsa:Action>  
(40)   <wsa:MessageID>  
(41)     uuid:e7c5726b-de29-4313-b4d4-b3425b200840  
(42)   </wsa:MessageID>  
(43)   <wsa:RelatesTo>  
(44)     uuid:e7c5726b-de29-4313-b4d4-b3425b200839  
(45)   </wsa:RelatesTo>  
(46) </s:Header>  
(47) <s:Body>  
(48)   <rsp:CommandResponse>  
(49)     <rsp:CommandId>  
(50)       uuid:f7c2726b-1e29-4313-34d4-23425b200831  
(51)     </rsp:CommandId>  
(52)   </rsp:CommandResponse>  
(53) </s:Body>  
(54) </s:Envelope>
```

Lines (36-38) indicate this message is a `rsp:CommandResponse` message. Line 49 contains the **CommandId** identifier that distinguishes this particular command within the scope of the shell instance that was specified in the `rsp:Command` message.

The `rsp:CommandId` is used in subsequent `rsp:Send` and `rsp:Receive` messages to control the I/O streaming to and from the command and to terminate the command using `rsp:Signal`.

### 3.7 Stream Encoding

In order to properly interact with a command, I/O streams must be supplied and consumed. This section discusses how stream I/O is encoded within `rsp:Send` and `rsp:Receive` messages.

Since shell processors typically segregate input and output (`stdin/stdout`) and usually separate the output in to a primary output stream (`stdout`) and an error or status stream



### 3.7.1 Encoded Data

The protocol says nothing about the format of the encoded data streams. In many cases, the command or shell processor may consistently return ANSI data, UNICODE data, or may return encodings which are specific to the command being executed, and may require input streams in either or both forms. The client must generally be aware of the behavior of the shell and any specific command prior to execution.

### 3.7.2 Stream Naming

In the absence of specific requirements for stream naming, both consumers and services should attempt to use "**stdin**" for input streams, and "**stdout**" for the default output stream, and "**stderr**" for the error or status output stream.

However, new shell processors designed explicitly to work with this specification may establish new naming.

### 3.7.3 Logical Records in Streams

Within a stream, there may be logical records of data. This may happen independently of the message boundaries. There are cases where the stream contains a logical block, paragraph, or object which is too large to fit into a single SOAP message, so the logical block must be distributed over several messages. In such cases, some kind of marker is required by the consuming side in order to correctly interpret the data item and reconstitute its original meaning.

Streams may have an additional marker which indicates that a logical record is beginning. This is indicated by the optional **Unit** attribute:

```
(1) <rsp:Stream Name="stderr" SequenceId="xs:unsignedLong"
(2)     Unit="xs:anyURI">
(3)     ...encoded data...
(4) </rsp:Stream>
```

This attributes may appear alone on the Stream element, or may appear in combination with encoded data. If it appears alone, it indicates that the unit of transmission identified by the URI value is about to begin. Any subsequent data within the same stream is considered to be part of the unit and the attribute need not reappear. The unit is considered to be in effect until the EndUnit marker is encountered:

```
(5) <rsp:Stream Name="stderr" SequenceId="xs:unsignedLong"
(6)     EndUnit="true">
(7)     ...encoded data...
(8) </rsp:Stream>
```

Units may be nested. That is, Unit "A" may begin, followed by "Unit B" and the first EndUnit applies to the most recently established unit in LIFO order.

This specification establishes no standard unit URI values. These are primarily for use in special types of shell processors which work with non-text data.

### 3.7.4 Stream Binding

A stream may be tied to a particular command, or it may relate to the shell processor itself. This is indicated by an optional attribute `CommandId`:

```
(1)      <rsp:Stream Name="stdout" SequenceId="3" CommandId="xs:anyURI">
(2)          c3Rpbmd1aXNoZWQsIG5vdCBvbmh5IGJ5IGhpc=
(3)      </rsp:Stream>
```

If the `CommandId` attribute is omitted, then the stream is bound to the shell processor. In this case, output streams usually contain shell messages, prompts, error messages, etc. Input streams sent to the shell are usually used for shell configuration, but **MUST NOT** be indirectly used to activate or execute a command. In other words, do not send a stream to the shell which contains "cd" as an input stream in order to get the "cd" command to execute, but send the "cd" in the form of a command.

If the `CommandId` attribute is present it **MUST** contain a `CommandId` URI which was obtained from a prior `rsp:CommandResponse` message. Input and output streams tied to a particular `CommandId` relate only to that command.

### 3.7.5 Stream Ordering And Size

As many named streams may coexist as required, and the same stream may be repeated any number of times within a given message. All content wrapped by a `rsp:Stream` wrapper is taken to be part of the stream indicated by the `Name` attribute.

The processing order of all data across streams is assumed to be the order of the `rsp:Stream` blocks within the messages. The processing order of data elements within a stream is based on the `SequenceId` attribute, regardless of the position of that block in the message.

This means that ordering may be preserved within a stream as well as across streams.

The protocol stack has no responsibility to analyze or validate stream content, but is only required to extract stream content and deliver it to the shell processor, or accept output from the shell processor and to wrap it as specified above.

The maximum size of a SOAP packet for all stream content is specified in WS-Management (`wsman:MaxEnvelopeSize`). There is no requirement on the size or number of the encoded stream blocks.

To signal the end of a particular stream, the `End` attribute should be added to the last block for that stream in the last message:

```
(1)      <s:Body>
(2)          ...
(3)      <rsp:Stream Name="stdout" SequenceId="33" End="true">
(4)          ...
(5)      </rsp:Stream>
```

The stream may have content or may simply be an end-marker:

```
(6)      <rsp:Stream Name="stdout" SequenceId="41" End="true"/>
```

Note that to preserve ordering across stream boundaries, it may be necessary to interlace the same stream many times between other streams:

```
(7)      <rsp:Stream Name="stdout" SequenceId="2">
(8)
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyBFZb24sIGJ1dCBieSB0==
(9)      </rsp:Stream>
(10)
(11)     <rsp:Stream Name="stderr" SequenceId="1">
(12)       TWFuIGlzIGRpc3Rpbmd1aXNoZhpByZWZb24sIGJ1dCBieSB0=
(13)     </rsp:Stream>
(14)     <rsp:Stream Name="stdout" SequenceId="3">
(15)       c3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpc=
(16)     </rsp:Stream>
(17)       ...
(18)     </s:Body>
```

In the above example, the "stdout" stream appears twice, because the output from the command itself needs to specify this ordering. Note that the SequenceId increases from 2 to 3 when this happens.

A SequenceId for a given stream MUST NOT appear out of order within the SOAP body. In other words, if SequenceId 3 shows up for a particular stream, then SequenceId 2 must not appear later within the s:Body element.

### 3.8 rsp:Send

The rsp:Send operation is used to pipe input to a running command and is initiated by sending a rsp:Send request message to the shell processor. The rsp:Send message may also be used to send input directly to the shell processor, whether or not a command is currently executing, if supported by the specific shell processor.

The rsp:Send request message MUST be of the following form:

```
(1) <s:Envelope ...>
(2) <s:Header ...>
(3) <wsa:Action>
(4)   http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Send
(5) </wsa:Action>
(6) ...EPR of shell instance...
(7) </s:Header>
(8) <s:Body ...>
(9)   <rsp:Send ...>
(10)    <rsp:Stream Name="xs:token" SequenceId="xs:unsignedLong" ?
(11)      CommandId="xs:anyURI"? End="xs:boolean"? >
(12)    xs:base64binary
(13)  </rsp:Stream> *
```

```
(14)    </rsp:Send>
(15)    </s:Body>
(16)    </s:Envelope>
```

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

/s:Envelope/s:Header/wsa:Action (line 4)

This required element MUST contain the value  
`http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Send`

/s:Envelope/s:Body/rsp:Send (lines 9-14)

This required element includes the input being sent to the shell processor which can include data from one or more streams or simply indicate the end of one or more streams.

/s:Envelope/s:Body/rsp:Send/rsp:Stream

This optional element contains any necessary Input stream data items necessary for the command to execute. The content is base64binary encoded.

/s:Envelope/s:Body/rsp:Send/rsp:Stream/@Name

The name of the stream. This is required.

/s:Envelope/s:Body/rsp:Send/rsp:Stream/@SequenceId

The REQUIRED sequence number of the data block for the stream. This begins at 0 and increases with each new appearance of this stream in the same message or subsequent messages.

/s:Envelope/s:Body/rsp:Send/rsp:Stream/@CommandId

If present, this indicates that the stream relates to a currently executing command. The value is the one returned in the `rsp:CommandResponse` message. If absent, the stream relates directly to the shell processor and not any command or commands that may be in progress. In practice, most streams will be tied to the currently executing command or commands.

/s:Envelope/s:Body/rsp:Send/rsp:Stream/@End

If this optional attribute is set to "true" it indicates that the stream is ending and that no more content will occur for this stream for the currently executing command. This should not be used for streams tied to the shell processor itself.

### 3.8.1 Faults

Typically, an `rsp:Send` fault will be issued with the appropriate detail code (see 5.4).

### 3.8.2 Rules for Use

[1] Only one `rsp:Send` can be outstanding per instance of the shell. Only when a `rsp:SendResponse` is received may another `rsp:Send` occur. However, a `rsp:Send` may occur asynchronously to any `rsp:Receive` request.

[2] If the input is being sent directly to the shell itself, then the `CommandId` attribute is omitted from the `rsp:Stream` element.

[3] If the input is being sent to a specific command or the shell and the command completes or fails without consuming all the input, then that additional unconsumed input is discarded and ignored.

As an example, assume a remote SORT utility continues to buffer input for sorting until a Control-Z character appears in the input stream. At this point, the SORT utility sorts the input and begins emitting output. If the client continues to send input beyond this point, the sort utility will ignore it, since the trigger condition for performing the sort and emitting the output has already occurred. This additional input will not be held over to the next command for the shell, but will simply be discarded and may be faulted with an `rsp:Send` fault (see 5.4).

[4] The client may send any mix of input streams that is required. The processor may issue a `rsp:Send` fault (5.4) indicating the wrong stream was sent. In this particular case, the client must retain the input and retransmit it in a future `rsp:Send` request. There is no requirement that the message be identical to the original, as long as the client maintains the original logical position in each input stream.

Note that if the processor requires a specific stream to be sent, it should accept the current streaming input and buffer it, but return a `rsp:DesiredStream` in the `rsp:SendResponse` message. It may then wait for that specific stream to show up and issue `rsp:Send` faults (see 5.4) indicating that the wrong stream was sent until it gets the required stream. In cases where the shell processor cannot effectively buffer, it may in fact issue a `rsp:Send` fault with the appropriate detail code (5.4).

[5] Sending empty stream content ("dummy" messages) is permitted. This may be required to prevent deadlock or livelock in certain scenarios with heavy input and output interaction.

[6] Intermittent connectivity can introduce problems with stream integrity. If the client issues an `rsp:Send` which is not received or acknowledged, it should resend the same content as it did previously until it gets a response. The `SequenceId` attributes are used in this case to ensure that the shell processor does not skip or duplicate data items.

### 3.8.3 `rsp:SendResponse`

Upon successful processing of a `rsp:Send` request message, a processor is expected to return a `rsp:SendResponse` response message, which MUST adhere to the following form:

```
(1)  <s:Envelope ...>
(2)    <s:Header ...>
(3)      <wsa:Action>
(4)        http://schemas.microsoft.com/wbem/wsman/1/windows/shell/SendResponse
(5)      </wsa:Action>
(6)    ...
(7)  </s:Header>
(8)  <s:Body>
```

```

(9)         <rsp:SendReponse>
(10)            <rsp:DesiredStream>
(11)                xs:token
(12)            </rsp:DesiredStream> ?
(13)        </rsp:SendResponse>
(14)    </s:Body>
(15) </s:Envelope>

```

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

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

This required element MUST contain the value

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/SendResponse>.

/s:Envelope/s:Body/rsp:SendResponse/rsp:DesiredStream

This OPTIONAL element allows the shell processor to request input from a particular stream in future rsp:Send messages.

If this is omitted, then the shell processor will accept input from any stream in future messages.

### 3.8.4 Examples

The following represents a hypothetical rsp:Send.

