[MS-PSRP]:
PowerShell Remoting Protocol

Release: Thursday, February 13, 2014

Revision Summary

Date	Revision History	Revision Class	Comments
12/05/2008	0.1	Major	Initial Availability
01/16/2009	1.0	Major	Updated and revised the technical content.
02/27/2009	1.0.1	Editorial	Revised and edited the technical content.
04/10/2009	2.0	Major	Updated and revised the technical content.
05/22/2009	3.0	Major	Updated and revised the technical content.
07/02/2009	4.0	Major	Updated and revised the technical content.
08/14/2009	5.0	Major	Updated and revised the technical content.
09/25/2009	6.0	Major	Updated and revised the technical content.
11/06/2009	7.0	Major	Updated and revised the technical content.
12/18/2009	8.0	Major	Updated and revised the technical content.
01/29/2010	9.0	Major	Updated and revised the technical content.
03/12/2010	9.0.1	Editorial	Revised and edited the technical content.
04/23/2010	9.0.2	Editorial	Revised and edited the technical content.
06/04/2010	9.1	Minor	Updated the technical content.
07/16/2010	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	9.1	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
06/17/2011	9.2	Minor	Clarified the meaning of the technical content.
09/23/2011	10.0	Major	Significantly changed the technical content.
12/16/2011	11.0	Major	Significantly changed the technical content.
03/30/2012	11.1	Minor	Clarified the meaning of the technical content.
07/12/2012	11.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	12.0	Major	Significantly changed the technical content.
01/31/2013	12.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	13.0	Major	Significantly changed the technical content.
11/14/2013	13.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/13/2014	14.0	Major	Significantly changed the technical content.

Contents

1 Introduction

This document specifies the PowerShell Remoting Protocol. The PowerShell Remoting Protocol encodes messages prior to sending them over the Web Services Management Protocol Extensions for Windows Vista [MS-WSMV] layer.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

base64
globally unique identifier (GUID)
little-endian

The following terms are specific to this document:

command: Any entity which can be executed in PowerShell.

command name: A sequence of characters used by the server higher layers to identify a command on the server. A command may contain a namespace component (fully-qualified command name) or may not (unqualified command name). The syntax for indicating a namespace qualified command is server-dependent.

command namespace: A context used by the server higher layers to disambiguate command names. This context may be empty and commands may be resolved with an empty context (no namespace qualifier).

decoding: The reversal of the **encoding** process, used by a **PowerShell client** or **PowerShell server** to correctly interpret a received object.

defragmentation: The construction of a PowerShell Remoting Protocol Message from fragments

deserialization: The mechanism by which PowerShell constructs an object from its XML representation.

encoding: The annotation of an object with metadata so that it can be sent to a **PowerShell client** or **PowerShell server**.

fragmentation: The breaking down of a PowerShell Remoting Protocol Message into fragments, with additional metadata such that fragments can be sequenced and sent using **WinRM** and reassembled (defragmented) at the receiving end.

host: An interface between a PowerShell **runspace** and a user capable of responding to the host method calls specified in section 2.2.3.17. For more details on host functionality, see sections 2.2.3.17 and 2.2.6.

nested pipeline: A pipeline that is executed in a **runspace** that is already running a pipeline. The original **runspace** pipeline is suspended while the **nested pipeline** runs and is resumed after the **nested pipeline** completes.

object: The root of the type hierarchy. For more information, see [ECMA-335].

pipeline: An ordered collection of **commands**, with the output of one **command** passed as input to the next.

PowerShell client: Any process that tries to initiate PowerShell **commands** using PowerShell remoting.

PowerShell server: Any process that accepts **commands** from a **PowerShell client** process (via **WinRM**).

runspace: An entity capable of running one (and only one) **pipeline** of **commands**.

RunspacePool: A group of **runspaces** with the same characteristics which can be opened and closed as needed.

serialization: A mechanism by which PowerShell converts an object into an XML representation.

session: The operational environment in which the PowerShell shell and its **commands** execute.

ScriptBlock: Represents a block of PowerShell script.

steppable pipeline: A special pipeline type that processes input objects one at a time, in a single step per object.

WinRM: The Windows Remote Management (WinRM) is the Microsoft implementation of **WS-MAN** protocol [MS-WSMV].

WS-MAN: The Web Services Management Protocol, as specified in [MS-WSMV].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

A reference marked "(Archived)" means that the reference document was either retired and is no longer being maintained or was replaced with a new document that provides current implementation details. We archive our documents online [Windows Protocol].

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[DMTF-DSP0226] Distributed Management Task Force, Inc., "Web Services for Management (WS-Management) Specification", version 1.0.0, February 2008, http://dmtf.org/sites/default/files/standards/documents/DSP0226_1.0.0.pdf

[ECMA-335] ECMA International, "Common Language Infrastructure (CLI) Partitions I to VI", ECMA-335, June 2006, http://www.ecma-international.org/publications/standards/Ecma-335.htm

[FIPS197] FIPS PUBS, "Advanced Encryption Standard (AES)", FIPS PUB 197, November 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, http://ieeexplore.ieee.org/servlet/opac?punumber=2355

[MS-NRBF] Microsoft Corporation, ".NET Remoting: Binary Format Data Structure".

[MS-NRTP] Microsoft Corporation, ".NET Remoting: Core Protocol".

[MS-WSMV] Microsoft Corporation, "Web Services Management Protocol Extensions for Windows Vista".

[PKCS1] RSA Laboratories, "PKCS #1: RSA Cryptography Standard", PKCS #1, Version 2.1, June 2002, http://www.rsa.com/rsalabs/node.asp?id=2125

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981, http://www.ietf.org/rfc/rfc791.txt

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC2396] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998, http://www.ietf.org/rfc/rfc2396.txt

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC2732] Hinden, R., Carpenter, B., and Masinter, L., "Format for Literal IPv6 Addresses in URL's", RFC 2732, December 1999, http://www.ietf.org/rfc/rfc2732.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.ietf.org/rfc/rfc2818.txt

[RFC3447] Jonsson, J., and Kaliski, B., "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003, http://www.ietf.org/rfc/rfc3447.txt

[RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003, http://www.ietf.org/rfc/rfc3548.txt

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, http://www.ietf.org/rfc/rfc4122.txt

[SOAP1.2-1/2003] Gudgin, M., Hadley, M., Mendelsohn, N., et al., "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation, June 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624

[SP800-38A] National Institute of Standards and Technology. "Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques", December 2001, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[WSAddressing] Box, D., Christensen, E., Ferguson, D., et al., "Web Services Addressing (WS-Addressing)", August 2004, http://www.w3.org/Submission/ws-addressing/

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, August 2006, http://www.w3.org/TR/2006/REC-xml-20060816/

[XMLNS-2ED] World Wide Web Consortium, "Namespaces in XML 1.0 (Second Edition)", August 2006, http://www.w3.org/TR/2006/REC-xml-names-20060816/

[XMLSCHEMA2] Biron, P.V., and Malhotra, A., Eds., "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MSFT-POWERSHELL] Microsoft Corporation, "Windows PowerShell Language Specification Version 2.0", http://www.microsoft.com/download/en/details.aspx?id=9706

1.3 Overview

Client applications use the PowerShell Remoting Protocol (PSRP) to send **pipelines** of commands to a server system over a network for execution by the server.

The PSRP is a stateful protocol where clients establish a **session** with a server and use that session to send structured pipelines of abstract commands to the server for execution. PSRP imposes state to maintain an authentication context and cryptographic operations as well as give higher layers on the server a way to preserve session state associated with the commands being executed on the server across multiple pipeline executions. The state associated with commands is contained in an abstraction informally called a "runspace".

Only one pipeline can be executed in a runspace at a time. A server allows the client to execute multiple pipelines concurrently by providing a bounded pool of runspaces formally called a RunspacePool. The RunspacePool bounds are specified by the client at session initiation time.

Note that the PSRP provides no mechanism for specifying which runspace in a pool is to be used when executing a pipeline. The only addressable construct is the RunspacePool. As a consequence, scenarios where pipelines depend on the runspace containing specific state established by previous pipelines must use a RunspacePool size of 1.

The PSRP pipeline is similar to the UNIX concept of a pipeline with the difference that PSRP represents pipeline commands and parameters in an abstract structured way, independent of any higher-layer syntax or semantics using an XML representation. A pipeline contains an ordered sequence of commands as well as parameters and arguments associated with each command. Other than classifying pipeline elements as commands, parameters, and typed arguments, the PSRP leaves all other semantic command interpretation to the higher layer responsible for actually executing the pipeline. For example, an implementation of the higher layer may translate the PSRP pipeline representation into UNIX syntax to be executed by the UNIX shell. An alternate implementation may translate the pipeline into a series of Web service requests orchestrated by the server higher layers.

After the client submits a pipeline line for execution, it may optionally send a sequence of input objects to the pipeline on the server. The server will pass this input to the higher layer where it should be used as input to the first command in the pipeline. The higher layers should orchestrate the execution of commands such that the output of one command in the pipeline becomes the input of the next command in an implementation-dependent way. Any objects emitted by the final command in the pipeline will be sent from the server back to the client.

In addition, the PSRP provides for the following capabilities:

- A mechanism for the client to request that a pipeline currently executing on the server be stopped.
- An "error" stream that will contain error objects generated by commands in the pipeline during execution.
- A set of messages that the server may send to the client allowing the server to request or display additional information such as progress messages, warnings, requests for confirmation of an operation, or requests for additional information. These messages are called the host methods and are sent from the server to the client. The client implementation may honor these messages by taking appropriate actions (displaying messages, sending the requested information). It is also perfectly acceptable for the client to ignore all display requests and fail all information requests from the server.
- A mechanism for the client to discover the set of available commands that may be executed on the server. The information returned by this mechanism is sufficient for the client to create structurally valid pipelines. This information is not guaranteed to remain valid after it has been retrieved as the set of commands exposed by the server is allowed to change at any time.

1.4 Relationship to Other Protocols

The PowerShell Remoting Protocol uses the Web Services Management Protocol Extensions for Windows Vista [MS-WSMV] to establish a connection and transfer data between the client and the server. [MS-WSMV] is built on top of the following protocols.

- SOAP (Version 1.2) [SOAP1.2-1/2003]
- The Hypertext Transfer Protocol (HTTP/1.1) [RFC2616] or HTTP Over TLS [RFC2818]
- The Transmission Control Protocol [RFC793]
- The Internet Protocol [RFC791]

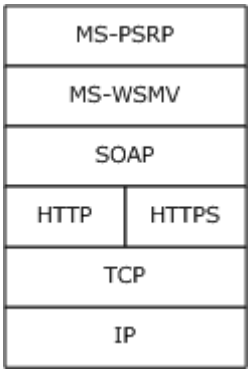


Figure 1: Relationship of PowerShell Remoting Protocol to other protocols

1.5 Prerequisites/Preconditions

A **PowerShell client** can only communicate with a **PowerShell server** using the PowerShell Remoting Protocol in the following circumstances.

- The PowerShell server has implemented the server role of the PowerShell Remoting Protocol to communicate with the PowerShell client.

- The PowerShell server has implemented the server role of remote shell operations specified in [MS-WSMV].

1.6 Applicability Statement

The PowerShell Remoting Protocol is required whenever a user wants to execute PowerShell **commands** on a server from a client.

1.7 Versioning and Capability Negotiation

The PowerShell Remoting Protocol is based on the Web Services Management Protocol Extensions for Windows Vista [MS-WSMV].

- **Supported Transports:** The PowerShell Remoting Protocol is implemented on top of the WS-WSMV protocol, as discussed in section 3.1.5.3
- **Protocol Versions:** The PowerShell Remoting Protocol supports the following explicit dialects: WSMAN1.1. These dialects are defined in section 3.1.5.3.1
- **PowerShell Protocol Version:** The PowerShell Remoting Protocol requires the option named protocolversion to be present in the OptionSet of the /Create message. This option is described in section 3.1.5.3.1 and is used by the server to send messages to the client in a format that client can understand.
- **Capability Negotiation:** The PowerShell Remoting Protocol does explicit capability negotiation as specified in sections 3.1.5.4.1 and 3.2.5.4.1.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The PowerShell remoting protocol uses remote shell operations, supported by the Web Services Management Protocol Extensions for Windows Vista [MS-WSMV], for transporting data between PowerShell clients and PowerShell servers. These remote shell operations are specified in [MS-WSMV], section 3.1.4.

For more information about how transport is done on PowerShell clients and on PowerShell servers, see the general protocol rules specified in sections 3.1.5.1 and 3.1.5.2.

2.2 Message Syntax

All messages are **little-endian**, except where otherwise specified.

2.2.1 PowerShell Remoting Protocol Message

The structure of a PowerShell Remoting Protocol Message is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Destination																															
MessageType																															
RPID																															
...																															
...																															
...																															
PID																															
...																															
...																															
...																															
Data (variable)																															
...																															

Destination (4 bytes): The destination of this message.

Possible values.

Value	Meaning
0x00000001	The message is targeted to a PowerShell client.
0x00000002	The message is targeted to a PowerShell server.

MessageType (4 bytes): The type of message. The value of this field specifies what action MUST be taken by the PowerShell client or PowerShell server upon receipt.

Possible values.

Value	Meaning
SESSION_CAPABILITY 0x00010002	Session capability. **Direction:** Bidirectional (PowerShell client to PowerShell server or PowerShell server to PowerShell client). **Target:** **RunspacePool**.
INIT_RUNSPACEPOOL 0x00010004	Initialize RunspacePool. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
PUBLIC_KEY 0x00010005	Public key. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
ENCRYPTED_SESSION_KEY 0x00010006	Encrypted session key. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
PUBLIC_KEY_REQUEST 0x00010007	Public key request. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
CONNECT_RUNSPACEPOOL 0x00010030	Connect to a RunspacePool. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
RUNSPACE_INIT_DATA 0x00010031	RunspacePool initialization data. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
SET_MAX_RUNSPACES 0x00021002	Set maximum runspaces in a RunspacePool. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
SET_MIN_RUNSPACES 0x00021003	Set minimum runspaces in a RunspacePool. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
RUNSPACE_AVAILABILITY 0x00021004	A response to either set maximum **runspaces** or set minimum runspaces in a RunspacePool or request for

Value	Meaning
	available runspaces in a RunspacePool. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
RUNSPACEPOOL_STATE 0x00021005	State information of a RunspacePool. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
CREATE_PIPELINE 0x00021006	Create a PowerShell and invoke it in the specified RunspacePool. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
GET_AVAILABLE_RUNSPACES 0x00021007	Get the number of available runspaces in a RunspacePool. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
USER_EVENT 0x00021008	Report a user-defined event from a remote runspace. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
APPLICATION_PRIVATE_DATA 0x00021009	Application private data: data private to the application using the PowerShell remoting protocol on the server and client, which is passed by the protocol without interpretation. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
GET_COMMAND_METADATA 0x0002100A	Get command metadata for commands available in a RunspacePool. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
RUNSPACEPOOL_HOST_CALL 0x00021100	Method call on the **host** associated with the RunspacePool on the server. **Direction:** PowerShell server to PowerShell client. **Target:** RunspacePool.
RUNSPACEPOOL_HOST_RESPONSE 0x00021101	Response from a host call executed on the PowerShell client RunspacePool's host. **Direction:** PowerShell client to PowerShell server. **Target:** RunspacePool.
PIPELINE_INPUT 0x00041002	Input to a PowerShell on the server. **Direction:** PowerShell client to PowerShell server. **Target:** pipeline.
END_OF_PIPELINE_INPUT 0x00041003	Close the input collection for the PowerShell on the server. **Direction:** PowerShell client to PowerShell server. **Target:** pipeline.

Value	Meaning
PIPELINE_OUTPUT 0x00041004	Output of a PowerShell on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline.
ERROR_RECORD 0x00041005	Error record from a PowerShell on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline.
PIPELINE_STATE 0x00041006	State information of a PowerShell on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline or RunspacePool.
DEBUG_RECORD 0x00041007	Debug record from a PowerShell on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline.
VERBOSE_RECORD 0x00041008	Verbose record from a PowerShell on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline.
WARNING_RECORD 0x00041009	Warning record from a PowerShell on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline.
PROGRESS_RECORD 0x00041010	Progress record from a PowerShell on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline.
PIPELINE_HOST_CALL 0x00041100	Method call on the host associated with the pipeline invocation settings on the server. **Direction:** PowerShell server to PowerShell client. **Target:** pipeline.
PIPELINE_HOST_RESPONSE 0x00041101	Response from a host call executed on the PowerShell client's host. **Direction:** PowerShell client to PowerShell server. **Target:** pipeline.

RPID (16 bytes): A **globally unique identifier (GUID)** specifying the instance ID of the RunspacePool on the PowerShell client.

PID (16 bytes): A GUID specifying the instance ID of the pipeline on the PowerShell client.

Data (variable): The contents of this field are determined by the **MessageType** field, and are fully specified in section 2.2.2.

2.2.2 Message Types

The following subsections specify the **Data** field for each type of PowerShell Remoting Protocol message.

2.2.2.1 SESSION_CAPABILITY Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a SESSION_CAPABILITY message when the **MessageType** field has a value of 0x00010002.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- Version of PowerShell
  - Property name: PSVersion
  - Property type: Version (see section 2.2.5.1.21)
- Version of the PowerShell remoting protocol (see section 3.1.5.3.1)
  - Property name: protocolversion
  - Property type: Version (see section 2.2.5.1.21)
- Version of PowerShell **serialization**
  - Property name: SerializationVersion
  - Property type: Version (see section 2.2.5.1.21)
- Time zone of the client
  - Property name: TimeZone
  - Property type: TimeZone (see section 2.2.3.10) or Null value (see section 2.2.5.1.20)
  - This property is optional and MAY be omitted.

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
  <MS>
    <Version N="protocolversion">2.2</Version>
    <Version N="PSVersion">2.0</Version>
    <Version N="SerializationVersion">1.1.0.1</Version>
    <BA
N="TimeZone">AAEAAAD/////AQAAAAAAAAAEAQAAABxTeXN0ZW0uQ3VycmVudFN5c3RlbVRpbWVab25lBAAAABdtX0Nh
Y2hlZERheWxpZ2h0Q2hhbmdlcw1tX3RpY2tzT2Zmc2V0Dm1fc3RhbmRhcmROYW1lDm1fZGF5bGlnaHROYW1lAwABARxTe
XN0ZW0uQ29sbGVjdGlvbnMuSGFzaHRhYmxlCQkCAAAAAMDc8bz///8KCgQCAAAAHFN5c3RlbS5Db2xsZWN0aW9ucy5IYX
NodGFibGUHAAAACkxvYWRGYWN0b3IHVmVyc2lvbghDb21wYXJlchBIYXNoQ29kZVByb3ZpZGVyCEhhc2hTaXplBEtleXM
GVmFsdWVzAAADAwAFBQsIHFN5c3RlbS5Db2xsZWN0aW9ucy5JQ29tcGFyZXIkU3lzdGVtLkNvbGxlY3Rpb25zLklIYXNo
Q29kZVByb3ZpZGVyCOxROD8BAAAACgoLAAAACQMAAAAJBAAAABADAAAAAQAAAAgI2QcAABAEAAAAAQAAAAkFAAAABAUAA
AAhU3lzdGVtLkdsb2JhbGl6YXRpb24uRGF5bGlnaHRUaW1lAwAAAAdtX3N0YXJ0BW1fZW5kB21fZGVsdGEAAAANDQwAkO
q4qG3LiAAQOyeuKMyIAGjEYQgAAAAL</BA>
  </MS>
</Obj>
```

2.2.2.2 INIT_RUNSPACEPOOL Message

The **Data** field of a PowerShell Remoting Protocol Message specifies an INIT_RUNSPACEPOOL message when the **MessageType** field has a value of 0x00010004.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- Minimum number of runspaces in the RunspacePool
  - Property name: MinRunspaces
  - Property type: Signed int (see section 2.2.5.1.11)
- Maximum number of runspaces in the RunspacePool
  - Property name: MaxRunspaces
  - Property type: Signed int (see section 2.2.5.1.11)
- Thread options provided by the higher layer; PSRP MUST NOT interpret this data.
  - Property name: PSThreadOptions
  - Property type: PSThreadOptions (see section 2.2.3.6)
- Apartment state provided by the higher layer; PSRP MUST NOT interpret this data.
  - Property name: ApartmentState
  - Property type: ApartmentState (see section 2.2.3.7)
- Host information
  - Property name: HostInfo
  - Property type: HostInfo (see section 2.2.3.14)
- Application arguments provided by the higher layer; PSRP MUST NOT interpret this data.
  - Property name: ApplicationArguments
  - Property type: Primitive Dictionary (see section 2.2.3.18) or Null Value (see section 2.2.5.1.20)

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="1">
<MS>
  <I32 N="MinRunspaces">1</I32>
  <I32 N="MaxRunspaces">1</I32>
  <Obj N="PSThreadOptions" RefId="2">
    <TN RefId="0">
      <T>System.Management.Automation.Runspaces.PSThreadOptions</T>
      <T>System.Enum</T>
```

```
    <T>System.ValueType</T>
    <T>System.Object</T>
  </TN>
  <ToString>Default</ToString>
  <I32>0</I32>
</Obj>
<Obj N="ApartmentState" RefId="3">
  <TN RefId="1">
    <T>System.Threading.ApartmentState</T>
    <T>System.Enum</T>
    <T>System.ValueType</T>
    <T>System.Object</T>
  </TN>
  <ToString>MTA</ToString>
  <I32>1</I32>
</Obj>
<Obj N="HostInfo" RefId="4">
  <MS>
    <Obj N="_hostDefaultData" RefId="5">
      <MS>
        <Obj N="data" RefId="6">
          <TN RefId="2">
            <T>System.Collections.Hashtable</T>
            <T>System.Object</T>
          </TN>
          <DCT>
            <En>
              <I32 N="Key">9</I32>
              <Obj N="Value" RefId="7">
                <MS>
                  <S N="T">System.String</S>
                  <S N="V">Windows PowerShell V2 (MS Internal Only)</S>
                </MS>
              </Obj>
            </En>
            <En>
              <I32 N="Key">8</I32>
              <Obj N="Value" RefId="8">
                <MS>
                  <S N="T">System.Management.Automation.Host.Size</S>
                  <Obj N="V" RefId="9">
                    <MS>
                      <I32 N="width">181</I32>
                      <I32 N="height">98</I32>
                    </MS>
                  </Obj>
                </MS>
              </Obj>
            </En>
            <En>
              <I32 N="Key">7</I32>
              <Obj N="Value" RefId="10">
                <MS>
                  <S N="T">System.Management.Automation.Host.Size</S>
                  <Obj N="V" RefId="11">
                    <MS>
                      <I32 N="width">120</I32>
                      <I32 N="height">98</I32>
                    </MS>
```

```
      </Obj>
    </MS>
  </Obj>
</En>
<En>
  <I32 N="Key">6</I32>
  <Obj N="Value" RefId="12">
    <MS>
      <S N="T">System.Management.Automation.Host.Size</S>
      <Obj N="V" RefId="13">
        <MS>
          <I32 N="width">120</I32>
          <I32 N="height">79</I32>
        </MS>
      </Obj>
    </MS>
  </Obj>
</En>
<En>
  <I32 N="Key">5</I32>
  <Obj N="Value" RefId="14">
    <MS>
      <S N="T">System.Management.Automation.Host.Size</S>
      <Obj N="V" RefId="15">
        <MS>
          <I32 N="width">120</I32>
          <I32 N="height">3000</I32>
        </MS>
      </Obj>
    </MS>
  </Obj>
</En>
<En>
  <I32 N="Key">4</I32>
  <Obj N="Value" RefId="16">
    <MS>
      <S N="T">System.Int32</S>
      <I32 N="V">25</I32>
    </MS>
  </Obj>
</En>
<En>
  <I32 N="Key">3</I32>
  <Obj N="Value" RefId="17">
    <MS>
      <S N="T">System.Management.Automation.Host.Coordinates</S>
      <Obj N="V" RefId="18">
        <MS>
          <I32 N="x">0</I32>
          <I32 N="y">0</I32>
        </MS>
      </Obj>
    </MS>
  </Obj>
</En>
<En>
  <I32 N="Key">2</I32>
  <Obj N="Value" RefId="19">
    <MS>
```

```
                    <S N="T">
                      System.Management.Automation.Host.Coordinates
                    </S>
                    <Obj N="V" RefId="20">
                      <MS>
                        <I32 N="x">0</I32>
                        <I32 N="y">4</I32>
                      </MS>
                    </Obj>
                  </MS>
                </Obj>
              </En>
              <En>
                <I32 N="Key">1</I32>
                <Obj N="Value" RefId="21">
                  <MS>
                    <S N="T">System.ConsoleColor</S>
                    <I32 N="V">5</I32>
                  </MS>
                </Obj>
              </En>
              <En>
                <I32 N="Key">0</I32>
                <Obj N="Value" RefId="22">
                  <MS>
                    <S N="T">System.ConsoleColor</S>
                    <I32 N="V">6</I32>
                  </MS>
                </Obj>
              </En>
            </DCT>
          </Obj>
        </MS>
      </Obj>
      <B N="_isHostNull">false</B>
      <B N="_isHostUINull">false</B>
      <B N="_isHostRawUINull">false</B>
      <B N="_useRunspaceHost">false</B>
    </MS>
  </Obj>
  <Nil N="ApplicationArguments" />
  </MS>
</Obj>
```

2.2.2.3 PUBLIC_KEY Messsage

The **Data** field of a PowerShell Remoting Protocol message specifies a PUBLIC_KEY message when the **MessageType** field has a value of 0x00010005.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- 2048-bit public key of a RSA public key pair [PKCS1] as represented in this section, encoded in **base64** format.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0x06								0x02								0x00								0x00							
0x00								0xA4								0x00								0x00							
0x52								0x53								0x41								0x31							
0x00								0x08								0x00								0x00							
Public Exponent																															
Modulus																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(Modulus cont'd for 56 rows)																															

Public Exponent (4 bytes): A 32-bit unsigned number in little-endian format specifying the public exponent of the key pair, referred to as **e** in [RFC3447] section 2.

Modulus (256 bytes): The RSA modulus, referred to as **n** in [RFC3447] section 2. The modulus MUST be encoded in little-endian format.

- Property name: PublicKey.
- Property type: String (see section 2.2.5.1.1).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
  <MS>
    <S N="PublicKey">
```

```
BgIAAACkAABSU0ExAAgAAAEAAQBxLtiI7U4s5gkx4zzFaRyhCgTwSYWBdxx6MfjJMXcuLewnq7RvIo6yfgcN2s8FXrelH
s8y34S0fdvM/fbSXjaacKOQoLVvOgyVf1x7EODpDADW2Tj9RIz52hcsVzNFfkfT4EhMvcJbDIqtEwIF6BmjHc5yNPsywT
FD6QU50BIySeV7IT3qhjxihQEbMt/shf0DcFX07JIs37FPPZpesaviyG3RZjhQbfCbJ66vlea+1ocVYgqM7W98ZIeHlRT
2XhrPSD+hwriUcfG3oOJIILpo2acpAxcz8KCEOkpocH4wA/IgF+9UcaeanOkBXqK3xc9LPtVuQ7otZYa+zvrTZXe4
    </S>
  </MS>
</Obj>
```

2.2.2.4 ENCRYPTED_SESSION_KEY Message

The **Data** field of a PowerShell Remoting Protocol message specifies an ENCRYPTED_SESSION_KEY message when the **MessageType** field has a value of 0x00010006.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- 256-bit symmetric key for AES encryption scheme [FIPS197] encrypted using the public key from the PUBLIC_KEY messsage (see section 2.2.2.3) using the RSAES-PKCS-v1_5 encryption scheme specified in [RFC3447] section 7.2, and encoded in base64 format.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0x01								0x02								0x00								0x00							
0x10								0x66								0x00								0x00							
0x00								0xa4								0x00								0x00							
Encrypted Key																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(Modulus cont'd for 56 rows)																															

  - Property name: EncryptedSessionKey.

- Property type: String (see section 2.2.5.1.1).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
  <MS>
    <S N="EncryptedSessionKey">
AQIAABBmAAAApAAAgY6iLhsPXjMGza6Rc6JeEfezwTaZjJhm+gj55YRVzv6QTyRkl3j9XuESv5WhNwHHZD0pAwDC5iZcx
FCKtZ4PSuBIy6EULAuvxUCvREZ2NueMLUzbOaLviFc4Y2Qf9rPEBfjK/iKyudKTiF4bY92RTZxoxVECaT4Z9EJI4QyigC
IUfjY7oXzcntkc09Its+v9HgoQY50qXCtqB+r1Npdx3gYPvtuTPsRGGPlmKnns6gVALeh8Tw/FPo8EMk+oGpfAUZjhxcN
pmrniujs8UTlDzV8JWa/sEjrpewEGTBRWs0AQ3yEj2ALZzpwDa+bHhSp8TtJV+V6ZN7MvTX2igcAwQA==
    </S>
  </MS>
</Obj>
```

2.2.2.5 PUBLIC_KEY_REQUEST Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a PUBLIC_KEY_REQUEST message when the **MessageType** field has a value of 0x00010007.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing an empty String (see section 2.2.5.1.1); that is, a string containing zero characters.

Example:

```
<S></S>
```

2.2.2.6 SET_MAX_RUNSPACES Message

The **Data** field of a PowerShell Remoting Protocol message specifies a SET_MAX_RUNSPACES message when the **MessageType** field has a value of 0x00021002.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- Call ID
  - Property name: ci.
  - Property type: Signed long (see section 2.2.5.1.13).
- Maximum number of runspaces in the RunspacePool
  - Property name: MaxRunspaces.
  - Property type: Signed int (see section 2.2.5.1.11)

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="5">
<MS>
  <I32 N="MaxRunspaces">3</I32>
  <I64 N="ci">1</I64>
</MS>
</Obj>
```

2.2.2.7 SET_MIN_RUNSPACES Message

The **Data** field of a PowerShell Remoting Protocol message specifies a SET_MIN_RUNSPACES message when the **MessageType** field has a value of 0x00021003.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- Call ID
  - Property name: ci.
  - Property type: Signed long (see section 2.2.5.1.13).
- Minimum number of runspaces in the RunspacePool.
  - Property name: MinRunspaces.
  - Property type: Signed int (see section 2.2.5.1.11)

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="6">
<MS>
  <I32 N="MinRunspaces">2</I32>
  <I64 N="ci">2</I64>
</MS>
</Obj>
```

2.2.2.8 RUNSPACE_AVAILABILITY Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a RUNSPACE_AVAILABILITY message when the **MessageType** field has a value of 0x00021004.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- Call ID
  - Property name: ci.
  - Property type: Signed long (see section 2.2.5.1.13).
- Response

- Property name: SetMinMaxRunspacesResponse.
- Property type: Boolean (see section 2.2.5.1.3) if the response is to a SET_MAX_RUNSPACES or SET_MIN_RUNSPACES message, or a Signed Long (see section 2.2.5.1.13) if the response is to a GET_AVAILABLE_RUNSPACES message.

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="2">
  <MS>
    <B N="SetMinMaxRunspacesResponse">true</B>
    <I64 N="ci">1</I64>
  </MS>
</Obj>
```

2.2.2.9 RUNSPACEPOOL_STATE Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a RUNSPACEPOOL_STATE message when the **MessageType** field has a value of 0x00021005.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- RunspacePool state information
  - Property name: RunspaceState.
  - Property type: RunspacePoolState (see section 2.2.3.4).
- Optional error information (included only if this message is triggered by an error).
  - Property name: ExceptionAsErrorRecord.
  - Property type: ErrorRecord (see section ErrorRecord). The FullyQualifiedErrorId property SHOULD have a value of "RemoteRunspaceStateInfoReason".

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="1">
  <MS>
    <I32 N="RunspaceState">2</I32>
  </MS>
</Obj>
```

2.2.2.10 CREATE_PIPELINE Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a CREATE_PIPELINE message when the **MessageType** field has a value of 0x00021006.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- Whether the PowerShell pipeline will take input
  - Property name: NoInput.
  - Property type: Boolean (see section 2.2.5.1.3).
- Apartment state provided by the higher layer; PSRP MUST NOT interpret this data.
  - Property name: ApartmentState.
  - Property type: ApartmentState (see section 2.2.3.7).
- Stream options that indicate how PowerShell MUST treat messages from debug, verbose, warning and error streams in the remote invocation scenario
  - Property name: RemoteStreamOptions.
  - Property type: RemoteStreamOptions (see section 2.2.3.8).
- Boolean indicating if the higher layer should add the pipeline being executed to the history field of the runspace. The PSRP layer MUST NOT interpret this data.
  - Property name: AddToHistory.
  - Property type: Boolean (see section 2.2.5.1.3).
- Host information
  - Property name: HostInfo.
  - Property type: HostInfo (see section 2.2.3.14).
- Description of the PowerShell pipeline to create
  - Property name: PowerShell.
  - Property type: PowerShell pipeline (see section 2.2.3.11)
- Boolean indicating whether the higher layer is to run the pipeline in nested or steppable mode. The PSRP layer MUST NOT interpret this data.
  - Property name: IsNested.
  - Property type: Boolean (see section 2.2.5.1.3).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
<MS>
  <Obj N="PowerShell" RefId="1">
    <MS>
      <Obj N="Cmds" RefId="2">
```

```
          <TN RefId="0">
            <T>System.Collections.Generic.List`1[[System.Management.Automation.PSObject,
  System.Management.Automation, Version=1.0.0.0, Culture=neutral,
  PublicKeyToken=31bf3856ad364e35]]</T>
            <T>System.Object</T>
          </TN>
          <LST>
            <Obj RefId="3">
              <MS>
                <S N="Cmd">123 </S>
                <B N="IsScript">true</B>
                <Nil N="UseLocalScope" />
                <Obj N="MergeMyResult" RefId="4">
                  <TN RefId="1">
                    <T>System.Management.Automation.Runspaces.PipelineResultTypes</T>
                    <T>System.Enum</T>
                    <T>System.ValueType</T>
                    <T>System.Object</T>
                  </TN>
                  <ToString>None</ToString>
                  <I32>0</I32>
                </Obj>
                <Obj N="MergeToResult" RefId="5">
                  <TNRef RefId="1" />
                  <ToString>None</ToString>
                  <I32>0</I32>
                </Obj>
                <Obj N="MergePreviousResults" RefId="6">
                  <TNRef RefId="1" />
                  <ToString>None</ToString>
                  <I32>0</I32>
                </Obj>
                <Obj N="MergeError" RefId="7">
                  <TNRef RefId="1" />
                  <ToString>None</ToString>
                  <I32>0</I32>
                </Obj>
                <Obj N="MergeWarning" RefId="8">
                  <TNRef RefId="1" />
                  <ToString>None</ToString>
                  <I32>0</I32>
                </Obj>
                <Obj N="MergeVerbose" RefId="9">
                  <TNRef RefId="1" />
                  <ToString>None</ToString>
                  <I32>0</I32>
                </Obj>
                <Obj N="MergeDebug" RefId="10">
                  <TNRef RefId="1" />
                  <ToString>None</ToString>
                  <I32>0</I32>
                </Obj>
                <Obj N="Args" RefId="11">
                  <TNRef RefId="0" />
                  <LST>
                    <Obj RefId="7b">
                      <MS>
                        <Nil N="N" />
                        <S N="V">powershell.exe</S>
```