```

(1) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope '
(2)   xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing '
(3)   xmlns:wsman='http://schemas.dmtf.org/wbem/wsman/1/wsman '
(4)   xmlns:rsp='http://schemas.microsoft.com/wbem/wsman/1/windows/shell '
(5) <s:Header>
(6)   <wsa:Action>
(7)     http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Send
(8)   </wsa:Action>
(9)   <wsa:MessageID>
(10)     uuid:e7c5726b-de29-4313-b4d4-b3425b200839
(11)   </wsa:MessageID>
(12)   <wsa:To>http://www.example.org/wsman</wsa:To>
(13)   <wsman:ResourceURI>wsman:shell</wsman:ResourceURI>
(14)   <wsman:SelectorSet>
(15)     <wsman:Selector Name="shell">
(16)       uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6
(17)     </wsman:Selector>
(18)   </wsman:SelectorSet>
(19)   <wsman:OperationTimeout>PT30S</wsman:OperationTimeout>
(20) </s:Header>
(21) <s:Body>
(22)   <rsp:Send>
(23)     <rsp:Stream Name="stdin" SequenceId="0"

```

```

(24)         CommandId="uuid:e7c5726b-de29-4313-b4d4-b3425b200839"
(25)         End="true">
(26)             TWFuIGlzIGRpc3Rpbmd1aXNohpcyByZWZzb24sIGJldCBieSB0=
(27)         </rsp:Stream>
(28)     </rsp:Send>
(29) </s:Body>
(30) </s:Envelope>

```

Lines (6-8) indicate this message is a Send request and that the shell processor is expected to respond with a rsp:SendResponse message.

Lines (12-18) contains the endpoint reference from the original wxf:ResourceCreated message per requirement R2.1-1 [\[WS-Management\]](#).

Line (19) indicates the response should be generated no more than 30 seconds after receipt of the Send request message.

Lines (23-28) show the input stream being sent to the currently executing command.

The End attribute on line 25 indicates that this is the last set of inputs to the "stdin" stream and that no more input will be occurring for that stream.

Here is the hypothetical response to the rsp:Send message:

```

(31) <s:Envelope 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.dmtf.org/wbem/wsman/1/wsman'
(34)     xmlns:rsp='http://schemas.microsoft.com/wbem/wsman/1/windows/shell'>
(35) <s:Header>
(36)     <wsa:Action>
(37)     http://schemas.microsoft.com/wbem/wsman/1/windows/shell/SendResponse
(38)     </wsa:Action>
(39)     <wsa:MessageID>
(40)         uuid:e7c5726b-de29-4313-b4d4-b3425b200840
(41)     </wsa:MessageID>
(42)     <wsa:RelatesTo>
(43)         uuid:e7c5726b-de29-4313-b4d4-b3425b200839
(44)     </wsa:RelatesTo>
(45) </s:Header>
(46) <s:Body>
(47)     <rsp:SendResponse/>
(48) </s:Body>
(49) </s:Envelope>

```

Lines (36-38) indicate this message is a rsp:SendResponse message.

Line (47) indicates that the input was accepted. Since the client already sent an "End" attribute on the send request, this message is nothing more than an acknowledgment of receipt. If the client had not sent the "End" attribute, this message might have contained:

```

(50) <s:Body>
(51)     <rsp:SendResponse>

```

```

(52)      <rsp:DesiredStream>
(53)      stdin
(54)      </rsp:DesiredStream>
(55)      </rsp:SendResponse>
(56)      </s:Body>

```

However, the server side processor is not required to indicate the desired stream unless there is more than one input stream and it actually requires data from the specified stream as its next input.

### 3.9 rsp:Receive

The `rsp:Receive` operation is used to collect output from a running command or the shell itself. It is initiated by sending a `rsp:Receive` request message to the shell processor. The `rsp:Receive` request is similar to the `wsen:Pull` request in `WS-Enumeration` but returns only `Stream` elements. Also note that a final `rsp:Signal` message must also be issued for the command after all the stream data has been received; it is not sufficient to simply use `rsp:Receive` to receive all the data.

The `rsp:Receive` request message MUST be of the following form:

```

(1)  <s:Envelope ...>
(2)  <s:Header ...>
(3)  <wsa:Action>
(4)  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Recieve
(5)  </wsa:Action>
(6)  <rsp:MaximizeResponseSize/> ?
(7)  ...other WS-Management headers...
(8)  </s:Header>
(9)  <s:Body ...>
(10) <rsp:Receive SequenceId="xs:unsignedInteger">
(11) <rsp:DesiredStreams CommandId="xs:anyURI"? />
(12) xs:list
(13) </rsp:DesiredStreams>?
(14) </rsp:Receive>
(15) </s:Body>
(16) </s:Envelope>

```

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

`/s:Envelope/s:Header/wsa:Action` (line 4)

This required element MUST contain the value  
<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Recieve>.

`/s:Envelope/s:Header/rsp:MaximizeResponseSize`

This OPTIONAL element is a hint to the service to batch as much information in the `SendResponse` as possible as long as the standard `WS-Management` headers `MaxEnvelopeSize` and `OperationTimeout` are not violated. The actual batch size and

timeout are controlled by these standard WS-Management header values. If this element is omitted, the service is still required to live within the limits defined by the WS-Management headers, but may issue responses which are significantly less in size than the maximum and which are returned well before the timeout has occurred.

/s:Envelope/s:Body/rsp:Receive/@SequenceId

This element ensures that intermittent connections and timeouts do not cause problems with stream integrity. The client places a zero value on the initial message and increases this by 1 with each subsequent `rsp:Receive` message until the operation completes. The `rsp:ReceiveResponse` will echo this value. If the client needs to retry an `rsp:Receive` operation because of a fault, it should *reuse* the `SequenceId` in order to ensure that the service provides continuity in the stream data. The client should only increase this value once an `rsp:ReceiveResponse` has been received with a copy of the value which was sent.

/s:Envelope/s:Body/rsp:Receive/rsp:DesiredStreams

This OPTIONAL element allows the client to request output from a particular stream and command. If absent, the shell processor is free to return data from any output stream or command in progress. If present, it lists all streams requested by the client. The shell processor MUST ensure that the response only contains output from this command and these streams. If no output is available for the requested stream or streams, the response will simply contain empty content. If this technique is used, it may result in losing the time-ordering of data items across streams.

/s:Envelope/s:Body/rsp:Receive/rsp:DesiredStream/@CommandId

This optional attribute indicates for which command the stream is being requested. The value is obtained from the prior `rsp:CommandResponse` for the current command. If absent, then the stream is being requested from the shell processor itself and not any outstanding executing command.

### 3.9.5 Faults

The primary fault is an `rsp:Receive` fault (5.5). Other WS-Management faults may apply.

### 3.9.6 `rsp:ReceiveResponse`

Upon successful processing of a `rsp:Receive` request message a processor returns a `rsp:ReceiveResponse` message, which MUST adhere to the following form:

```
(1) <s:Envelope ...>
(2) <s:Header ...>
(3) <wsa:Action>
(4)
http://schemas.microsoft.com/wbem/wsman/1/windows/shell/ReceiveResponse
(5) </wsa:Action>
(6) ...
(7) </s:Header>
(8) <s:Body>
(9) <rsp:ReceiveResponse SequenceId="xs:unsignedLong"?>
(10) <rsp:Stream Name="xs:string" CommandId="xs:anyURI"?
```

```

(11)     SequenceId="xs:unsignedLong"? End="xs:boolean"? >
(12)   ...
(13)   </rsp:Stream> *
(14)   <rsp:CommandState CommandId="xs:anyURI" State="xs:anyURI">
(15)     <rsp:ExitCode xs:int </rsp:ExitCode> ?
(16)   </rsp:CommandState>
(17)
(18) </rsp:ReceiveResponse>
(19) </s:Body>
(20) </s:Envelope>

```

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

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

This required element MUST contain the value  
<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/ReceiveResponse>.

/s:Envelope/s:Body/rsp:ReceiveResponse

This wrapper element contains a list of Stream elements.

/s:Envelope/s:Body/rsp:ReceiveResponse/@SequenceId

This value is a copy of the corresponding rsp:Receive SequenceId attribute in the original request message. This is used to ensure that the response corresponds to a particular request, even in cases where faults have been issued between operations.

The service may only send new stream data if the SequenceId is a new value not previously seen in a rsp:Receive request. If the service gets an rsp:Receive request with the same value as previously used in the prior rsp:Receive request, the service must send the same stream data as it did for the previous request, since the client either did not receive the stream data or was unable to process it.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:Stream

This element contains streamed output data. This element may be repeated any number of times for multiple streams or a single stream, in any order.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:Stream/@SequenceId

A OPTIONAL element which indicates the sequence number (starting at 0) for this block of data in the specific stream. Each new block for this stream must use the succeeding integer value, whether this block is in the same message or subsequent messages.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:Stream/@CommandId

Indicates the command to which this output relates. If absent, the output is assumed to be from the shell processor itself.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:Stream/@End

If this optional attribute is set to "true" it indicates the end of that particular stream. It is used for optimization purposes if the shell processor in fact knows that no more output will occur for this stream. In some cases, this cannot be determined.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:CommandState

Reports the status of the executing command.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:CommandState/@CommandId

Indicates the identity of the command for which status is being reported.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:CommandState/@State

This REQUIRED attribute indicates the specific state of the command for which status is being reported. It may contain one of the following values:

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandState/Done>

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandState/Pending>

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandState/Running>

Any additional status is reported through stream output.

/s:Envelope/s:Body/rsp:ReceiveResponse/rsp:CommandState/rsp:ExitCode

This OPTIONAL element contains a processor specific exit code. It is recommended that zero be the value for no-error, and that nonzero be used for error conditions.

### 3.9.7 Rules For Use

[1] When the shell was opened with wxf:Create, the client and service established a contractual obligation regarding the output streams that would be in use. The client must be prepared to accept output from any of these streams. If additional output streams are available and were not part of the initial wxf:Create handshake, the server side processor MUST omit these streams in the response.

[2] The rsp:Receive message may execute concurrently or asynchronously with rsp:Send messages, but only one rsp:Receive may be outstanding at any given time.

[3] A client SHOULD immediately issue a rsp:Receive message when a command is launched, whether or not it will be sending input using rsp:Send messages. To prevent deadlock, livelock, or timeout situations, the server may return rsp:Receive messages with empty string content, but typically it will delay responding until output is available, providing that wsman:OperationTimeout rules are not violated. The client should continue to issue rsp:Receive messages as soon as the previous rsp:ReceiveResponse has been received.

[4] At least one rsp:Receive MUST be issued in order to get a final rsp:ReceiveResponse which indicates that a command has terminated. The server may reject processing new rsp:Command requests until it has successfully returned a rsp:ReceiveResponse with a rsp:CommandState element, indicating that the command has terminated or succeeded.

When the client receives the final output from a command, as illustrated above, it MUST subsequently send a rsp:Signal with a code of <http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Exit> to signal to the command processor that it may discard all final state information for the most recently executed command.

The reason for this final step is that the server will cache the final rsp:ReceiveResponse content in order to allow for intermittent connections. The server cannot be certain that the client has received the final command status. Only by acknowledging this with a subsequent rsp:Signal can the server safely release the final command status information.

[5] While some individual streams may optionally end using the rsp:Stream/@End attribute, the completion of a command and consequently its entire output is distinct and signaled using the rsp:CommandState element with the <rsp:State> element having the value rsp:CommandState/NormalTermination or rsp:CommandState/AbnormalTermination.

The server MAY notify that a command is blocked while waiting for input stream content by returning `rsp:CommandState` element with the `<rsp:Status>` element set to `rsp:CommandState/Waiting`.

[6] Because of timeouts and intermittent connectivity, `rsp:Receive` requests may be faulted or may timeout with no response. In such cases, the client can terminate the operation, but may retry in order to attempt to continue executing the command. In order for the service to ensure continuity of the stream data, the sender places a `SequenceId` in the `rsp:Receive` request and the service echos the same value in the `rsp:ReceiveResponse`. Only once the client has successfully retrieved stream data (even though faults may have occurred between messages) should the client increment the `SequenceId` value. If the service sees a new value not previously seen, it returns new stream data and repeats the value. IF the service sees a value which was previously seen, then it must either issue an `rsp:Receive` fault or replay the previous batch of data with precisely the same content as it did before. Many implementations will cache the most recent response, so this will be straightforward. Implementations which do not cache data may not be able to honor the replay request, but are required to detect that the value has been used before and issue the fault.

### 3.9.8 Examples

Here is a hypothetical receive request:

```
(1) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope '  
(2)   xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing '  
(3)   xmlns:wsman='http://schemas.dmtf.org/wbem/wsman/1/wsman '  
(4)   xmlns:rsp='http://schemas.microsoft.com/wbem/wsman/1/windows/shell '  
(5)   <s:Header>  
(6)     <wsa:Action>  
(7)   http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Receive  
(8)     </wsa:Action>  
(9)     <wsa:MessageID>  
(10)      uuid:e7c5726b-de29-4313-b4d4-b3425b200839  
(11)     </wsa:MessageID>  
(12)     <wsa:To>http://www.example.org/wsman</wsa:To>  
(13)     <wsman:ResourceURI>  
(14)   http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd  
(15)     </wsman:ResourceURI>  
(16)     <wsman:SelectorSet>  
(17)       <wsman:Selector Name="ShellId">  
(18)         uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6  
(19)       </wsman:Selector>  
(20)     </wsman:SelectorSet>  
(21)     <wsman:OperationTimeout>PT30S</wsman:OperationTimeout>  
(22)   </s:Header>
```

```
(23)    <s:Body>
(24)        <rsp:Receive/>
(25)    </s:Body>
(26) </s:Envelope>
```

Lines (6-8) indicate this message is a `rsp:Receive` request and that the shell processor is expected to respond with a `rsp:ReceiveResponse` message.

Lines (12-20) contains the endpoint reference from the original `wxf:ResourceCreated` message.

Line (21) indicates it should be generated no more than 30 seconds after receipt of the `Receive` request message.

If more than one output stream is available, the client may request a specific stream using the following `s:Body`:

```
(27)    <s:Body>
(28)        <rsp:Receive>
(29)            <rsp:DesiredStream CommandId="cid:1230"/>
(30)                stdout
(31)            </rsp:DesiredStream>
(32)        </rsp:Receive>
(33) </s:Body>
```

Here is the hypothetical response:

```
(34) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope '
(35)     xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing '
(36)     xmlns:wsman='http://schemas.dmtf.org/wbem/wsman/1/wsman '
(37)     xmlns:rsp='http://schemas.microsoft.com/wbem/wsman/1/windows/shell '
(38)
(39) <s:Header>
(40)     <wsa:Action>
(41) http://schemas.microsoft.com/wbem/wsman/1/windows/shell/ReceiveResponse
(42)     </wsa:Action>
(43)     <wsa:MessageID>
(44)         uuid:e7c5726b-de29-4313-b4d4-b3425b200840
(45)     </wsa:MessageID>
(46)     <wsa:RelatesTo>
(47)         uuid:e7c5726b-de29-4313-b4d4-b3425b200839
(48)     </wsa:RelatesTo>
(49) </s:Header>
(50) <s:Body>
(51)     <rsp:ReceiveResponse>
(52)         <rsp:Stream Name="stdout" SequenceId="0"
```

```

(53)         CommandId="C122" End="true">
(54)         TWFuIGlzIGRpc3Rpbmd1aXNohpcyByZWZzb24sIGJldCBieSB0=
(55)         </rsp:Stream>
(56)         <rsp:CommandState>
(57)             <rsp:Status>
(58)                 rsp:CommandState/NormalTermination
(59)             </rsp:Status>
(60)             <rsp:ExitCode> 0 </rsp:ExitCode>
(61)         </rsp:CommandState>
(62) </rsp:ReceiveResponse>
(63) </s:Body>
(64) </s:Envelope>

```

Lines (38-40) indicate this message is a `rsp:ReceiveResponse` message.

Lines (49-60) represent the output. The actual command processor output is returned in the "stdout" stream (Line 52) in the literal output of the shell processor.

Line (56) indicates that the command is finished and that the command completed successfully and returned a processor-specific exit code of 0 (line 58). The shell processor is now ready to receive another Command request. If error text is required in addition to the numeric codes, these could have been returned in an additional **stderr** stream.

After receiving this message, the client MUST emit a `rsp:Signal` with a control code of `http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Exit` to acknowledge receipt of this termination state information and this last set of outputs.

### 3.10 `rsp:Signal`

The `rsp:Signal` operation is used to control the shell processor. It can be sent either asynchronously or synchronously. This is used to terminate each command after completion, and can also be used to effect an early termination of the currently running command ("Control-Break"). It may be used for other purposes such as to "pause" or "resume" execution, or any other need for asynchronous signaling.

A signal may be sent to a specific command, or to the shell in general. Whether or not a given shell supports targeting commands or just targeting the shell itself is specific to the implementation.

The Signal request message MUST be of the following form:

```

(1) <s:Envelope ...>
(2)   <s:Header ...>
(3)     <wsa:Action>
(4)       http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Signal
(5)     </wsa:Action>
(6)     ...other WS-Management headers...
(7)
(8)   </s:Header>

```

```

(9)      <s:Body ...>
(10)     <rsp:Signal CommandId="xs:anyURI"? >
(11)     <rsp:Code> xs:anyURI </rsp :Code>
(12)     </rsp:Signal>
(13)     </s:Body>
(14) </s:Envelope>

```

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

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

This required element MUST contain the value  
<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Signal>

/s:Envelope/s:Body/rsp:Signal

This wrapper element specifies the signal code associated with the specific command or the shell itself.

/s:Envelope/s:Body/rsp:Signal/rsp:Code

This required element contains the control code being sent to the shell or to a specific command. This may be any valid URI recognized by the shell processor. If the following semantics apply, the following URIs should be used in preference to others:

**<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Terminate>**

Indicates the shell should terminate any currently running command or commands and return to an idle state. This represents the behavior of Control-C. The underlying implementation should take whatever steps are necessary in order to terminate the command and bring the shell back to an idle state.

**<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Break>**

This URI represents the behavior Control-Break. While most of the tools do not differentiate between Control-Break and Control-C some handle them as separate and different signals. The underlying implementation should take whatever steps are necessary to ensure that the running command can process Control-Break separately from Control-C.

**<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Pause>**

Indicates the shell should suspend any currently running command or commands and wait for further instructions.

**<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Resume>**

Indicates the shell should resume execution if it was previously paused.

**<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Exit>**

This control code is sent by the client to acknowledge receipt of the end of the command output from the service and to clean up any resources associated with command execution. Because the service may have to cache this information until it knows the client has received the entire stream, the client MUST send this before a new command can be executed.

/s:Envelope/s:Body/rsp:Signal/@CommandId

This optional attribute directs the signal to a specific command instead of the shell. This is used when terminating individual commands, but typically not used for other cases.

### 3.10.1 Faults

In addition to WS-Management faults, a rsp:Signal fault may be returned (5.6):

### 3.10.2 **rsp:SignalResponse**

Upon successful processing of a `rsp:Signal` request message, a processor is expected to return a `rsp:SignalResponse` response message:

```
(1) <s:Envelope ...>
(2)   <s:Header ...>
(3)     <wsa:Action>
(4)
http://schemas.microsoft.com/wbem/wsman/1/windows/shell/SignalResponse
(5)   </wsa:Action>
(6)   <wsa:MessageId> ... </wsa:MessageId>
(7)   <wsa:RelatesTo> ... </wsa:RelatesTo>
(8)   ...other WS-Management headers...
(9) </s:Header>
(10) <s:Body>
(11)   <rsp:SignalResponse>
(12)   optional shell-specific content
(13)   </rsp:SignalResponse>
(14) </s:Body>
(15)
```

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

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

This required element **MUST** contain the value

`http://schemas.microsoft.com/wbem/wsman/1/windows/shell/SignalResponse`.

`/s:Envelope/s:Body`

Shell processors **MAY** include shell-specific status information in the `s:Body` within an `rsp:SignalResponse` element. This is not a requirement and the content is not usually actionable. It is primarily for informational or statistical purposes, and might include information on how long the shell was open, or how much quota was charged to the user. This should not be used to indicate an error, since a fault is the appropriate mechanism.

The `wsa:RelatesTo` field is used to correlate the response with the original signal.

### 3.10.3 **Examples**

Here is a hypothetical Signal request:

```
(1) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope '
(2)   xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing '
(3)   xmlns:rsp='http://schemas.microsoft.com/wbem/wsman/1/windows/shell '
(4) <s:Header>
(5)   <wsa:Action>
```

```

(6)      http://schemas.microsoft.com/wbem/wsman/1/windows/shell/Signal
(7)      </wsa:Action>
(8)      <wsa:MessageID>
(9)          uuid:e7c5726b-de29-4313-b4d4-b3425b200839
(10)     </wsa:MessageID>
(11)     <wsa:To>http://www.example.org/wsman</wsa:To>
(12)     <wsman:ResourceURI>
(13) http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd
(14)     </wsman:ResourceURI>
(15)     <wsman:SelectorSet>
(16)         <wsman:Selector Name="ShellId">
(17)             uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6
(18)         </wsman:Selector>
(19)     </wsman:SelectorSet>
(20) </s:Header>
(21) <s:Body>
(22) <rsp:Signal>
(23)
http://schemas.microsoft.com/wbem/wsman/1/windows/shell/signal/Terminate
(24) </rsp:Signal>
(25) </s:Body>
(26) </s:Envelope>

```

Lines (5-7) indicate this message is a `rsp:Signal` request and that the shell processor is expected to respond with a `rsp:SignalResponse` message.

Lines (11-19) contains the endpoint reference from the original `wxf:ResourceCreated` message per requirement R2.1-1 [\[WS-Management\]](#).

Line (23) indicates that the shell is being requested to terminate execution (Control-C).

Here is the response:

```

(27) <s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope'
(28) xmlns:wsa='http://schemas.xmlsoap.org/ws/2004/08/addressing'
(29) <s:Header>
(30)     <wsa:Action>
(31) http://schemas.microsoft.com/wbme/wsman/1/windows/shell/SignalResponse
(32) </wsa:Action>
(33) <wsa:MessageID>
(34)     uuid:e7c5726b-de29-4313-b4d4-b3425b200840
(35) </wsa:MessageID>
(36) <wsa:RelatesTo>
(37)     uuid:e7c5726b-de29-4313-b4d4-b3425b200839
(38) </wsa:RelatesTo>
(39) </s:Header>

```