```
                    </MS>
                  </Obj>
                </LST>
              </Obj>
            </MS>
          </Obj>
        </LST>
      </Obj>
      <B N="IsNested">false</B>
    </MS>
  </Obj>
  <B N="NoInput">true</B>
  <Obj N="ApartmentState" RefId="12">
    <TN RefId="2">
      <T>System.Threading.ApartmentState</T>
      <T>System.Enum</T>
      <T>System.ValueType</T>
      <T>System.Object</T>
    </TN>
    <ToString>MTA</ToString>
    <I32>1</I32>
  </Obj>
  <Obj N="RemoteStreamOptions" RefId="13">
    <TN RefId="3">
      <T>System.Management.Automation.RemoteStreamOptions</T>
      <T>System.Enum</T>
      <T>System.ValueType</T>
      <T>System.Object</T>
    </TN>
    <ToString>AddInvocationInfo</ToString>
    <I32>15</I32>
  </Obj>
  <B N="AddToHistory">false</B>
  <Obj N="HostInfo" RefId="14">
    <MS>
      <B N="_isHostNull">true</B>
      <B N="_isHostUINull">true</B>
      <B N="_isHostRawUINull">true</B>
      <B N="_useRunspaceHost">true</B>
    </MS>
  </Obj>
  <B N="IsNested">false</B>
 </MS>
</Obj>
```

2.2.2.11 GET_AVAILABLE_RUNSPACES Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a GET_AVAILABLE_RUNSPACES message when the **MessageType** field has a value of 0x00021007.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- Call ID
  - Property name: ci.

- Property type: Signed long (see section 2.2.5.1.13).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="7">
  <MS>
    <I64 N="ci">3</I64>
  </MS>
</Obj>
```

2.2.2.12 USER_EVENT Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a USER_EVENT message when the **MessageType** field has a value of 0x00021008.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- Event identifier
  - Property name: PSEventArgs.EventIdentifier.
  - Property type: Signed int (see section 2.2.5.1.11).
- Source identifier
  - Property name: PSEventArgs.SourceIdentifier.
  - Property type: String (see section 2.2.5.1.1).
- Time when event was generated
  - Property name: PSEventArgs.TimeGenerated.
  - Property type: Date/Time (see section 2.2.5.1.4).
- Sender of the event
  - Property name: PSEventArgs.Sender.
  - Property type: Any Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).
- Event arguments
  - Property name: PSEventArgs.SourceArgs.
  - Property type: Any Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).
- Message data
  - Property name: PSEventArgs.MessageData.

- Property type: Any Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).

- Name of the computer where the event was fired.
  - Property name: PSEventArgs.ComputerName.
  - Property type: Null (see section 2.2.5.1.20) or String (see section 2.2.5.1.1).
- ID of the runspace.
  - Property name: PSEventArgs.RunspaceId.
  - Property type: GUID (see section 2.2.5.1.18).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
  <MS>
    <I32 N="PSEventArgs.EventIdentifier">1</I32>
    <S N="PSEventArgs.SourceIdentifier">ae6245f2-c179-4a9a-a039-47b60fc44500</S>
    <DT N="PSEventArgs.TimeGenerated">2009-06-17T10:57:23.1578277-07:00</DT>
    <Obj N="PSEventArgs.Sender" RefId="1">
      <TN RefId="0">
        <T>System.Timers.Timer</T>
        <T>System.ComponentModel.Component</T>
        <T>System.MarshalByRefObject</T>
        <T>System.Object</T>
      </TN>
      <ToString>System.Timers.Timer</ToString>
      <Props>
        <B N="AutoReset">true</B>
        <B N="Enabled">true</B>
        <Db N="Interval">5000</Db>
        <Nil N="Site" />
        <Nil N="SynchronizingObject" />
        <Nil N="Container" />
      </Props>
    </Obj>
    <Obj N="PSEventArgs.SourceArgs" RefId="2">
      <TN RefId="1">
        <T>System.Object[]</T>
        <T>System.Array</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <Ref RefId="1" />
        <Obj RefId="3">
          <TN RefId="2">
            <T>System.Timers.ElapsedEventArgs</T>
            <T>System.EventArgs</T>
            <T>System.Object</T>
          </TN>
          <ToString>System.Timers.ElapsedEventArgs</ToString>
          <Props>
            <DT N="SignalTime">2009-06-17T10:57:23.1568275-07:00</DT>
```

```
          </Props>
        </Obj>
      </LST>
    </Obj>
    <Nil N="PSEventArgs.MessageData" />
    <Nil N="PSEventArgs.ComputerName" />
    <G N="PSEventArgs.RunspaceId">fb9c87e8-1190-40a7-a681-6fc9b9f84a17</G>
  </MS>
</Obj>
```

2.2.2.13 APPLICATION_PRIVATE_DATA Message

The **Data** field of a PowerShell Remoting Protocol message specifies an APPLICATION_PRIVATE_DATA message when the **MessageType** field has a value of 0x00021009.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9). Note that the PowerShell Remoting Protocol does not generate or interpret any application private data; it merely provides a mechanism for the higher layer on the PowerShell server to send application private data to a PowerShell client, and a mechanism for the higher-layer on the PowerShell client to be notified when application private data is reported by the PowerShell server.

- Application private data that the higher layer provides to the PowerShell server when a RunspacePool is created on the server. The PowerShell Remoting Protocol does not interpret this data; it merely passes it to the higher-layers on the client.
  - Property name: ApplicationPrivateData
  - Property type: A Primitive Dictionary (see section 2.2.3.18) or Null Value (see section 2.2.5.1.20).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
  <MS>
    <Obj N="ApplicationPrivateData" RefId="1">
      <TN RefId="0">
        <T>System.Management.Automation.PSPrimitiveDictionary</T>
        <T>System.Collections.Hashtable</T>
        <T>System.Object</T>
      </TN>
      <DCT>
        <En>
          <S N="Key">BashPrivateData</S>
          <Obj N="Value" RefId="2">
            <TNRef RefId="0" />
            <DCT>
              <En>
                <S N="Key">BashVersion</S>
                <Version N="Value">2.0</Version>
              </En>
            </DCT>
```

```
          </Obj>
        </En>
      </DCT>
    </Obj>
  </MS>
</Obj>
```

2.2.2.14 GET_COMMAND_METADATA Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a GET_COMMAND_METADATA message when the **MessageType** field has a value of 0x0002100A.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- List of wildcard patterns specifying the command names that the server SHOULD return. If the value of this property is equal to Null (see section 2.2.5.1.20), then it MUST be treated as if a List with a single "*" String was specified.
  - Property name: Name
  - Property type: List (see section 2.2.5.2.6.3) of Wildcards (see section 2.2.3.20).
- Command types.
  - Property name: CommandType
  - Property type: CommandType (see section 2.2.3.19).
- Wildcard patterns describing the **command namespaces** containing the commands that the server SHOULD return. If the value of this property is Null, then it MUST be treated as if a List with a single empty String was specified.
  - Property name: Namespace
  - Property type: List (see section 2.2.5.2.6.3) of Wildcards (see section 2.2.3.20).
- Extra arguments passed to the higher-layer above the PowerShell Remoting Protocol and not interpreted by the PowerShell Remoting Protocol.
  - Property name: ArgumentList
  - Property type: List (see section 2.2.5.2.6.3) of objects. For more information, see section 2.2.3.24.

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
  <MS>
    <Obj N="Name" RefId="1">
      <TN RefId="0">
        <T>System.String[]</T>
        <T>System.Array</T>
```

```
        <T>System.Object</T>
      </TN>
      <LST>
        <S>Get-*</S>
      </LST>
    </Obj>
    <Obj N="CommandType" RefId="2">
      <TN RefId="1">
        <T>System.Management.Automation.CommandTypes</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Alias, Function, Filter, Cmdlet</ToString>
      <I32>15</I32>
    </Obj>
    <Obj N="Namespace" RefId="3">
      <TNRef RefId="0" />
      <LST />
    </Obj>
    <Nil N="ArgumentList" />
  </MS>
</Obj>
```

2.2.2.15 RUNSPACEPOOL_HOST_CALL Message

The **Data** field of a PowerShell Remoting Protocol message specifies a RUNSPACEPOOL_HOST_CALL message when the **MessageType** field has a value of 0x00021100.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- Call ID
  - Property name: ci.
  - Property type: Signed long (see section 2.2.5.1.13).
- Host method identifier
  - Property name: mi.
  - Property type: Host Method Identifier (see section 2.2.3.17).
- Parameters for the method
  - Property name: mp.
  - Property type: Host Parameters Encoded (see section 2.2.6).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
```

```
    <MS>
      <I64 N="ci">1</I64>
      <Obj N="mi" RefId="1">
        <TN RefId="0">
          <T>System.Management.Automation.Remoting.RemoteHostMethodId</T>
          <T>System.Enum</T>
          <T>System.ValueType</T>
          <T>System.Object</T>
        </TN>
        <ToString>ReadLine</ToString>
        <I32>11</I32>
      </Obj>
      <Obj N="mp" RefId="2">
        <TN RefId="1">
          <T>System.Collections.ArrayList</T>
          <T>System.Object</T>
        </TN>
        <LST />
      </Obj>
    </MS>
  </Obj>
```

2.2.2.16 RUNSPACEPOOL_HOST_RESPONSE Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a RUNSPACEPOOL_HOST_RESPONSE message when the **MessageType** field has a value of 0x00021101.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- Call ID
  - Property name: ci.
  - Property type: Signed long (see section 2.2.5.1.13).
- ID of the host method that the response is coming from
  - Property name: mi.
  - Property type: Host Method Identifier (see section 2.2.3.17).
- Return value of the method
  - Property name: mr.
  - Property type: Host Parameter Encoding in Host Method Calls (see section 2.2.6).
- Exception thrown by a host method invocation
  - Property name: me.
  - Property type: ErrorRecord (see section ErrorRecord). The FullyQualifiedErrorId property SHOULD have a value of "RemoteHostExecutionException".

Note that if either the mr property or the me property is present, the other may be omitted.

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="11">
  <MS>
    <S N="mr">Line read from the host</S>
    <I64 N="ci">1</I64>
    <Obj N="mi" RefId="12">
      <TN RefId="4">
        <T>System.Management.Automation.Remoting.RemoteHostMethodId</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>ReadLine</ToString>
      <I32>11</I32>
    </Obj>
  </MS>
</Obj>
```

2.2.2.17 PIPELINE_INPUT Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a PIPELINE_INPUT message when the **MessageType** field has a value of 0x00041002.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing the input object. The object can be of any type specified in section 2.2.5.

2.2.2.18 END_OF_PIPELINE_INPUT Message

The **Data** field of a PowerShell Remoting Protocol Message specifies an END_OF_PIPELINE_INPUT message when the **MessageType** field has a value of 0x00041003.

In messages of this type, the **Data** field is empty (has a length of zero bytes).

2.2.2.19 PIPELINE_OUTPUT Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a PIPELINE_OUTPUT message when the **MessageType** field has a value of 0x00041004.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing the output object. The object can be of any type specified in section 2.2.5.

2.2.2.20 ERROR_RECORD Message

The **Data** field of a PowerShell Remoting Protocol Message specifies an ERROR_RECORD message when the **MessageType** field has a value of 0x00041005.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing the ErrorRecord (see section 2.2.3.14).

Example:

```
<Obj RefId="0">
  <TN RefId="0">
    <T>System.Management.Automation.ErrorRecord</T>
    <T>System.Object</T>
  </TN>
  <ToString>Can't open file</ToString>
  <MS>
    <Obj N="Exception" RefId="1">
      <TN RefId="1">
        <T>System.IO.IOException</T>
        <T>System.SystemException</T>
        <T>System.Exception</T>
        <T>System.Object</T>
      </TN>
      <ToString>System.IO.IOException: Can't open file</ToString>
      <Props>
        <S N="Message">
          Can't open file</S><Obj N="Data" RefId="2">
            <TN RefId="2">
              <T>System.Collections.ListDictionaryInternal</T>
              <T>System.Object</T>
            </TN>
            <DCT />
          </Obj><Nil N="InnerException" /><Nil N="TargetSite" /><Nil N="StackTrace" /><Nil
N="HelpLink" /><Nil N="Source" />
      </Props>
    </Obj>
    <Nil N="TargetObject" />
    <S N="FullyQualifiedErrorId">System.IO.IOException</S>
    <Obj N="InvocationInfo" RefId="3">
      <TN RefId="3">
        <T>System.Management.Automation.InvocationInfo</T>
        <T>System.Object</T>
      </TN>
      <ToString>System.Management.Automation.InvocationInfo</ToString>
      <Props>
        <Obj N="MyCommand" RefId="4">
          <TN RefId="4">
            <T>System.Management.Automation.ScriptInfo</T>
            <T>System.Management.Automation.CommandInfo</T>
            <T>System.Object</T>
          </TN>
          <ToString>write-error -category OpenError -exception (new-object io.ioexception
"Can't open file") </ToString>
          <Props>
            <SBK N="ScriptBlock">write-error -category OpenError -exception (new-object
io.ioexception "Can't open file")</SBK>
            <S N="Definition">write-error -category OpenError -exception (new-object
io.ioexception "Can't open file")</S>
            <S N="Name"></S>
            <S N="CommandType">Script</S>
            <S N="Visibility">Public</S>
            <S N="ModuleName"></S>
            <Nil N="Module" />
            <Obj N="ParameterSets" RefId="5">
              <TN RefId="5">

<T>System.Collections.ObjectModel.ReadOnlyCollection`1[[System.Management.Automation.CommandP
```

```
arameterSetInfo, System.Management.Automation, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35]]</T>
                <T>System.Object</T>
              </TN>
              <LST>
                <S></S>
              </LST>
            </Obj>
          </Props>
        </Obj>
        <Obj N="BoundParameters" RefId="6">
          <TN RefId="6">
            <T>System.Collections.Generic.Dictionary`2[[System.String, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089],[System.Object, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
            <T>System.Object</T>
          </TN>
          <DCT />
        </Obj>
        <Obj N="UnboundArguments" RefId="7">
          <TN RefId="7">
            <T>System.Collections.Generic.List`1[[System. Object, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
            <T>System.Object</T>
          </TN>
          <LST/>
        </Obj>
        <I32 N="ScriptLineNumber">0</I32>
        <I32 N="OffsetInLine">0</I32>
        <S N="ScriptName"></S>
        <S N="Line"></S>
        <S N="PositionMessage"></S>
        <S N="InvocationName"></S>
        <I32 N="PipelineLength">1</I32>
        <I32 N="PipelinePosition">1</I32>
        <B N="ExpectingInput">false</B>
        <S N="CommandOrigin">Runspace</S>
      </Props>
    </Obj>
    <I32 N="ErrorCategory_Category">1</I32>
    <S N="ErrorCategory_Activity">Write-Error</S>
    <S N="ErrorCategory_Reason">IOException</S>
    <S N="ErrorCategory_TargetName"></S>
    <S N="ErrorCategory_TargetType"></S>
    <S N="ErrorCategory_Message">OpenError: (:) [Write-Error], IOException</S>
    <B N="SerializeExtendedInfo">true</B>
    <Ref N="InvocationInfo_BoundParameters" RefId="6" />
    <Obj N="InvocationInfo_CommandOrigin" RefId="8">
      <TN RefId="8">
        <T>System.Management.Automation.CommandOrigin</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Runspace</ToString>
      <I32>0</I32>
    </Obj>
    <B N="InvocationInfo_ExpectingInput">false</B>
    <S N="InvocationInfo_InvocationName"></S>
    <S N="InvocationInfo_Line"></S>
```

```
      <I32 N="InvocationInfo_OffsetInLine">0</I32>
      <Obj N="InvocationInfo_PipelineIterationInfo" RefId="9">
        <TN RefId="9">
          <T>System.Int32[]</T>
          <T>System.Array</T>
          <T>System.Object</T>
        </TN>
        <LST>
          <I32>0</I32>
          <I32>0</I32>
        </LST>
      </Obj>
      <I32 N="InvocationInfo_PipelineLength">1</I32>
      <I32 N="InvocationInfo_PipelinePosition">1</I32>
      <S N="InvocationInfo_PositionMessage"></S>
      <I32 N="InvocationInfo_ScriptLineNumber">0</I32>
      <S N="InvocationInfo_ScriptName"></S>
      <Ref N="InvocationInfo_UnboundArguments" RefId="7" />
      <Obj N="CommandInfo_CommandType" RefId="10">
        <TN RefId="10">
          <T>System.Management.Automation.CommandTypes</T>
          <T>System.Enum</T>
          <T>System.ValueType</T>
          <T>System.Object</T>
        </TN>
        <ToString>Script</ToString>
        <I32>64</I32>
      </Obj>
      <S N="CommandInfo_Definition">write-error -category OpenError -exception (new-object
  io.ioexception "Can't open file") </S>
      <S N="CommandInfo_Name"></S>
      <Obj N="CommandInfo_Visibility" RefId="11">
        <TN RefId="11">
          <T>System.Management.Automation.SessionStateEntryVisibility</T>
          <T>System.Enum</T>
          <T>System.ValueType</T>
          <T>System.Object</T>
        </TN>
        <ToString>Public</ToString>
        <I32>0</I32>
      </Obj>
      <Obj N="PipelineIterationInfo" RefId="12">
        <TN RefId="12">
          <T>System.Collections.ObjectModel.ReadOnlyCollection`1[[System.Int32, mscorlib,
  Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
          <T>System.Object</T>
        </TN>
        <LST>
          <I32>0</I32>
          <I32>0</I32>
        </LST>
      </Obj>
      <Nil N="PSMessageDetails" />
    </MS>
  </Obj>
```

2.2.2.21 PIPELINE_STATE Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a PIPELINE_STATE message when the **MessageType** field has a value of 0x00041006.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9).

- State information of PowerShell
  - Property name: PipelineState.
  - Property type: PSInvocationState (see section 2.2.3.5).
- Optional error information (included only if this message is triggered by an error).
  - Property name: ExceptionAsErrorRecord.
  - Property type: ErrorRecord (see section ErrorRecord). The FullyQualifiedErrorId property SHOULD have a value of "RemotePSInvocationStateInfoReason".

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj RefId="0">
  <MS>
    <I32 N="PipelineState">3</I32>
    <Obj N="ExceptionAsErrorRecord" RefId="1">
      <TN RefId="0">
        <T>System.Management.Automation.ErrorRecord</T>
        <T>System.Object</T>
      </TN>
      <ToString>The pipeline has been stopped.</ToString>
      <MS>
        <Obj N="Exception" RefId="2">
          <TN RefId="1">
            <T>System.Management.Automation.PipelineStoppedException</T>
            <T>System.Management.Automation.RuntimeException</T>
            <T>System.SystemException</T>
            <T>System.Exception</T>
            <T>System.Object</T>
          </TN>
          <ToString>System.Management.Automation.PipelineStoppedException: The pipeline has
been stopped._x000D__x000A_   at
System.Management.Automation.Internal.PipelineProcessor.SynchronousExecuteEnumerate(Object
input, Hashtable errorResults, Boolean enumerate) in
c:\e\win7_powershell\admin\monad\src\engine\pipeline.cs:line 586</ToString>
          <Props>
            <S N="ErrorRecord">The pipeline has been stopped.</S>
            <S N="StackTrace">   at
System.Management.Automation.Internal.PipelineProcessor.SynchronousExecuteEnumerate(Object
input, Hashtable errorResults, Boolean enumerate) in
c:\e\win7_powershell\admin\monad\src\engine\pipeline.cs:line 586</S>
            <S N="Message">The pipeline has been stopped.</S>
            <Obj N="Data" RefId="3">
              <TN RefId="2">
```

```
                <T>System.Collections.ListDictionaryInternal</T>
                <T>System.Object</T>
              </TN>
              <DCT />
            </Obj>
            <Nil N="InnerException" />
            <S N="TargetSite">System.Array SynchronousExecuteEnumerate(System.Object,
System.Collections.Hashtable, Boolean)</S>
            <Nil N="HelpLink" />
            <S N="Source">System.Management.Automation</S>
          </Props>
        </Obj>
        <Nil N="TargetObject" />
        <S N="FullyQualifiedErrorId">PipelineStopped</S>
        <Nil N="InvocationInfo" />
        <I32 N="ErrorCategory_Category">14</I32>
        <S N="ErrorCategory_Activity"></S>
        <S N="ErrorCategory_Reason">PipelineStoppedException</S>
        <S N="ErrorCategory_TargetName"></S>
        <S N="ErrorCategory_TargetType"></S>
        <S N="ErrorCategory_Message">OperationStopped: (:) [], PipelineStoppedException</S>
        <B N="SerializeExtendedInfo">false</B>
      </MS>
    </Obj>
  </MS>
</Obj>
```

2.2.2.22 DEBUG_RECORD Message

The **Data** field of a PowerShell Remoting Protocol message specifies a DEBUG_RECORD message when the **MessageType** field has a value of 0x00041007.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing the InformationalRecord (section 2.2.3.16), which SHOULD have the following type names:

- System.Management.Automation.DebugRecord
- System.Management.Automation.InformationalRecord
- System.Object

Example:

```
<Obj RefId="0">
  <TN RefId="0">
    <T>System.Management.Automation.DebugRecord</T>
    <T>System.Management.Automation.InformationalRecord</T>
    <T>System.Object</T>
  </TN>
  <ToString>Debug message</ToString>
  <MS>
    <S N="InformationalRecord_Message">Debug message</S>
    <B N="InformationalRecord_SerializeInvocationInfo">true</B>
    <Obj N="InvocationInfo_BoundParameters" RefId="1">
      <TN RefId="1">
```

```
        <T>System.Collections.Generic.Dictionary`2[[System.String, mscorlib, Version=2.0.0.0,
  Culture=neutral, PublicKeyToken=b77a5c561934e089],[System.Object, mscorlib, Version=2.0.0.0,
  Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <DCT>
        <En>
          <S N="Key">Debug</S>
          <Obj N="Value" RefId="2">
            <TN RefId="2">
              <T>System.Management.Automation.SwitchParameter</T>
              <T>System.ValueType</T>
              <T>System.Object</T>
            </TN>
            <ToString>True</ToString>
            <Props>
              <B N="IsPresent">true</B>
            </Props>
          </Obj>
        </En>
        <En>
          <S N="Key">Message</S>
          <S N="Value">Debug message</S>
        </En>
      </DCT>
    </Obj>
    <Obj N="InvocationInfo_CommandOrigin" RefId="3">
      <TN RefId="3">
        <T>System.Management.Automation.CommandOrigin</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Runspace</ToString>
      <I32>0</I32>
    </Obj>
    <B N="InvocationInfo_ExpectingInput">false</B>
    <S N="InvocationInfo_InvocationName">write-debug</S>
    <S N="InvocationInfo_Line"></S>
    <I32 N="InvocationInfo_OffsetInLine">0</I32>
    <Obj N="InvocationInfo_PipelineIterationInfo" RefId="4">
      <TN RefId="4">
        <T>System.Int32[]</T>
        <T>System.Array</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <I32>0</I32>
        <I32>1</I32>
      </LST>
    </Obj>
    <I32 N="InvocationInfo_PipelineLength">1</I32>
    <I32 N="InvocationInfo_PipelinePosition">1</I32>
    <S N="InvocationInfo_PositionMessage"></S>
    <I32 N="InvocationInfo_ScriptLineNumber">0</I32>
    <S N="InvocationInfo_ScriptName"></S>
    <Obj N="InvocationInfo_UnboundArguments" RefId="5">
      <TN RefId="5">
```

```
        <T>System.Collections.Generic.List`1[[System.Object, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <LST />
    </Obj>
    <Obj N="CommandInfo_CommandType" RefId="6">
      <TN RefId="6">
        <T>System.Management.Automation.CommandTypes</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Cmdlet</ToString>
      <I32>8</I32>
    </Obj>
    <S N="CommandInfo_Definition">Write-Debug [-Message] &lt;String&gt; [-Verbose] [-Debug]
[-ErrorAction &lt;ActionPreference&gt;] [-WarningAction &lt;ActionPreference&gt;] [-
ErrorVariable &lt;String&gt;] [-WarningVariable &lt;String&gt;] [-OutVariable &lt;String&gt;]
[-OutBuffer &lt;Int32&gt;]_x000D__x000A_</S>
    <S N="CommandInfo_Name">Write-Debug</S>
    <Obj N="CommandInfo_Visibility" RefId="7">
      <TN RefId="7">
        <T>System.Management.Automation.SessionStateEntryVisibility</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Public</ToString>
      <I32>0</I32>
    </Obj>
    <Obj N="InformationalRecord_PipelineIterationInfo" RefId="8">
      <TN RefId="8">
        <T>System.Collections.ObjectModel.ReadOnlyCollection`1[[System.Int32, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <I32>0</I32>
        <I32>1</I32>
      </LST>
    </Obj>
  </MS>
</Obj>
```

2.2.2.23 VERBOSE_RECORD Message

The **Data** field of a PowerShell Remoting Protocol Message contains the data of a VERBOSE_RECORD message when the **MessageType** field has a value of 0x00041008.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing the InformationalRecord (section 2.2.3.16), which SHOULD have the following type names:

- System.Management.Automation.VerboseRecord
- System.Management.Automation.InformationalRecord

- System.Object

Example:

```
<Obj RefId="0">
  <TN RefId="0">
    <T>System.Management.Automation.VerboseRecord</T>
    <T>System.Management.Automation.InformationalRecord</T>
    <T>System.Object</T>
  </TN>
  <ToString>Verbose message</ToString>
  <MS>
    <S N="InformationalRecord_Message">Verbose message</S>
    <B N="InformationalRecord_SerializeInvocationInfo">true</B>
    <Obj N="InvocationInfo_BoundParameters" RefId="1">
      <TN RefId="1">
        <T>System.Collections.Generic.Dictionary`2[[System.String, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089],[System.Object, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <DCT>
        <En>
          <S N="Key">Verbose</S>
          <Obj N="Value" RefId="2">
            <TN RefId="2">
              <T>System.Management.Automation.SwitchParameter</T>
              <T>System.ValueType</T>
              <T>System.Object</T>
            </TN>
            <ToString>True</ToString>
            <Props>
              <B N="IsPresent">true</B>
            </Props>
          </Obj>
        </En>
        <En>
          <S N="Key">Message</S>
          <S N="Value">Verbose message</S>
        </En>
      </DCT>
    </Obj>
    <Obj N="InvocationInfo_CommandOrigin" RefId="3">
      <TN RefId="3">
        <T>System.Management.Automation.CommandOrigin</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Runspace</ToString>
      <I32>0</I32>
    </Obj>
    <B N="InvocationInfo_ExpectingInput">false</B>
    <S N="InvocationInfo_InvocationName">write-verbose</S>
    <S N="InvocationInfo_Line"></S>
    <I32 N="InvocationInfo_OffsetInLine">0</I32>
    <Obj N="InvocationInfo_PipelineIterationInfo" RefId="4">
      <TN RefId="4">
        <T>System.Int32[]</T>
```

```
        <T>System.Array</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <I32>0</I32>
        <I32>1</I32>
      </LST>
    </Obj>
    <I32 N="InvocationInfo_PipelineLength">1</I32>
    <I32 N="InvocationInfo_PipelinePosition">1</I32>
    <S N="InvocationInfo_PositionMessage"></S>
    <I32 N="InvocationInfo_ScriptLineNumber">0</I32>
    <S N="InvocationInfo_ScriptName"></S>
    <Obj N="InvocationInfo_UnboundArguments" RefId="5">
      <TN RefId="5">
        <T>System.Collections.Generic.List`1[[System.Object, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <LST />
    </Obj>
    <Obj N="CommandInfo_CommandType" RefId="6">
      <TN RefId="6">
        <T>System.Management.Automation.CommandTypes</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System. Object</T>
      </TN>
      <ToString>Cmdlet</ToString>
      <I32>8</I32>
    </Obj>
    <S N="CommandInfo_Definition">Write-Verbose [-Message] &lt;String&gt; [-Verbose] [-Debug]
[-ErrorAction &lt;ActionPreference&gt;] [-WarningAction &lt;ActionPreference&gt;] [-
ErrorVariable &lt;String&gt;] [-WarningVariable &lt;String&gt;] [-OutVariable &lt;String&gt;]
[-OutBuffer &lt;Int32&gt;]_x000D__x000A_</S>
    <S N="CommandInfo_Name">Write-Verbose</S>
    <Obj N="CommandInfo_Visibility" RefId="7">
      <TN RefId="7">
        <T>System.Management.Automation.SessionStateEntryVisibility</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Public</ToString>
      <I32>0</I32>
    </Obj>
    <Obj N="InformationalRecord_PipelineIterationInfo" RefId="8">
      <TN RefId="8">
        <T>System.Collections.ObjectModel.ReadOnlyCollection`1[[System.Int32, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <I32>0</I32>
        <I32>1</I32>
      </LST>
    </Obj>
  </MS>
```

```
</Obj>
```

2.2.2.24 WARNING_RECORD Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a WARNING_RECORD message when the **MessageType** field has a value of 0x00041009.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing the InformationalRecord (section 2.2.3.16), which SHOULD have the following type names:

- System.Management.Automation.WarningRecord
- System.Management.Automation.InformationalRecord
- System.Object

Example:

```
<Obj RefId="0">
  <TN RefId="0">
    <T>System.Management.Automation.WarningRecord</T>
    <T>System.Management.Automation.InformationalRecord</T>
    <T>System.Object</T>
  </TN>
  <ToString>Warning message</ToString>
  <MS>
    <S N="InformationalRecord_Message">Warning message</S>
    <B N="InformationalRecord_SerializeInvocationInfo">true</B>
    <Obj N="InvocationInfo_BoundParameters" RefId="1">
      <TN RefId="1">
        <T>System.Collections.Generic.Dictionary`2[[System.String, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089],[System.Object, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <DCT>
        <En>
          <S N="Key">Message</S>
          <S N="Value">Warning message</S>
        </En>
      </DCT>
    </Obj>
    <Obj N="InvocationInfo_CommandOrigin" RefId="2">
      <TN RefId="2">
        <T>System.Management.Automation.CommandOrigin</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Runspace</ToString>
      <I32>0</I32>
    </Obj>
    <B N="InvocationInfo_ExpectingInput">false</B>
    <S N="InvocationInfo_InvocationName">write-warning</S>
    <S N="InvocationInfo_Line"></S>
    <I32 N="InvocationInfo_OffsetInLine">0</I32>
```

```
    <Obj N="InvocationInfo_PipelineIterationInfo" RefId="3">
      <TN RefId="3">
        <T>System.Int32[]</T>
        <T>System.Array</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <I32>0</I32>
        <I32>1</I32>
      </LST>
    </Obj>
    <I32 N="InvocationInfo_PipelineLength">1</I32>
    <I32 N="InvocationInfo_PipelinePosition">1</I32>
    <S N="InvocationInfo_PositionMessage"></S>
    <I32 N="InvocationInfo_ScriptLineNumber">0</I32>
    <S N="InvocationInfo_ScriptName"></S>
    <Obj N="InvocationInfo_UnboundArguments" RefId="4">
      <TN RefId="4">
        <T>System.Collections.Generic.List`1[[System.Object, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <LST />
    </Obj>
    <Obj N="CommandInfo_CommandType" RefId="5">
      <TN RefId="5">
        <T>System.Management.Automation.CommandTypes</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Cmdlet</ToString>
      <I32>8</I32>
    </Obj>
    <S N="CommandInfo_Definition">Write-Warning [-Message] &lt;String&gt; [-Verbose] [-Debug]
[-ErrorAction &lt;ActionPreference&gt;] [-WarningAction &lt;ActionPreference&gt;] [-
ErrorVariable &lt;String&gt;] [-WarningVariable &lt;String&gt;] [-OutVariable &lt;String&gt;]
[-OutBuffer &lt;Int32&gt;]_x000D__x000A_</S>
    <S N="CommandInfo_Name">Write-Warning</S>
    <Obj N="CommandInfo_Visibility" RefId="6">
      <TN RefId="6">
        <T>System.Management.Automation.SessionStateEntryVisibility</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Public</ToString>
      <I32>0</I32>
    </Obj>
    <Obj N="InformationalRecord_PipelineIterationInfo" RefId="7">
      <TN RefId="7">
        <T>System.Collections.ObjectModel.ReadOnlyCollection`1[[System.Int32, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <I32>0</I32>
        <I32>1</I32>
      </LST>
```

```
    </Obj>
  </MS>
</Obj>
```

2.2.2.25 PROGRESS_RECORD Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a PROGRESS_RECORD message when the **MessageType** field has a value of 0x00041010.

In messages of this type, the **Data** field is UTF-8 encoded XML, equivalent to the XML created by serializing the progress record (see section 2.2.5.1.25).

Example:

```
<PR>
  <AV>activity description</AV>
  <AI>1</AI>
  <Nil />
  <PI>-1</PI>
  <PC>-1</PC>
  <T>Processing</T>
  <SR>-1</SR>
  <SD>status description</SD>
</PR>
```

2.2.2.26 PIPELINE_HOST_CALL Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a PIPELINE_HOST_CALL message when the **MessageType** field has a value of 0x00041100.

In messages of this type, the **Data** field is formatted identically to the RUNSPACEPOOL_HOST_CALL message (specified in 2.2.2.15).

2.2.2.27 PIPELINE_HOST_RESPONSE Message

The **Data** field of a PowerShell remoting protocol message specifies a PIPELINE_HOST_RESPONSE message when the **MessageType** field has a value of 0x00041101.

In messages of this type, the **Data** field is formatted identically to the RUNSPACEPOOL_HOST_RESPONSE message (specified in section 2.2.2.16).

2.2.2.28 CONNECT_RUNSPACEPOOL Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a CONNECT_RUNSPACEPOOL message when the **MessageType** field has a value of 0x00010008. This message is not supported for protocol versions 2.0 and 2.1.

In messages of this type, the **Data** field contains UTF-8 encoded XML created by serializing a Complex Object (see section 2.2.5.2) with the following optional extended properties (see section 2.2.5.2.9):

- Minimum number of runspaces in the RunspacePool
  - Property name: MinRunspaces

- Property type: Signed int (see section 2.2.5.1.11)

- Maximum number of runspaces in the RunspacePool
  - Property name: MaxRunspaces
  - Property type: Signed int (see section 2.2.5.1.11)

Example:

```
<Obj RefId="1">
  <MS>
    <I32 N="MinRunspaces">1</I32>
    <I32 N="MaxRunspaces">1</I32>
  </MS>
</Obj>
```

2.2.2.29 RUNSPACE_INIT_DATA Message

The **Data** field of a PowerShell Remoting Protocol Message specifies a RUNSPACEPOOL_INIT_DATA message when the **MessageType** field has a value of 0x0002100B. This message is not supported for protocol versions 2.0 and 2.1.