```
(40) <s:Body/>
(41) </s:Envelope>
```

Line (31) indicates this message is a `rsp:SignalResponse` message and it is correlated with the original message using the `wsa:RelatesTo` value (line 37).

### 3.11 WS-Eventing

The use of WS-Eventing mechanisms is not defined for this version of the specification.

In a future version, it is expected that administrators could subscribe to real-time events relating to shell usage or command execution.

## 4.0 Security Considerations

### 4.1 WS-Management Security

Message- and transport-level security are already defined for WS-Management [[WS-Management](#)] and this specification layers over those mechanisms. However, special restrictions regarding authorization should be observed due to the nature of the shell itself.

### 4.2 Authorization

Since a shell contains state information, typically authorization to open a shell is checked during the `wxf:Create` operation. At this point, implementations should establish a durable security context for the entire lifetime of the shell and associate any user-level security tokens with that shell at that time.

Any subsequent messages and operations against that specific shell instance should be checked to ensure that the same user which created the shell is the one carrying out the operations.

It is NOT permitted for other users to make use of a running shell instance, even if they have permissions to create shells themselves with the same or even greater privileges.

There is no requirement that the shell be referenced from precisely the same machine or IP address that created it, as long as it can be established that the same user is the one who created it. This allows the original user to handle degenerate cases: to close shells or do other cleanup from a different machine or vantage point than the machine that created the shell.

## 5.0 Faults

Faults are encoded as described in Chapter 11 of WS-Management [[WS-Management](#)]. Any faults defined in WS-Management may occur within WS-Shell, and additionally the faults defined in this section may occur.

### 5.1 `wxf:InvalidRepresentation`

Fault Subcode	wfx:InvalidRepresentation
Action URI	http://schemas.xmlsoap.org/ws/2004/09/transfer/fault
Code	s:Sender
Reason	The XML content is not valid
Detail	<p>&lt;s:Detail&gt;</p> <p><i>One of the fault detail codes defined for this fault in WS-Management or one of the codes below</i></p> <p>&lt;/s:Detail&gt;</p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidWorkingDirectory  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidEnvironmentVariable  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidStream  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidIdleTimeout  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidLifetime  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidExtension</p>
Comments	<p>This is returned when the processor cannot deliver the command to the shell processor, but not when the shell processor itself returns an error, which is considered normal output.</p> <p>Valid usage of this fault is when the command is syntactically invalid or missing, or required extensions in the command extension content is missing or incorrect.</p>
Applicability	wfx:Create
Remedy	Client corrects the operation payload and resubmits it.

## 5.2 wsman:QuotaLimit

Fault Subcode	wsman:QuotaLimit
Action URI	http://schemas.dmtf.org/wbem/wsman/1/wsman/fault
Code	s:Sender
Reason	The server has exceeded a quota limit
Detail	<p>&lt;s:Detail&gt;</p> <p><i>One of the fault detail codes defined for this fault in WS-Management or one of the codes below</i></p> <p>&lt;/s:Detail&gt;</p>

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/MaxMemoryPerShell>

*The operation has exceeded maximum allowed memory per shell*

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/MaxProcessesPerShell>

*The operation has exceeded maximum allowed number of processes per shell*

**Comments** This is returned when the processor cannot deliver the command to the shell processor, but not when the shell processor itself returns an error, which is considered normal output.

Valid usage of this fault is when the command is syntactically invalid or missing, or required extensions in the command extension content is missing or incorrect.

**Applicability** Any operation

**Remedy** Client adjusts the current quota settings on the server.

### 5.3 rsp:CommandFault

<b>Fault Subcode</b>	rsp:CommandFault
----------------------	------------------

<b>Action URI</b>	<a href="http://schemas.microsoft.com/wbem/wsman/1/windows/shell/fault">http://schemas.microsoft.com/wbem/wsman/1/windows/shell/fault</a>
-------------------	---

<b>Code</b>	s:Sender
-------------	----------

<b>Reason</b>	The client has exceeded a quota limit
---------------	---------------------------------------

<b>Detail</b>	<s:Detail> <i>One of the fault detail codes below</i> </s:Detail>
---------------	---

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidCommand>  
*The command was invalid and could not be delivered to the processor.*

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidExtension>  
*The extension content was not valid for the shell or command.*

**Comments** This is returned when the processor cannot deliver the command to the shell processor, but not when the shell processor itself returns an error, which is considered normal output.

Valid usage of this fault is when the command is syntactically invalid or missing, or required extensions in the command extension content is missing or incorrect.

Applicability	rsp:Command
Remedy	Client corrects the command and resubmits it.

## 5.4 rsp:SendFault

Fault Subcode	rsp:SendFault
Action URI	http://schemas.microsoft.com/wbem/wsman/1/windows/shell/fault
Code	s:Sender
Reason	The server cannot process the input stream(s)
Detail	<p>&lt;s:Detail&gt;</p> <p><i>One of the fault detail codes below</i></p> <p>&lt;/s:Detail&gt;</p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/Buffer <i>The receiving buffer is full and the delivery was not accepted.</i></p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidStream <i>The stream which was delivered was not valid for the shell.</i></p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/StreamEncoding <i>The stream is not encoded or delimited properly.</i></p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/Unexpected <i>The processor did not expect input at this time.</i></p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidCommandId <i>The referenced CommandId is not valid.</i></p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/Busy <i>The shell processor is busy and cannot accept input at this time.</i></p>
Comments	This is returned for an rsp:Send operation when there is a buffer overrun (input is delivered faster than can be processed), or the wrong stream is being used or not correctly formatted or sequenced, or input was not even expected at all.
Applicability	rsp:Send
Remedy	Client retries with correct stream usage.

## 5.5 rsp:ReceiveFault

Fault Subcode	rsp:ReceiveFault
Action URI	http://schemas.microsoft.com/wbem/wsman/1/windows/shell/fault
Code	s:Sender
Reason	The receive operation cannot complete
Detail	<p>&lt;s:Detail&gt;</p> <p>One of the fault detail codes below</p> <p>&lt;/s:Detail&gt;</p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/Unexpected The processor did not expect to deliver stream output at this time.</p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidStream The requested stream is not valid.</p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidCommandId The referenced CommandId is not valid.</p> <p>http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/SequenceId The SequenceId indicates a break in stream continuity and the service cannot continue returning results.</p>
Comments	<p>This is returned for any operation whenever the rsp:Receive request cannot be processed. Either there is no command in progress, or there is no more output, or the message is incorrectly formatted.</p> <p>Other situations use WS-Management faults.</p>
Applicability	rsp:Receive
Remedy	Client may retry with correct stream or after properly launching a command.

## 5.6 rsp:SignalFault

Fault Subcode	rsp:SignalFault
Action URI	http://schemas.microsoft.com/wbem/wsman/1/windows/shell/fault
Code	s:Sender
Reason	The client sent an invalid signal
Detail	<s:Detail> <i>One of the fault detail codes below</i> </s:Detail>  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/UnkownSignal <i>The signal URI is unknown.</i>  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidSignalContext <i>The signal is not applicable in the current situation.</i>  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/AlreadyProcessing <i>The signal is already being processed (sent to fault repeated deliveries of the same signal).</i>  http://schemas.microsoft.com/wbem/wsman/1/windows/shell/faultDetail/InvalidCommandId <i>The referenced CommandId is not valid.</i>
Comments	This is returned when the signal that was delivered was incorrect, either because it was not recognized or incorrect for the current execution context.
Applicability	rsp:Signal
Remedy	Client retries with correct signal.

## 6.0 Standard Shell Processor in Microsoft Windows<sup>®</sup>

### 6.1 Introduction

The Microsoft Windows *Vista* release has an implementation of the Remote Shell Protocol described in this specification. There are a few specific usages of the protocol documented within this section.

## 6.2 ResourceURI and ShellId

The wsman:ResourceURI for creating new instances of a remote shell for use with the cmd.exe command processor in Windows is:

**<http://schemas.microsoft.com/wbem/wsman/1/windows/shell/cmd>**

This ResourceURI is used for all operations described in this specification. In the case of wxf:Create and wsen:Enumerate, this is the only part of the WS-Management address. If specific shell instances are targeted, they use the following selector format:

```
(1)      <wsman:SelectorSet>
(2)          <wsman:Selector Name="ShellId">
(3)              uuid:2164cfa5-d192-4910-baf4-1aff787ca5f6
(4)          </wsman:Selector>
(5)      </wsman:SelectorSet>
```

...in which the uuid: URI contains the identity of the shell instance. This is the same value as used in the ShellId for the schema returned by wxf:Get and wsen:Enumerate.

## 6.3 wxf:Create Usage

During creation of a new shell, the environment variables can be defined in terms of preexisting environment variables using the normal %varname% syntax available in Windows:

```
(6)      <rsp:Shell>
(7)          <rsp:WorkingDirectory> c:\mydir </rsp:WorkingDirectory>
(8)          <rsp:Environment>
(9)              <rsp:Variable Name="path"> %path%;c:\temp </rsp:Variable>
(10)         </rsp:Environment>
(11)     ....
```

The following WS-Management option is used to control whether the user profile is fully loaded while the shell is being initialized using wxf:Create:

```
(12)     <wsman:OptionSet>
(13)         <wsman:Option Name="LoadUserProfile">
(14)             true | false
(15)         </wsman:Option>
(16)     </wsman:OptionSet>
```

If the option is not set, the default behavior will be as if **true** had been set. In all cases, the user will be impersonated, but loading the user profile creates a more exact replica of the user environment, at a runtime cost.

## 6.4 Reliability Issues

Because messages may not be successfully delivered in either requests or responses, it is important for both sides of the message model to avoid duplicate execution of commands or dropped information.

The Windows implementation specifically is able to deal with replay of the most recent pair of request-response messages. Specifically, if a client issues a request and receives no reply, there are two possibilities:

- (1) The request was received and processed, but the response was delayed or lost
- (2) The request was never received

In either case, the client MAY replay the most recent message that was sent, *precisely* as it was originally sent, complete with the original MessageID.

The service is designed to cache the entire most recent response and associate it with the original request MessageID. If a request is received which has a MessageID which matches the most recent known MessageID, the service will simply replay the reply it attempted to send the first time. It retains this reply in a cache until a new request arrives with a different MessageID, in which case the new response is cached. In this manner, the most recent pair of messages may be replayed any number of times until the client has received the response and moves to the next message. This mimics the behavior that would occur in datagram-based scenarios in which duplicate messages might arrive due to router replication. The MessageID and RelatesTo values are used to ensure that only a single logical pair of exchanges occurs, even though more than one copy of the physical message may have occurred on the wire.

All timeout-related problems and retries work within this basic mechanism.

## 7.0 The Schema and WSDL Files

The schema for this specification can be located at

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell.xsd>

The WSDL for this specification can be located at

<http://schemas.microsoft.com/wbem/wsman/1/windows/shell.wsdl>

## 8.0 References

[RFC 2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, March 1997. (See <http://www.ietf.org/rfc/rfc2119.txt>.)

**[SOAP 1.1]**

Box, D., et al, "Simple Object Access Protocol (SOAP) 1.1," May 2000. (See <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.)

**[SOAP 1.2]**

Gudgin, M., et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. (See <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>.)

**[WS-Addressing]**

Box, D., et al, "Web Services Addressing (WS-Addressing)," August 2004. (See <http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/>.)

**[WS-Enumeration]**

Alexander, J., et al, "Web Service Enumeration (WS-Enumeration)," September 2004. (See <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-enumeration.pdf>.)

**[WS-Management]**

"Web Services for Management (WS-Management)," Version 1.0.0a, April 2006. (See [http://www.dmtf.org/standards/published\\_documents/DSP0226.pdf](http://www.dmtf.org/standards/published_documents/DSP0226.pdf))

**[WS-Transfer]**

Alexander, J., et al, "Web Service Transfer (WS-Transfer)," September 2004. (See <http://schemas.xmlsoap.org/ws/2004/09/transfer>.)