In messages of this type, the **Data** field contains UTF-8 encoded XML that is equivalent to the XML created by serializing a Complex Object (see section 2.2.5.2) with the following optional extended properties (see section 2.2.5.2.9):

- Minimum number of runspaces in the RunspacePool
  - Property name: MinRunspaces
  - Property type: Signed int (see section 2.2.5.1.11)
- Maximum number of runspaces in the RunspacePool
  - Property name: MaxRunspaces
  - Property type: Signed int (see section 2.2.5.1.11)

Example:

```
<Obj RefId="1">
  <MS>
    <I32 N="MinRunspaces">1</I32>
    <I32 N="MaxRunspaces">1</I32>
  </MS>
</Obj>
```

2.2.3 Other Object Types

The following sections specify other object types used by the PowerShell Remoting Protocol.

2.2.3.1 Coordinates

This data type represents a position in the screen buffer of a user interface.

This data type is a Complex Object (section 2.2.5.2) with the following extended properties (section 2.2.5.2.9):

- Hardcoded type of the object
  - Property name: T.
  - type: String (see section 2.2.5.1.1).
  - Property value: System.Management.Automation.Host.Coordinates
- Coordinates value
  - Property name: V.
  - Property type: Complex Object (section 2.2.5.2) with the following extended properties (section 2.2.5.2.9):
    - X coordinate (0 is the leftmost column).
      - Property name: x.
      - Property type: Signed int (see section 2.2.5.1.11).
    - Y coordinate (0 is the topmost row).
      - Property name: y.
      - Property type: Signed int (see section 2.2.5.1.11).

The Complex Objects described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj N="Value" RefId="17">
  <MS>
    <S N="T">System.Management.Automation.Host.Coordinates</S>
    <Obj N="V" RefId="18">
      <MS>
        <I32 N="x">0</I32>
        <I32 N="y">0</I32>
      </MS>
    </Obj>
  </MS>
</Obj>
```

2.2.3.2 Size

This data type represents a size of a screen buffer area of a user interface.

This data type is a Complex Object (see section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- Hardcoded type of the object
  - Property name: T.

- Property type: String (see section 2.2.5.1.1).
- Property value: System.Management.Automation.Host.Size

- Size value
  - Property name: V.
  - Property type: Complex Object (section 2.2.5.2) with the following extended properties (section 2.2.5.2.9):
    - Width of an area.
      - Property name: width.
      - Property type: Signed int (see section 2.2.5.1.11).
    - Height of an area.
      - Property name: height.
      - Property type: Signed int (see section 2.2.5.1.11).

The Complex Objects described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj N="Value" RefId="8">
  <MS>
    <S N="T">System.Management.Automation.Host.Size</S>
    <Obj N="V" RefId="9">
      <MS>
        <I32 N="width">181</I32>
        <I32 N="height">98</I32>
      </MS>
    </Obj>
  </MS>
</Obj>
```

2.2.3.3 Color

This data type represents a color used in a user interface.

This data type is a Complex Object (section 2.2.5.2) with the following extended properties (section 2.2.5.2.9):

- Hard-coded type of the object
  - Property name: T.
  - type: String (see section 2.2.5.1.1).
  - Property value: System.ConsoleColor
- Color value

- Property name: V.
- Property type: signed int (section 2.2.5.1.11)
- Property value: Taken from the following table:

Value	Meaning
1 DarkBlue	Dark blue color.
2 DarkGreen	Dark green color.
3 DarkCyan	Dark cyan color.
4 DarkRed	Dark red color.
5 DarkMagenta	Dark magenta color.
6 DarkYellow	Dark yellow color.
7 Gray	Gray color.
8 DarkGray	Dark gray color.
9 Blue	Blue color.
10 Green	Green color.
11 Cyan	Cyan color.
12 Red	Red color.
13 Magenta	Magenta color.
14 Yellow	Yellow color.
15 White	White color.

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example:

```
<Obj N="Value" RefId="21">
  <MS>
   <S N="T">System.ConsoleColor</S>
   <I32 N="V">5</I32>
  </MS>
</Obj>
```

2.2.3.4 RunspacePoolState

This data type represents the state of a RunspacePool.

This data type is a signed int (see section 2.2.5.1.11) with the following allowed values.

Value	Meaning
0	BeforeOpen
1	Opening
2	Opened
3	Closed
4	Closing
5	Broken
6	NegotiationSent
7	NegotiationSucceeded
8	Connecting
9	Disconnected

2.2.3.5 PSInvocationState

This data type represents a state of a pipeline invocation.

This data type is a signed int (see section 2.2.5.1.11) with the following allowed values.

Value	Meaning
0	Not started
1	Running
2	Stopping
3	Stopped
4	Completed
5	Failed

Value	Meaning
6	Disconnected

2.2.3.6 PSThreadOptions

This data type represents thread options for an application or a higher-layer protocol on the server. Note that the PowerShell remoting protocol does not interpret this data type; it merely passes the data type from the higher-layers on the PowerShell client to the higher-layers on the PowerShell server.

This data type is an enum (see section Contents of enums) based on the default underlying type (signed int; see section 2.2.5.1.11) that defines the following named constants.

Value	Meaning
0 Default	The default value.
1 UseNewThread	Use a new thread.
2 ReuseThread	Reuse an existing thread.
3 UseCurrentThread	Use the current thread.

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Management.Automation.Runspaces.PSThreadOptions
- System.Enum
- System.ValueType
- System.Object

For an example, see section 2.2.2.2.

2.2.3.7 ApartmentState

This data type represents the apartment state of an application or higher-layer protocol built on top of the PowerShell remoting protocol. Note that the PowerShell remoting protocol does not interpret this data type; it merely passes the data type from the higher-layers on the PowerShell client to the higher-layers on the PowerShell server.

This data type is an enum (see section 2.2.5.2.7) based on the default underlying type (signed int; see section 2.2.5.1.11) that defines the following named constants.

Value	Meaning
0	Single-threaded apartment (STA).

Value	Meaning
STA	
1 MTA	Multi-threaded apartment (MTA).
2 Unknown	Unknown.

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Threading.ApartmentState
- System.Enum
- System.ValueType
- System.Object

For an example, see section 2.2.2.2.

2.2.3.8 RemoteStreamOptions

This data type specifies a set of zero or more options of a remote stream.

This data type represents the set of options by **encoding** them as a set of bit flags within a Signed Int (section 2.2.5.1.11). A given remote stream option is included in the set by setting the corresponding bit, or excluded by clearing the bit. The possible remote stream options and their corresponding values are listed in the following table:

Value	Meaning
0x01 AddInvocationInfoToErrorRecord	Add invocation information to ErrorRecord objects.
0x02 AddInvocationInfoToWarningRecord	Add invocation information to WarningRecord objects.
0x04 AddInvocationInfoToDebugRecord	Add invocation information to DebugRecord objects.
0x08 AddInvocationInfoToVerboseRecord	Add invocation information to VerboseRecord objects.

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Management.Automation.RemoteStreamOptions
- System.Enum
- System.ValueType
- System.Object

For an example, see section 2.2.2.10.

2.2.3.9 ErrorCategory

This data type represents a category of an error.

This data type is a signed Int (section 2.2.5.1.11), which can have the following values:

Value	Meaning
0 NotSpecified	The error category is unspecified.
1 OpenError	The error occurred while trying to perform an open.
2 CloseError	The error occurred while trying to perform a close.
3 DeviceError	The error originated with the device.
4 DeadlockDetected	A deadlock was detected.
5 InvalidArgument	An argument was invalid.
6 InvalidData	The data was invalid.
7 InvalidOperation	An operation was invalid.
8 InvalidResult	A result was invalid.
9 InvalidType	A type was invalid.
10 MetadataError	There is an error with the metadata.
11 NotImplemented	The operation is not implemented.
12 NotInstalled	The specified resource was not installed.
13 ObjectNotFound	The object was not found.
14 OperationStopped	The operation was stopped.

Value	Meaning
15 OperationTimeout	The operation timed out.
16 SyntaxError	There was an error with the syntax.
17 ParserError	There was an error with the parser.
18 PermissionDenied	Permission was denied.
19 ResourceBusy	The resource is busy.
20 ResourceExists	The resource already exists.
21 ResourceUnavailable	The resource was unavailable.
22 ReadError	The error occurred while trying to perform a read.
25 SecurityError	The error relates to security.

For an example, see section 2.2.2.20.

2.2.3.10 TimeZone

This data type represents a time zone.

This data type is an array of bytes (see section 2.2.5.1.17) containing an instance of the .Net type System::CurrentSystemTimeZone class (as specified in section 2.2.3.10.1) and serialized as described in [MS-NRBF].

2.2.3.10.1 CurrentSystemTimeZone

The syntax below follows the .NET Remoting Description Notation, as specified in [MS-NRTP], section 2.2.5.

CurrentSystemTimeZone is a Class, the Library name of which is "mscorlib". It is used to contain the time zone information.

```
namespace System
{
  class CurrentSystemTimeZone
  {
    System.Collections.Hashtable  m_CachedDaylightChanges;
    String                        m_daylightName;
    String                        m_standardName;
    Int64                         m_ticksOffset;
```

```
    }
  }
```

m_CachedDaylightChanges: A Hashtable from int to DaylightTime using default comparer (see section 2.2.3.10.2) used to cache DaylightTime values (see section 2.2.3.10.3) for a given year. As this field is only used for caching data that can be recalculated from other fields, it MAY be ignored.

m_daylightName: A string value that specifies the daylight saving time zone name. If daylight saving time is not used in the time zone, an empty string ("") is returned.

m_standardName: A string value that specifies the standard time zone name.

m_ticksOffset: Standard offset in ticks to the Universal time if no daylight saving is in used. For example, the offset for PST (Pacific Standard Time) would be -8 * 60 * 60 * 1000 * 10000.

2.2.3.10.2 Hashtable From int to DaylightTime Using Default Comparer

The syntax below follows the .NET Remoting Description Notation, as specified in [MS-NRTP], section 2.2.5.

Hashtable is a Class, the Library name of which is "mscorlib". It is used to contain a collection of key-value pairs. Keys are Int32 values, and Values are DaylightTime values (see section 2.2.3.10.3).

```
namespace System.Collections
{
  class Hashtable
  {
    Single                                LoadFactor;
    Int32                                 Version;
    System.Collections.IComparer          Comparer;
    System.Collections.IHashCodeProvider  HashCodeProvider;
    Int32                                 HashSize;
    System.Object[]                       Keys;
    System.Object[]                       Values;
  }
}
```

LoadFactor: The maximum ratio of elements to buckets.

Version: The version number of the HashTable contents.

Comparer: Reserved. The value of this field MUST be NullObject (as specified in [MS-NRTP], section 3.1.1).

HashCodeProvider: Reserved. The value of this field MUST be NullObject (as specified in [MS-NRTP], section 3.1.1).

HashSize: The number of buckets in the hash table.

Keys: An array of keys. All keys MUST be of type Int32. A key represents a year associated with a DaylightTime value (see section 2.2.3.10.3).

Values: An array of values. All values MUST be of the type DaylightTime (see section 2.2.3.10.3). The length of the **Values** array MUST be the same as length of **Keys** array.

2.2.3.10.3 DaylightTime

The syntax below follows the .NET Remoting Description Notation, as specified in [MS-NRTP], section 2.2.5.

DaylightTime is a Class, the Library name of which is "mscorlib". It is used to contain the information about daylight saving time.

```
namespace System.Globalization
{
  class DaylightTime
  {
    DateTime           m_start;
    DateTime           m_end;
    TimeSpan           m_delta;
  }
}
```

m_start: The start date of a daylight saving period.

m_end: The end date of a daylight saving period.

m_delta: The delta to standard offset.

2.2.3.11 PowerShell Pipeline

This data type represents a pipeline to be executed.

A PowerShell pipeline is an object with the following extended properties (see section 2.2.5.2.9).

- Boolean, indicating to the higher layer if this is a nested pipeline. The PSRP layer MUST NOT interpret this data.
  - Property name: IsNested.
  - Property type: Boolean (see section 2.2.5.1.3).
- Commands in the pipeline.
  - Property name: Cmds.
  - Property type: List (see section List) of individual command objects (see section 2.2.3.12) in the order they appear in the pipeline.

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

For an example, see section 2.2.2.10.

2.2.3.12 Command

This data type represents a command in a pipeline.

Command is an object with the following extended properties (see section 2.2.5.2.9).

- The name of command or text of script to execute. (The format of a script is unspecified, as the PowerShell Remoting Protocol directly passes the script to the remote runspace implemented in the higher layer on the server, which in turn parses and executes the script.)
  - Property name: Cmd.
  - Property type: String (see section 2.2.5.1.1).
- A Boolean indicating to the higher layer whether the command to execute is a script.
  - Property name: IsScript.
  - Property type: Boolean (see section 2.2.5.1.3).
- A Boolean indicating to the higher layer whether to use local scope or global scope to invoke the commands.
  - Property name: UseLocalScope.
  - Property type: Boolean (see section 2.2.5.1.3) or Null value (see section 2.2.5.1.20).
- A flag indicating to the higher layer whether error and output streams MUST be merged on pipeline invocation. This property SHOULD have the same value as MergeToResults.
  - Property name: MergeMyResults.
  - Property type: PipelineResultTypes (see section 2.2.3.31).
- A flag indicating to the higher layer whether error and output streams MUST be merged on pipeline invocation. This property SHOULD have the same value as MergeMyResults.
  - Property name: MergeToResults.
  - Property type: PipelineResultTypes (see section 2.2.3.31).
- A flag indicating to the higher layer whether execution MUST merge error and output streams coming from previous commands in the pipeline.
  - Property name: MergePreviousResults.
  - Property type: PipelineResultTypes (see section 2.2.3.31).
- A flag indicating to the higher layer whether the error stream MUST be merged with the output stream on pipeline invocation.
  - Property name: MergeError.
  - Property Type: PipelineResultTypes (see section 2.2.3.31).
- A flag indicating to the higher layer whether the warning stream MUST be merged with the output stream on pipeline invocation.
  - Property name: MergeWarning.
  - Property Type: PipelineResultTypes (see section 2.2.3.31).
- A flag indicating to the higher layer whether the verbose stream MUST be merged with the output stream on pipeline invocation.

    - Property name: MergeVerbose.
    - Property Type: PipelineResultTypes (see section 2.2.3.31).
- A flag indicating to the higher layer whether the debug stream MUST be merged with the output stream on pipeline invocation.
    - Property name: MergeDebug.
    - Property Type: PipelineResultTypes (see section 2.2.3.31).
- Arguments of the command.
    - Property name: Args.
    - Property type: List (see section 2.2.5.2.6.3) of individual command parameter objects (see section 2.2.3.13) in the order they appear in the command invocation.

The Complex Object described in this section SHOULD have no associated type names (see section 2.2.5.2.3).

For an example, see section 2.2.2.10.

2.2.3.13 Command Parameter

This data type represents a parameter of a command implemented by a higher layer on the server.

A command parameter is an object with the following extended properties (see section 2.2.5.2.9).

- Name of the parameter
    - Property name: N.
    - Property type: String (see section 2.2.5.1.1) if the parameter has a name; otherwise a Null value (see section 2.2.5.1.20).
- Parameter value
    - Property name: V.
    - Property type: Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

For an example, see section 2.2.2.10.

2.2.3.14 HostInfo

This data type represents host information.

This data type is a Complex Object (see section 2.2.5.2 and 2.2.5.2.8) with the following extended properties (see section 2.2.5.3.4.2):

- A dictionary of elements with host-related information. See the following table for information about required keys.
    - Property name: _hostDefaultData

    - Property type: Dictionary (see section 2.2.6.1.6) with Keys that are Signed Ints (see section 2.2.5.1.11) and Values are of the type described in the following table.
- Flag specifying if the host object associated with the runspace/RunspacePool is null.
    - Property name: _isHostNull.
    - Property type: Boolean (see section 2.2.5.1.3).
- Flag specifying if the UI implementation of the host interface is null.
    - Property name: _ isHostUINull.
    - Property type: Boolean (see section 2.2.5.1.3).
- Flag specifying if the RawUI implementation of the host interface is null.
    - Property name: _ isHostRawUINull.
    - Property type: Boolean (see section 2.2.5.1.3).
- Flag specifying whether a PowerShell invocation MUST use the host associated with its associated RunspacePool.
    - Property name: _ useRunspaceHost.
    - Property type: Boolean (see section 2.2.5.1.3).

The following are the elements which MUST be included in the _hostDefaultData dictionary.

Data	Type of dictionary value	Key (for dictionary)
ForegroundColor	Color - see section 2.2.3.3	0
BackgroundColor	Color - see section 2.2.3.3	1
CursorPosition	Coordinates - see section 2.2.3.1	2
WindowPosition	Coordinates - see section 2.2.3.1	3
CursorSize	Int32 - see section 2.2.5.1.11	4
BufferSize	Size - see section 2.2.3.2	5
WindowSize	Size - see section 2.2.3.2	6
MaxWindowSize	Size - see section 2.2.3.2	7
MaxPhysicalWindowSize	Size - see section 2.2.3.2	8
WindowTitle	String - see section 2.2.5.1.1	9

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

For an example, see section 2.2.2.2.

2.2.3.15 ErrorRecord

This data type represents information about an error.

This data type is a Complex Object (section 2.2.5.2) with the following extended properties (section 2.2.5.2.9):

- An optional higher-layer object that describes the error. Implementations of PSRP MUST NOT interpret this object.
  - Property name: Exception.
  - Property type: Any Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).
- An optional higher-layer object that caused the error. Implementations of PSRP MUST NOT interpret this object.
  - Property name: TargetObject.
  - Property type: Any Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).
- An optional higher-layer object describing what invocation caused the error. Implementations of PSRP MUST NOT interpret this object.
  - Property name: InvocationInfo.
  - Property type: A Complex Object encoded as specified in section 2.2.3.15.1.
- A string which uniquely identifies this error condition.
  - Property name: FullyQualifiedErrorId
  - Property type: String (section 2.2.5.1.1)
- Error category.
  - Property name: ErrorCategory_Category
  - Property type: ErrorCategory (section 2.2.3.9)
- An optional string describing the activity that encountered the error.
  - Property name: ErrorCategory_Activity
  - Property type: Null Value (section 2.2.5.1.20) or String (section 2.2.5.1.1).
- An optional string describing the cause of the error.
  - Property name: ErrorCategory_Reason
  - Property type: Null Value (section 2.2.5.1.20) or String (section 2.2.5.1.1).
- An optional string describing the object upon which the ErrorCategory_Activity has operated.
  - Property name: ErrorCategory_TargetName
  - Property type: Null Value (section 2.2.5.1.20) or String (section 2.2.5.1.1).

- An optional string describing the type of the object upon which the ErrorCategory_Activity has operated.
  - Property name: ErrorCategory_TargetType
  - Property type: Null Value (section 2.2.5.1.20) or String (section 2.2.5.1.1).
- An optional string describing the error.
  - Property name: ErrorCategory_Message
  - Property type: Null Value (section 2.2.5.1.20) or String (section 2.2.5.1.1).
- An optional string describing the error. This property can be missing; when this property is missing, the condition MUST be treated in the same way as if the property had been set to the Null Value.
  - Property name: ErrorDetails_Message
  - Property type: Null Value (section 2.2.5.1.20) or String (section 2.2.5.1.1).
- An optional string describing the recommended action the user can take. This property can be missing; when this property is missing, the condition MUST be treated in the same way as if the property had been set to the Null Value.
  - Property name: ErrorDetails_RecommendedAction
  - Property type: Null Value (section 2.2.5.1.20) or String (section 2.2.5.1.1).
- Flag indicating if other (section 2.2.3.15.1) properties below have been included in the object or not.
  - Property name: SerializeExtendedInfo
  - Property type: Boolean (section 2.2.5.1.3). TRUE means that InvocationInfo-specific extended properties (section 2.2.3.15.1) are present in the ErrorRecord.
- The status, when this record was created, of the pipeline provided by the higher-layer. This SHOULD be the same as value as InvocationInfo_PipelineIterationInfo (section 2.2.3.15.1). This property is present if and only if SerializeExtendedInfo property is TRUE.
  - Property name: PipelineIterationInfo
  - Property type: List (section 2.2.5.2.6.3) of Signed Ints (section 2.2.5.1.11).

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Management.Automation.ErrorRecord
- System.Object

For an example, see section 2.2.2.20.

2.2.3.15.1 InvocationInfo-specific Extended Properties

Error records (section 2.2.3.15) and informational records (section 2.2.3.16) can optionally include extended properties that the higher layer provides in order to describe the higher-layer invocation

that caused the error. MS-PSRP implementations MUST NOT interpret this data. Note that these properties can describe a higher-layer command whose name was directly mentioned in a Command data type (section 2.2.3.12), but these properties can also describe an internal higher-layer command that was invoked by an implementation of another higher layer command.

The following is a complete list of InvocationInfo-specific extended properties:

- The **command name** used to invoke this command; if invoked through an alias, then this is the alias name.
  - Property name: InvocationInfo_InvocationName
  - Property type: String (section 2.2.5.1.1)
- The command line parameters.
  - Property name: InvocationInfo_BoundParameters
  - Property type: Dictionary (section 2.2.5.2.6.4) where keys (representing parameter names) are Strings (section 2.2.5.1.1) and values (representing parameter values) are any Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).
- The unbound command line parameters.
  - Property name: InvocationInfo_UnboundArguments
  - Property type: List (section 2.2.5.2.6.3), where elements (representing parameter values) are any Primitive Type Object (section 2.2.5.1) or Complex Object (section 2.2.5.2).
- The command origin.
  - Property name: InvocationInfo_CommandOrigin
  - Property type: CommandOrigin (section 2.2.3.30)
- Flag indicating whether or not the command was expecting pipeline input.
  - Property name: InvocationInfo_ExpectingInput
  - Property type: Boolean (section 2.2.5.1.3)
- The text of the line that contained this command invocation.
  - Property name: InvocationInfo_Line
  - Property type: String (section 2.2.5.1.1)
- The offset of the first character in InvocationInfo_Line that is associated with this command.
  - Property name: InvocationInfo_OffsetInLine
  - Property type: Signed Int (section 2.2.5.1.11)
- A human-readable message indicating where the command appeared in the command line.
  - Property name: InvocationInfo_PositionMessage
  - Property type: String (section 2.2.5.1.1)

- The name of the script (if executing a script) that invoked this command.
  - Property name: InvocationInfo_ScriptName
  - Property type: String (section 2.2.5.1.1)
- The line number (if executing a script) of the line that invoked this command.
  - Property name: InvocationInfo_ScriptLineNumber
  - Property type: Signed Int (section 2.2.5.1.11)
- A number provided by the higher layer. PSRP does not interpret this data.
  - Property name: InvocationInfo_HistoryId
  - Property type: Signed Long (section 2.2.5.1.13)
- The number of commands in the pipeline.
  - Property name: InvocationInfo_PipelineLength
  - Property type: Signed Int (section 2.2.5.1.11)
- The position of the current command in the pipeline.
  - Property name: InvocationInfo_PipelinePosition
  - Property type: Signed Int (section 2.2.5.1.11)
- The status of the pipeline when this record was created provided by the higher-layer. This SHOULD be set to the same value as PipelineIterationInfo.
  - Property name: InvocationInfo_PipelineIterationInfo
  - Property type: A List (section 2.2.5.2.6.3)) of Signed Int (section 2.2.5.1.11) structures.

2.2.3.16 InformationalRecord (DebugRecord, WarningRecord or VerboseRecord)

InformationalRecord (that is, DebugRecord, WarningRecord or VerboseRecord) is a structure that contains additional information that a pipeline can output in addition to the regular data output.

This data type is a Complex Object (section 2.2.5.2) with the following extended properties (see section 2.2.5.2.9):

- The message that a higher-layer pipeline or command wants to associate with the informational record.
  - Property name: InformationalRecord_Message
  - Property type: String (section 2.2.5.1.1)
- Flag indicating whether or not other properties (section 2.2.3.15.1) listed below have been included in the **object** or not.
  - Property name: InformationalRecord_SerializeInvocationInfo

- Property type: Boolean (section 2.2.5.1.3) value. When set to TRUE, indicates that InvocationInfo-specific extended properties (section 2.2.3.15.1) are present in the ErrorRecord.

- The status, when this record was created, of the pipeline provided by the higher-layer. This SHOULD be set to the same value as InvocationInfo_PipelineIterationInfo (section 2.2.3.15.1). This property is present if and only if the SerializeExtendedInfo property is set to TRUE.

  - Property name: InformationalRecord_PipelineIterationInfo

  - Property type: List (section 2.2.5.2.6.3) of Signed Int (section 2.2.5.1.11) structures.

The Complex Object described in this section SHOULD include the following type names (section 2.2.5.2.3):

- System.Management.Automation. InformationalRecord

- System.Object

For a complete list of type names and for examples, see sections 2.2.2.22, 2.2.2.23, and 2.2.2.24.

2.2.3.17 Host Method Identifier

This data type represents a method to be executed on a host.

This data type is an enum (as specified in section 2.2.5.2.7) based on the default underlying type (signed int, as specified in section 2.2.5.1.11) that defines the named constants listed in the following tables.

The following table lists the possible values for method identifiers when a PowerShell server invokes a host method on the client. What the host methods SHOULD or MUST do is also defined in the table.

The PowerShell client MUST hand over requests for execution of a host method to a higher-layer host. The host will either perform the action described in the Method Details column of the following table, or indicate that there was an error executing the host method (if the method is not supported or not implemented, for instance).

If the Return Value column indicates that the method returns a return value, then the PowerShell client MUST send a RUNSPACEPOOL_HOST_RESPONSE message (see section 2.2.2.16) or a PIPELINE_HOST_RESPONSE message (see section 2.2.2.27). If the higher-layer host reported an error after executing the host method, then the response message MUST include the "me" property. If the higher-layer host returned a return value after executing the host method, then the response message MUST include the "mr" property and the PowerShell client MUST make sure that the data type of the "mr" property is the same as the type of return value described in the following Return Value column.

If the Return Value column indicates that the method does not return a value, then the PowerShell client MUST NOT send a RUNSPACEPOOL_HOST_RESPONSE message (see section 2.2.2.16) or a PIPELINE_HOST_RESPONSE message (see section 2.2.2.27).

Host Read Only Properties

Name of method/property	Method identifier	Return value	Method details
GetName	1	String (section	SHOULD return a string identifying the

Name of method/property	Method identifier	Return value	Method details
		2.2.5.1.1)	hosting application in a user friendly way.
GetVersion	2	Version number (section 2.2.5.1.21)	SHOULD return the version number of the hosting application.
GetInstanceId	3	GUID (section 2.2.5.1.18)	SHOULD return a GUID that uniquely identifies the hosting application.
GetCurrentCulture	4	CultureInfo (section 2.2.6.1.2)	SHOULD return the host's culture.
GetCurrentUICulture	5	CultureInfo (section 2.2.6.1.2)	MUST return the host's UI culture.

Host Methods

Name of method/property	Method identifier	Return value	Method details
SetShouldExit	6	None.	SHOULD shut down the hosting application and close the current PowerShell runspace.
EnterNestedPrompt	7	None.	SHOULD interrupt the current pipeline and start a nested pipeline.
ExitNestedPrompt	8	None.	SHOULD stop the nested pipeline and resume the current pipeline.
NotifyBeginApplication	9	None.	Called by PowerShell to indicate that it is executing a command line application.
NotifyEndApplication	10	None.	Called by PowerShell to indicate that it has finished executing a command line application.

Host UI Methods

Name of method/property	Method identifier	Return value	Method details
ReadLine	11	String (section 2.2.5.1.1)	SHOULD read a line of characters from a user.
ReadLineAsSecureString	12	Secure String (section 2.2.5.1.24)	SHOULD read a line of characters from a user, with the user input not echoed.
Write1	13	None.	SHOULD write specified characters on the hosting application.
Write2	14	None.	SHOULD write the specified characters with the specified foreground and background color on the hosting

Name of method/property	Method identifier	Return value	Method details
			application.
WriteLine1	15	None.	SHOULD write a carriage return on the hosting application.
WriteLine2	16	None.	SHOULD write the specified line on the hosting application.
WriteLine3	17	None.	SHOULD write the specified line with the specified foreground and background color on the hosting application.
WriteErrorLine	18	None.	SHOULD write a line to the error display of the hosting application.
WriteDebugLine	19	None.	SHOULD write a line to the debug display of the hosting application.
WriteProgress	20	None.	SHOULD display a progress record on the hosting application.
WriteVerboseLine	21	None.	SHOULD write a line on the verbose display of the hosting application.
WriteWarningLine	22	None.	SHOULD write a line on the warning display of the hosting application.
Prompt	23	Dictionary (section 2.2.6.1.6) with String (section 2.2.5.1.1) keys representing the name of a field prompted for and values of arbitrary type.	SHOULD prompt the user with a set of choices.
PromptForCredential1	24	PSCredential (section 2.2.3.25)	SHOULD prompt the user for entering credentials with the specified caption, message, user name and target name.
PromptForCredential2	25	PSCredential (section 2.2.3.25)	SHOULD prompt the user for entering credentials with the specified caption, message, username, target name, allowed credential types and options.
PromptForChoice	26	Signed int (section 2.2.5.1.11	SHOULD display a list of choices to the user and MUST

Name of method/property	Method identifier	Return value	Method details
			return the index of the selected option.

Host RawUI Read/Write Properties

Name of method/property	Method identifier	Return value	Method details
GetForegroundColor	27	Color (section 2.2.3.3)	SHOULD return the foreground color of the hosting application.
SetForegroundColor	28	None.	SHOULD set the foreground color of the hosting application.
GetBackgroundColor	29	Color (section 2.2.3.3)	SHOULD return the background color of the hosting application.
SetBackgroundColor	30	None.	SHOULD set the background color of the hosting application.
GetCursorPosition	31	Coordinates (section 2.2.3.1)	SHOULD return the current cursor position in the hosting application.
SetCursorPosition	32	None.	SHOULD set the current cursor position in the hosting application.
GetWindowPosition	33	Coordinates (section 2.2.3.1)	SHOULD return the position of the view window relative to the screen buffer.
SetWindowPosition	34	None.	SHOULD set the position of the view window relative to the screen buffer.
GetCursorSize	35	Signed int (section 2.2.5.1.11)	SHOULD return the cursor size as a percentage.
SetCursorSize	36	None.	SHOULD set the cursor size based on the percentage value specified.
GetBufferSize	37	Size (section 2.2.3.2)	SHOULD return the current size of the screen buffer, measured in character cells.
SetBufferSize	38	None.	SHOULD set the size of the screen buffer with the specified size in character cells.
GetWindowSize	39	Size (section 2.2.3.2)	SHOULD return the current view window size.
SetWindowSize	40	None.	SHOULD set the view window size based on the size specified.
GetWindowTitle	41	String (section 2.2.5.1.1)	SHOULD return the title of the hosting application's window.
SetWindowTitle	42	None.	SHOULD set the title of the hosting

Name of method/property	Method identifier	Return value	Method details
			application's window.

Host RawUI Read Only Properties

Name of method/property	Method identifier	Return value	Method details
GetMaxWindowSize	43	Size (section 2.2.3.2)	SHOULD return the maximum window size possible for the current buffer, current font, and current display hardware.
GetMaxPhysicalWindowSize	44	Size (section 2.2.3.2)	SHOULD return the maximum window size possible for the current font and current display hardware, ignoring the current buffer size,.
GetKeyAvailable	45	Boolean (section 2.2.5.1.3)	SHOULD examine if a keystroke is waiting on the input, returning TRUE if so and FALSE otherwise.

Host RawUI Methods

Name of method/property	Method identifier	Return value	Method details
ReadKey	46	KeyInfo (section 2.2.3.26)	SHOULD read a key stroke from the keyboard, blocking until a key is typed.
FlushInputBuffer	47	None.	SHOULD reset the keyboard input buffer.
SetBufferContents1	48	None.	SHOULD copy the specified buffer cell array into the screen buffer at the specified coordinates (as specified in section 2.2.3.1).
SetBufferContents2	49	None.	SHOULD copy the specified buffer cell into all the cells within the specified rectangle.
GetBufferContents	50	Array (section 2.2.6.1.4) of BufferCell elements (section 2.2.3.28)	SHOULD return the contents in a specified rectangular region of the hosting application's window and MUST return an array of buffer cells.
ScrollBufferContents	51	None.	SHOULD scroll a region on the screen buffer.

IHostSupportsInteractiveSession Methods

Name of method/property	Method identifier	Return value	Method details
PushRunspace	52	None.	SHOULD store the current working runspace in a

Name of method/property	Method identifier	Return value	Method details
			stack and replace it with the new specified runspace.
PopRunspace	53	None.	SHOULD retrieve the last stored runspace from the stack and make it the current active runspace.

IHostSupportsInteractiveSession Read Only Properties

Name of method/property	Method identifier	Return value	Method details
GetIsRunspacePushed	54	Boolean (section 2.2.5.1.3)	SHOULD validate if there is a runspace currently pushed on a stack in the host, returning true if so and false otherwise.
GetRunspace	55	Any object (section 2.2.5) that the higher layer uses to represent the current runspace. The PowerShell Remoting Protocol MUST transparently pass the data received from the host implemented in the higher layer. The PowerShell Remoting Protocol MUST ignore this value.	SHOULD return the currently active runspace.

IHostSupportsMultipleChoiceSelect Methods

Name of method/property	Method identifier	Return value	Method details
PromptForChoiceMultipleSelection	56	Collection (section 2.2.6.1.5) of signed ints (section 2.2.5.1.11)	SHOULD display a list of choices to the user and return a list of options selected by the user.

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Management.Automation.Remoting.RemoteHostMethodId
- System.Enum
- System.ValueType
- System.Object

2.2.3.18 Primitive Dictionary

This data type represents a dictionary, which contains only objects that are primitive types.

This data type is a dictionary (see section 2.2.5.2.6.4) with the restriction that keys are strings (see section 2.2.5.1.1) and values are any of the following:

- Any Primitive Type Object (see section 2.2.5.1) except ScriptBlock (see section 2.2.5.1.23) or Secure String (section 2.2.5.1.24).
- A list (see section 2.2.5.2.6.3) of Primitive Type Objects (see section 2.2.5.1) except ScriptBlock (see section 2.2.5.1.23) or Secure String (section 2.2.5.1.24).
- Another Primitive Dictionary.

The dictionary described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Management.Automation.PSPrimitiveDictionary
- System.Collections.Hashtable
- System.Object

For an example see section 2.2.2.13.

2.2.3.19 CommandType

This data type specifies a set of zero or more command types. A command type optionally defines one of many possible command categories implemented at a higher layer.

The PowerShell Remoting Protocol does not interpret this data type, but instead passes it directly from higher layers on the client to higher layers on the server.

This data type represents the set of command types by encoding them as a 32-bit wide bit field within a Signed Int (section 2.2.5.1.11).

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Enum
- System.ValueType
- System.Object

For an example, see section 2.2.3.22.

2.2.3.20 Wildcard

This data type represents a wildcard pattern that can be matched against a String (see section 2.2.5.1.1).

This data type is a String (see section 2.2.5.1.1) with the contents interpreted according to section 2.13.2 Patterns Matching Multiple Characters in IEEE Std 1003.1, 2004 Edition with the following exceptions:

- The backtick character ("`") is used as an escape character, instead of a backslash character ("\").
- The exclamation character ("!") in a bracket expression does not have a special meaning.
- All character comparisons are case-insensitive.

2.2.3.21 CommandMetadataCount

This data type is an object with the following extended properties (see section 2.2.5.2.9):

- An integer value.
  - Property name: Count.
  - Property type: Signed Int (see section 2.2.5.1.11).

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- Selected.Microsoft.PowerShell.Commands.GenericMeasureInfo
- System.Management.Automation.PSCustomObject
- System.Object

Example:

```
<Obj RefId="0">
  <TN RefId="0">
    <T>Selected.Microsoft.PowerShell.Commands.GenericMeasureInfo</T>
    <T>System.Management.Automation.PSCustomObject</T>
    <T>System.Object</T>
  </TN>
  <MS>
    <I32 N="Count">1</I32>
  </MS>
</Obj>
```

2.2.3.22 CommandMetadata

This data type represents the metadata of a command. CommandMetadata is an object with the following extended properties (see section 2.2.5.2.9):

- The name of a command
  - Property name: Name.
  - Property type: a non-empty String (see section 2.2.5.1.1).
- The URI to the documentation of the command. If the higher layer provides a URI for documentation of the command, then the PowerShell Remoting Protocol MUST set HelpUri to the value provided by the higher layer; otherwise the value of HelpUri MUST be set to Null (section 2.2.5.1.20). The higher layer SHOULD provide the URI for documentation of all commands.
  - Property name: HelpUri.
  - Property type: String (see section 2.2.5.1.1).
- The CommandType of the command
  - Property name: CommandType.
  - Property type: CommandType (see section 2.2.3.19).

- Types of objects that a command can send as output (see section 2.2.2.19).
  - Property name: OutputType
  - Property type: List (see section 2.2.5.2.6.3) of Strings (see section 2.2.5.1.1) where each string specifies a type name (see section 2.2.5.2.3).
- Metadata of parameters that the command can accept as Command Parameters (section 2.2.3.13).
  - Property name: Parameters
  - Property type: Dictionary (see section 2.2.6.1.6). Type of dictionary keys: Strings (see section 2.2.5.1.1) that specify parameter name (see property "N" in section 2.2.3.13). Type of dictionary values: ParameterMetadata (see section 2.2.3.23).

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- Selected.System.Management.Automation.*command-type* where *command type* is replaced with one of the CommandType names (section 2.2.3.19)
- System.Management.Automation.PSCustomObject
- System.Object

Example:

```
<Obj RefId="0">
  <TN RefId="0">
    <T>Selected.System.Management.Automation.CmdletInfo</T>
    <T>System.Management.Automation.PSCustomObject</T>
    <T>System.Object</T>
  </TN>
  <MS>
    <S N="Name">Get-Variable</S>
    <S N="Namespace">Microsoft.PowerShell.Utility</S>
    <S N="HelpUri">http://go.microsoft.com/fwlink/?LinkID=113336</S>
    <Obj N="CommandType" RefId="1">
      <TN RefId="1">
        <T>System.Management.Automation.CommandTypes</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Cmdlet</ToString>
      <I32>8</I32>
    </Obj>
    <Nil N="ResolvedCommandName" />
    <Obj N="OutputType" RefId="2">
      <TN RefId="2">

<T>System.Collections.ObjectModel.ReadOnlyCollection`1[[System.Management.Automation.PSTypeNa
me, System.Management.Automation, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35]]</T>
        <T>System.Object</T>
      </TN>
      <LST>
        <S>System.Management.Automation.PSVariable</S>
```

```
      </LST>
    </Obj>
    <Obj N="Parameters" RefId="3">
      <TN RefId="3">
        <T>System.Collections.Generic.Dictionary`2[[System.String, mscorlib, Version=2.0.0.0,
Culture=neutral,
PublicKeyToken=b77a5c561934e089],[System.Management.Automation.ParameterMetadata,
System.Management.Automation, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35]]</T>
        <T>System.Object</T>
      </TN>
      <DCT>
        <En>
          <S N="Key">Name</S>
          <Obj N="Value" RefId="4">
            <TN RefId="4">
              <T>System.Management.Automation.ParameterMetadata</T>
              <T>System.Object</T>
            </TN>
            <ToString>System.Management.Automation.ParameterMetadata</ToString>
            <Props>
              <S N="Name">Name</S>
              <S N="ParameterType">System.String[]</S>
              <Obj N="Aliases" RefId="5">
                <TN RefId="5">
                  <T>System.Collections.ObjectModel.Collection`1[[System.String, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]]</T>
                  <T>System.Object</T>
                </TN>
                <LST />
              </Obj>
              <B N="IsDynamic">false</B>
              <B N="SwitchParameter">false</B>
            </Props>
          </Obj>
        </En>
      </DCT>
    </Obj>
  </MS>
</Obj>
```

2.2.3.23 ParameterMetadata

This data type specifies the metadata of a command parameter (see also section 2.2.3.13).

ParameterMetadata is an object with the following extended properties (see section 2.2.5.2.9):

- The name of a parameter.
  - Property name: Name.
  - Property type: a non-empty String (see section 2.2.5.1.1).
- The type of the parameter.
  - Property name: ParameterType.

    - Property type: String (see section 2.2.5.1.1) representing a type name (see section 2.2.5.2.3).
- Alternative names of the parameter
    - Property name: Aliases.
    - Property type: List (see section 2.2.5.2.6.3) of Strings (see section 2.2.5.1.1).
- The SwitchParameter property is True if ParameterType is equal to "System.Management.Automation.SwitchParameter" and False otherwise.
    - Property name: SwitchParameter.
    - Property type: Bool (see section 2.2.5.1.3).
- True if this parameter is included as a consequence of the data specified in the ArgumentList property (section 2.2.3.24).
    - Property name: IsDynamic
    - Property type: Bool (see section 2.2.5.1.3).

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Management.Automation.ParameterMetadata
- System.Object

2.2.3.24 ArgumentList

This data type specifies additional data that is passed to the higher layer on the server. The higher layer MAY use this data to control the parameter metadata (section 2.2.3.23) that gets returned. This data type MUST be a list (see section 2.2.5.2.6.3) of objects. Individual objects in the list can be of any type.

2.2.3.25 PSCredential

This data type represents a user name and a password.

This data type is a Complex Object (see section 2.2.5.2 and 2.2.5.2.8) with the following adapted properties (see section 2.2.5.3.4.1):

- User name
    - Property name: UserName
    - Property type: String (see section 2.2.5.1.1).
- Password.
    - Property name: Password
    - Property type: Secure String (see section 2.2.5.1.24).

The Complex Object described in this section MUST have the following type names (section 2.2.5.2.3):

- System.Management.Automation.PSCredential
- System.Object

Example (inside a PIPELINE_HOST_RESPONSE message):

```
<Obj RefId="0">
  <MS>
    <Obj N="mr" RefId="1">
      <TN RefId="0">
        <T>System.Collections.Hashtable</T>
        <T>System.Object</T>
      </TN>
      <DCT>
        <En>
          <S N="Key">Credential</S>
          <Obj N="Value" RefId="2">
            <TN RefId="1">
              <T>System.Management.Automation.PSCredential</T>
              <T>System.Object</T>
            </TN>
            <ToString>System.Management.Automation.PSCredential</ToString>
            <Props>
              <S N="UserName">\username</S>
              <SS N="Password">np7uo8n2ZhbN5Pp9LMpf03WLccPK1NQWYFQrg1UzyA8=</SS>
            </Props>
          </Obj>
        </En>
      </DCT>
    </Obj>
    <I64 N="ci">1</I64>
    <Obj N="mi" RefId="3">
      <TN RefId="2">
        <T>System.Management.Automation.Remoting.RemoteHostMethodId</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>Prompt</ToString>
      <I32>23</I32>
    </Obj>
  </MS>
</Obj>
```

2.2.3.26 KeyInfo

This data type represents information about a keyboard event.

This data type is a Complex Object (see section 2.2.5.2 and 2.2.5.2.8) with the following extended properties (see section 2.2.5.2.9):

- A virtual key code that identifies the given key in a device-independent manner.
  - Property name: virtualKeyCode
  - Property type: Signed Int (see section 2.2.5.1.11).

- Character corresponding to the pressed keys.
  - Property name: character
  - Property type: Character (see section 2.2.5.1.2).
- State of the control keys.
  - Property name: controlKeyState
  - Property type: ControlKeyStates (see section 2.2.3.27).
- True if the event was generated when a key was pressed; false otherwise.
  - Property name: keyDown
  - Property type: Boolean (see section 2.2.5.1.3).

The Complex Object described in this section SHOULD have no associated type names (section 2.2.5.2.3).

Example (inside a PIPELINE_HOST_RESPONSE message):

```
<Obj RefId="0">
  <MS>
    <Obj N="mr" RefId="1">
      <MS>
        <I32 N="virtualKeyCode">65</I32>
        <C N="character">97</C>
        <I32 N="controlKeyState">0</I32>
        <B N="keyDown">true</B>
      </MS>
    </Obj>
    <I64 N="ci">1</I64>
    <Obj N="mi" RefId="2">
      <TN RefId="0">
        <T>System.Management.Automation.Remoting.RemoteHostMethodId</T>
        <T>System.Enum</T>
        <T>System.ValueType</T>
        <T>System.Object</T>
      </TN>
      <ToString>ReadKey</ToString>
      <I32>46</I32>
    </Obj>
  </MS>
</Obj>
```

2.2.3.27 ControlKeyStates

This data type represents a set of zero or more control keys that are held down.

This data type represents the set of control keys by encoding them as a set of bit flags within a Signed Int (section 2.2.5.1.11). If a given control key is held down, then a corresponding bit is set; otherwise, the bit is cleared.

Value	Meaning
0x0001	RightAltPressed
0x0002	LeftAltPressed
0x0004	RightCtrlPressed
0x0008	LeftCtrlPressed
0x0010	ShiftPressed
0x0020	NumLockOn
0x0040	ScrollLockOn
0x0080	CapsLockOn
0x0100	EnhancedKey

For an example, see section 2.2.3.26.

2.2.3.28 BufferCell

This data type represents the contents of a cell of a Host's screen buffer.

This data type is a Complex Object (see section 2.2.5.2 and 2.2.5.2.8) with the following adapted properties (see section 2.2.5.3.4.1):

- Character visible in the cell
  - Property name: Character
  - Property type: Character (see section 2.2.5.1.2).
- Foreground color
  - Property name: ForegroundColor
  - Property type: Color (see section 2.2.3.3).
- Background color
  - Property name: BackgroundColor
  - Property type: Color (see section 2.2.3.3).
- Type of the buffer cell
  - Property name: BufferCellType
  - Property type: BufferCellType (see section 2.2.3.29).

2.2.3.29 BufferCellType

This data type represents the type of a cell of a screen buffer.

This data type is an enum (see section 2.2.5.2.7) based on the default underlying type (signed int; see section 2.2.5.1.11) that defines the following named values):

Named Value	Meaning
0 - Complete	The character occupies one BufferCell.
1 - Leading	The character occupies two BufferCells and this is the leading one.
2 - Trailing	The character occupies two BufferCells and this is the trailing one.

2.2.3.30 CommandOrigin

This data type describes what caused a higher layer command to run. PSRP MUST NOT interpret values of this type.

This data type is an enum (see section 2.2.5.2.7) based on the default underlying type, Signed Int (section 2.2.5.1.11), that defines the following named constants:

Value	Meaning
0 - Runspace	The command was invoked directly by the user.
1 – Internal	The command was invoked by another command.

2.2.3.31 PipelineResultTypes

The PipelineResultTypes data type specifies a set of zero or more pipeline result types.

The PowerShell Remoting Protocol does not interpret this data type, but instead passes it directly from the higher layers on the client to the higher layers on the server.

This data type represents the set of pipeline result types by encoding them as a set of bit flags within a Signed Int (section 2.2.5.1.11). A given pipeline result type is included in the set by setting the corresponding bit, or excluded by clearing the bit. The possible pipeline result types and their corresponding values are listed in the following table:

Value	Description
0x00 None	No Results
0x01 Output	Pipeline output
0x02 Error	Pipeline error output
0x04 Warning	Pipeline warning output.
0x08 Verbose	Pipeline verbose output.
0x10	Pipeline debug output.

Value	Description
Debug	
0x20 All	All pipeline output.
0x40 Null	No pipeline output.

The Complex Object described in this section SHOULD have the following type names (section 2.2.5.2.3):

- System.Management.Automation.Runspaces.PipelineResultTypes
- System.Enum
- System.ValueType
- System.Object

For an example, see section 2.2.2.10.

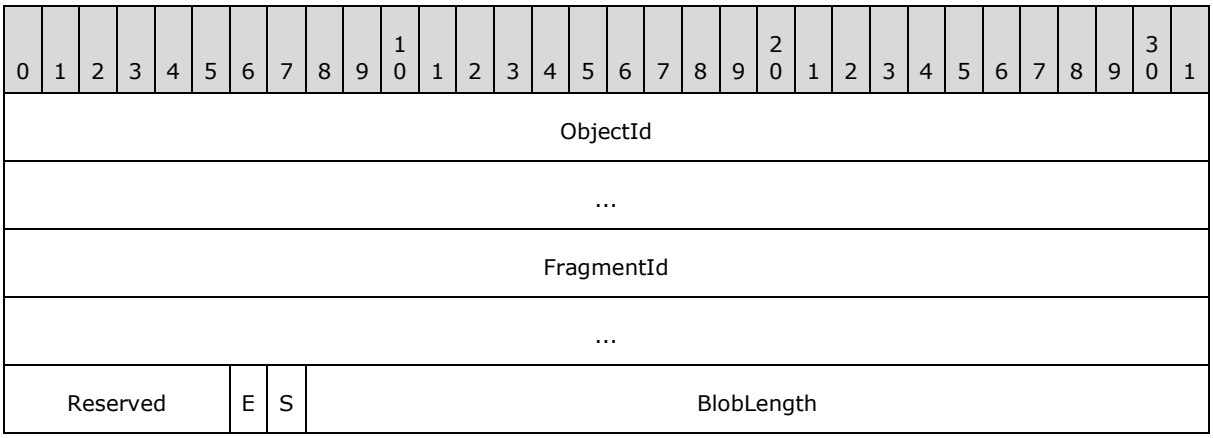
2.2.4 Packet Fragment

A **WS-MAN** packet can carry only a limited amount of data (as specified in [MS-WSMV] section 3.1.4.1.7). Some PowerShell Remoting Protocol Messages (as specified in section 2.2.1) may not fit into a single WS-MAN packet. To overcome this, the PowerShell Remoting Protocol fragments messages before sending.

An individual fragment MUST be sent in a single WS-MAN packet; in other words, an individual fragment cannot be broken down into smaller pieces and sent in separate WS-MAN packets.

A single WS-MAN packet, however, can contain multiple fragments. For instance, fragments belonging to a SESSION_CAPABILITY message and a INIT_RUNSPACEPOOL message could be sent together in the open content of a single wxf:Create WS-MAN packet.

Each message MUST be **fragmented** into one or more fragments with the fragment structure as described in the following section. Each fragment MUST fit into the payload of a WS-MAN message.



...	Blob (variable)
...	

ObjectId (8 bytes): An unsigned 8-byte integer specifying the ID of the PowerShell message (see section 2.2.1) to which the fragment belongs. As a PowerShell message may be sent as multiple packets, the receiver will use the ObjectId to map them to the same PowerShell message. The value of this field MUST be greater than 0 and unique within a given RunspacePool and its associated pipelines. The value is in the network-byte order.

FragmentId (8 bytes): An unsigned 8-byte integer that identifies where in the sequence of message fragments this fragment falls. The FragmentId values determine the order in which different fragments are combined to construct the PowerShell Remoting Protocol Message on the receiver's end. The value is in the network-byte order. The value of this field MUST start with 0.

Reserved (6 bits): Reserved for future use. MUST be set to 0 and ignored upon receipt.

E (1 bit): Specifies if the packet represents the End fragment. This will be used by the receiver to combine different packets for the same **deserialized** object. A value of 1 means the packet is End fragment.

If a deserialized object fits into 1 packet, then both the **E** field and the **S** field MUST be 1

Value	Meaning
0	Not an End fragment.
1	End fragment.

S (1 bit): Specifies if the packet represents the Start fragment. A value of 1 means the packet is Start fragment. If a deserialized object fits into 1 packet, then both the **E** field and the **S** field MUST be 1. The Start fragment MUST have a FragmentId of 0.

Value	Meaning
0	Not a Start fragment.
1	Start fragment.

BlobLength (4 bytes): The length, in bytes, of the **Blob** field. This field MUST be set to a value greater than or equal to 0 and less than or equal to 32768. The value is in network-byte order

Blob (variable): An entire PowerShell Remoting Protocol Message (as specified in section 2.2.1) or a part of a fragmented PowerShell Remoting Protocol Message

2.2.5 Serialization

An object MUST be converted to an XML document by the higher layer before passing it to the PowerShell Remoting Protocol. If the object type is listed in section 2.2.5.1, the higher layer MUST encode the object as specified in that section. For all other object types, the higher layer MUST encode the object as specified in section 2.2.5.2. The resulting XML document MAY have an XML declaration, as specified in [XML] section 2.8. All XML elements and attributes described in this section belong to the following XML namespace:

```
http://schemas.microsoft.com/powershell/2004/04
```

Serialization MAY indicate the XML namespace [XMLNS-2ED] using the xmlns attribute.

The name of the root XML element depends on the type of the element being serialized. Serialization of Primitive Type Objects (section 2.2.5.1) and serialization of Complex Objects (section 2.2.5.2) describe in detail serialization of different types of objects.

The PowerShell Remoting Protocol is only responsible for transferring the XML between the PowerShell client and PowerShell server. The higher layer uses the information provided in section 2 to construct the object from the XML.

2.2.5.1 Serialization of Primitive Type Objects

The following sections specify a complete list of primitive types, and describe how to serialize Primitive Type Objects. A Primitive Type Object is an object that contains only a value of a primitive type.

An object which in addition to a value of a primitive type contains some extra information from section 2.2.5.3.4 (i.e. ToString or extended properties) is called an Extended Primitive Object. An Extended Primitive Object is a kind of Complex Object. Serialization of Complex Objects is covered in section 2.2.5.2. Note that Extended Primitive Objects never have adapted properties (see section 2.2.5.3.4.1).

2.2.5.1.1 String

Represents a string of characters.

XML Element: <S>

XML Content: follows the XML schema specification [XMLSCHEMA2] for the string data type. Contents of the string MUST be encoded as described in section 2.2.5.3.2.

Example:

```
<S>This is a string</S>
```

2.2.5.1.2 Character

Represents a single Unicode character.

XML Element: <C>

XML Content: 16-bit unsigned integer equivalent to the specified Unicode character, serialized as described in XML schema specification [XMLSCHEMA2] for the unsignedShort data type.

Example:

```
<!-- serialization of character "a" -->
<C>97</C>
```

2.2.5.1.3 Boolean

Represents a Boolean (TRUE/FALSE) value.

XML Element: <B>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the boolean data type.

Example:

```
<B>true</B>
```

2.2.5.1.4 Date/Time

Represents a date and time.

XML Element: <DT>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the dateTime data type with the exception of making timezone information mandatory.

Example:

```
<DT>2008-04-11T10:42:32.2731993-07:00</DT>
```

2.2.5.1.5 Duration

Represents a length of time.

XML Element: <TS>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the duration data type.

Example:

```
<!-- 9 seconds, 26.9026 milliseconds -->
<TS>PT9.0269026S</TS>
```

2.2.5.1.6 Unsigned Byte

Represents an unsigned byte (8 bits).

XML Element: <By>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the unsignedByte data type.

Example:

```
<By>254</By>
```

2.2.5.1.7 Signed Byte

Represents a signed byte (8 bits).

XML Element: <SB>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the byte data type.

Example:

```
<SB>-127</SB>
```

2.2.5.1.8 Unsigned Short

Represents an unsigned short (16 bits).

XML Element: <U16>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the unsignedShort data type.

Example:

```
<U16>65535</U16>
```

2.2.5.1.9 Signed Short

Represents a signed short (16 bits).

XML Element: <I16>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the short data type.

Example:

```
<I16>-32767</I16>
```

2.2.5.1.10 Unsigned Int

Represents an unsigned integer (32 bits).

XML Element: <U32>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the unsignedInt data type.

Example:

```
<U32>4294967295</U32>
```

2.2.5.1.11 Signed Int

Represents an signed integer (32 bits).

XML Element: <I32>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the int data type.

Example:

```
<I32>-2147483648</I32>
```

2.2.5.1.12 Unsigned Long

Represents an unsigned long (64 bits).

XML Element: <U64>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the unsignedLong data type.

Example:

```
<U64>18446744073709551615</U64>
```

2.2.5.1.13 Signed Long

Represents a signed long (64 bits).

XML Element: <I64>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the long data type.

Example:

```
<I64>-9223372036854775808</I64>
```

2.2.5.1.14 Float

Represents IEEE single-precision 32-bit floating point type [IEEE754].

XML Element: <Sg>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the float data type.

Example:

```
<Sg>12.34</Sg>
```

2.2.5.1.15 Double

Represents IEEE double-precision 64-bit floating point type [IEEE754].

XML Element: <Db>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the float data type.

Example:

```
<Db>12.34</Db>
```

2.2.5.1.16 Decimal

Represents arbitrary precision decimal numbers as defined in [ECMA-335].

XML Element: <D>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the decimal data type.

Example:

```
<D>12.34</D>
```

2.2.5.1.17 Array of Bytes

Represents an array of bytes.

XML Element: <BA>

XML Content: contents of the byte array represented as a string in base64-encoding [RFC3548]

Example:

```
<!-- array with 4 bytes: {1, 2, 3, 4} -->
<BA>AQIDBA==</BA>
```

2.2.5.1.18 GUID

Represents a 16-byte (128-bit) number which is assumed to be unique in any context as defined in [RFC4122].

XML Element: <G>

XML Content: UUID string representation defined by [RFC4122].

Example:

```
<G>792e5b37-4505-47ef-b7d2-8711bb7affa8</G>
```

2.2.5.1.19 URI

Represents a Uniform Resource Identifier (URI) reference as defined in section 4 of [RFC2396], as amended by [RFC2732].

XML Element: <URI>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the uriReference data type. Contents of the URI MUST be encoded as described in section 2.2.5.3.2 below.

Example:

```
<URI>http://www.microsoft.com/</URI>
```

2.2.5.1.20 Null Value

Represents a NULL value.

XML Element: <Nil>

XML Content: Empty element

Example:

```
<Nil />
```

2.2.5.1.21 Version

Represents a version number that consists of two to four components: major, minor, build, and revision.

XML Element: <Version>

XML Contents: Version is represented as a string and serialized using XML schema specification for string data type. String representation of a version is "major.minor[.build[.revision]]" (optional components are shown in square brackets). All defined components MUST be integers greater than or equal to 0. For example, if the major number is 6, the minor number is 2, the build number is 1, and the revision number is 3, then string representation of the version would be "6.2.1.3".

Example:

```
<Version>6.2.1.3</Version>
```

2.2.5.1.22 XML Document

Represents an XML document as defined in [XML].

XML Element: <XD>

XML Content: XML document represented as a string, serialized using XML schema specification for string data type. String representation of the XML document MUST be encoded as described in the following section 2.2.5.3.2.

Example:

```
<XD>&lt;name attribute="value"&gt;Content&lt;/name&gt;</XD>
```

2.2.5.1.23 ScriptBlock

Represents a block of PowerShell script.

XML Element: <SBK>

XML Content: The contents of the **ScriptBlock** represented as a string, serialized using XML schema specification for string data type. String representation of the ScriptBlock MUST be encoded as described in the following section 2.2.5.3.2.

Example:

```
<SBK>get-command -type cmdlet</SBK>
```

2.2.5.1.24 Secure String

Represents a string that SHOULD be protected from eavesdropping and modification (that is, a password).

XML Element: <SS>

XML Content: The contents of the SecureString encrypted with the AES-256 algorithm [FIPS197] in Cipher Block Chaining Mod as specified in [SP800-38A] section 6.2, using the session key (see section 3.1.1.2.7 and/or 3.2.1.2.7) and encoded in base64 format. The key exchange MUST take place before sending a PowerShell Remoting Protocol message (section 2.2.1) containing a SecureString.

Example:

```
<SS>
bs7MU5rXWiJF7UZcgbJtYUAX55zJJFuCyDsFx2AOgb0BwFjmZso6+0dZj9dU9JfhyE9TQqi4hFTX6INJYOb54lW12eN6l
yHBXCS9EwsfCkOpfpSEnDhGZd0gxCDHmUvM5+fy5zlwL+5m3FtxSWsye/OgCZwlyPoa2EwUaq8uCE4ymuDeQ5vt1nMJEl
RFre8/paddAqHHGebGEepwW6coLdoiG2EuIwk0n+cmXyNzYJNnn/CEMpDTDsFNnkrp4CyIVfOEsn4cFjGhDkpj3qHMubV
Wy29F2f1n3ztJDNf4IX07q+xJeX8ncmFn70FNiFSONizkLD3APKFl9zSIBF6AzQ==
</SS>
```

2.2.5.1.25 Progress Record

Represents the status of an ongoing operation at a point in time.

XML Element: <PR>

XML Content: The following data is included (all strings MUST be encoded as described in section 2.2.5.3.2; elements containing integers follow XML schema [XMLSCHEMA2] specification for int data type).

Activity: An <AV> XML element with a string describing the activity for which progress is being reported.

ActivityId: An <AI> XML element with an integer identifying the activity for which progress is being reported.

CurrentOperation: An <CO> XML element with a string describing the current operation of the many required to accomplish the activity (such as copying sample.txt).

ParentActivityId: An <PI> XML element with an integer identifying the parent activity for which this record is a subordinate; a negative value indicates that the activity for which progress is being reported has no parent

PercentComplete: An <PC> XML element with an integer with an estimate of the percentage of total work that is completed for the activity

RecordType: An <T> XML element with a string indicating if the activity is in progress (Processing string) or complete (Completed string).

SecondsRemaining: An <SR> XML element with an integer estimating of time needed to complete the activity for which progress is being reported

StatusDescription: An <SD> XML element with a string containing the current status of the operation; for example, 35 of 50 items copied, 95% completed, or 100 files purged.

Example:

```
<PR>
  <AV>activity description</AV>
  <AI>1</AI>
  <Nil />
  <PI>-1</PI>
  <PC>-1</PC>
  <T>Processing</T>
  <SR>-1</SR>
  <SD>status description</SD>
</PR>
```

2.2.5.2 Serialization of Complex Objects

This section describes how to serialize Complex Objects. A Complex Object is one of the following:

- An object of a non-primitive type (a type not covered in the section 2.2.5.1).
- An Extended Primitive Object - an object which in addition to a value of a primitive type (a type covered in section 2.2.5.1) contains some extra information from section 2.2.5.3.4 (for example, ToString or extended properties).

A Complex Object sent by the higher layer to the PowerShell Remoting Protocol for transport MUST have been encoded using one of the following representations.

- As a reference to an earlier object (section 2.2.5.2.1).
- As an <Obj> Element (section 2.2.5.2.2).

The higher layer may choose to encode a subset of the Complex Object's properties, or may choose to represent the Complex Object as a string. The type of the source Complex Object may be lost in the encoding.

2.2.5.2.1 Referencing Earlier Objects

2.2.5.2.1.1 RefId Attribute

All <Obj> elements representing Complex Objects (see section 2.2.5.2.2) SHOULD have an optional RefId attribute that identifies the object so that it can be referenced later. The object identifier used MUST be unique during the lifetime of a serializer/deserializer pair (see the following section for details). The identifier can be any string that is valid in an XML attribute.

2.2.5.2.1.2 <Ref> Element

When a particular object has been already serialized by a given instance of the serializer (see the following section 2.2.5.3.3 for details of serializer lifetime), the serializer SHOULD choose to output only <Ref> element (instead of <Obj> element with full object data).

Example:

```
<!-- there are 2 objects in the list - the second object is the same as the first object -->
```

```
<Obj><LST>
  <Obj RefId="RefId-0">
    <TN RefId="RefId-0">
      <T>System.Drawing.Point</T>
      <T>System.ValueType</T>
      <T>System.Object</T>
    </TN>
    <ToString>{X=12,Y=34}</ToString>
    <Props>
      <B N="IsEmpty">false</B>
      <I32 N="X">12</I32>
      <I32 N="Y">34</I32>
    </Props>
  </Obj>
  <Ref RefId="RefId-0" />
</LST></Obj>
```

2.2.5.2.2 <Obj> Element

The <Obj> element can include the following subelements in any order.

- Type names (section 2.2.5.2.3).
- ToString (section 2.2.5.2.4).
- Element generated by one of the following:
  - Value of a primitive type (when the Complex Object is an Extended Primitive Object) (section 2.2.5.2.5).
  - Contents of known containers (section 2.2.5.2.6).
  - Contents of enums (section 2.2.5.2.7).
- Adapted Properties (section 2.2.5.2.8).
- Extended properties (section 2.2.5.2.9).

2.2.5.2.3 Type Names

Serialization of Complex Objects can include a list of type names (see section 2.2.5.3.4.5). Serialization MUST preserve the information provided by the higher layer about type names of an object. As specified in section 2.2.5.3.4.5, an object might not provide any type names at all, in which case the <TN> and <TNRef> elements MUST be omitted.

If the type information has been already serialized earlier in the same instance of the serializer, this information can be referenced using the <TNRef> element with the RefId attribute set to the identity of the earlier result of serializing type information. If type information has not been serialized earlier, a <TN> element is written.

The <TN> element contains <T> elements, each of which contains the name of a type associated with the object being serialized. <T> elements MUST be ordered from the most specific (that is, point) to least specific (that is, object). Type names MUST be encoded as described in section 2.2.5.3.2. Mapping type names to concrete types is outside the scope of the protocol and is an implementation detail.

The <TN> element always has a RefId attribute which identifies the type information; the <TN> element may be referenced later by <TNRef> elements. The type identifier used MUST be unique during the lifetime of a serializer/deserializer pair (see section 2.2.5.3.3 for details). The identifier can be any string that is valid in an XML attribute.

Example:

```
<Obj><LST>
  <Obj RefId="RefId-0">
    <TN RefId="RefId-0">
      <T>System.Drawing.Point</T>
      <T>System.ValueType</T>
      <T>System.Object</T>
    </TN>
    <ToString>{X=12,Y=34}</ToString>
    <Props>
      <B N="IsEmpty">false</B>
      <I32 N="X">12</I32>
      <I32 N="Y">34</I32>
    </Props>
  </Obj>
  <Obj RefId="RefId-1">
    <TNRef RefId="RefId-0" />
    <ToString>{X=56,Y=78}</ToString>
    <Props>
      <B N="IsEmpty">false</B>
      <I32 N="X">56</I32>
      <I32 N="Y">78</I32>
    </Props>
  </Obj>
</LST></Obj>
```

2.2.5.2.4 ToString

Serialization of Complex Objects can include a string that represents the object (see section 2.2.5.3.4.4). Serialization MUST preserve information that the higher layer provides about string representation of an object. As described in section 2.2.5.3.4.4, an object might not provide a string representation, in which case the ToString element MUST be omitted.

XML Element: <ToString>

XML Content: Follows the XML schema specification [XMLSCHEMA2] for the "string" data type. Contents of the string MUST be encoded as described in section 2.2.5.3.2.

Example:

```
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Drawing.Point</T>
    <T>System.ValueType</T>
    <T>System.Object</T>
  </TN>
  <ToString>{X=12,Y=34}</ToString>
  <Props>
    <B N="IsEmpty">false</B>
    <I32 N="X">12</I32>
    <I32 N="Y">34</I32>
```

```
  </Props>
</Obj>
```

2.2.5.2.5 Contents of Extended Primitive Objects

If the Complex Object being serialized is an Extended Primitive Object, then the value of the primitive type is serialized as described in section 2.2.5.1.

Example (compare with the serialization of a string without notes in section 2.2.5.1.1):

```
<Obj RefId="RefId-0">
  <S>This is a string</S>
  <MS>
    <S N="Note1">My note</S>
  </MS>
</Obj>
```

2.2.5.2.6 Contents of Known Containers

2.2.5.2.6.1 Stack

The Stack container specifies a data structure for accessing a collection of elements based on a last-in, first-out order.

XML Element: <STK>

XML Contents: Results of serializing all elements of the stack, starting with the topmost element.

Example:

```
<!-- serialization of a stack created with the following pseudo code:
  s = new stack();  s.push(1);  s.push(2);  s.push(3);  -->
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Collections.Stack</T>
    <T>System.Object</T>
  </TN>
  <STK>
    <I32>3</I32>
    <I32>2</I32>
    <I32>1</I32>
  </STK>
</Obj>
```

2.2.5.2.6.2 Queue

The Queue container specifies a data structure for accessing a collection of elements based on a first-in, first-out order.

XML Element: <QUE>

XML Contents: Results of serializing all elements of the queue, starting with the first element.

Example:

```
<!-- serialization of a queue created with the following pseudo code:
  s = new queue();  s.enqueue(1);  s.enqueue(2);  s.enqueue(3);  -->
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Collections.Queue</T>
    <T>System.Object</T>
  </TN>
  <QUE>
    <I32>1</I32>
    <I32>2</I32>
    <I32>3</I32>
  </QUE>
</Obj>
```

2.2.5.2.6.3 List

The List container specifies an ordered collection of elements.

XML Element: <LST> (an alternative element may be also used: <IE>).

XML Contents: Results of serializing all elements of the collection (starting with the first element).

Example:

```
<!-- serialization of a collection created with the following pseudo code:
  a = new array();  a.add(1);  a.add(2);  a.add(3);  -->
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Object[]</T>
    <T>System.Array</T>
    <T>System.Object</T>
  </TN>
  <LST>
    <I32>1</I32>
    <I32>2</I32>
    <I32>3</I32>
  </LST>
</Obj>
```

2.2.5.2.6.4 Dictionaries

The Dictionaries container specifies an associative array; that is, a collection of keys and a collection of values in which every key is associated with one value.

XML Element: <DCT>

XML Contents: For each (key, value) pair, write <En>"key" "associated value"</En>, replacing "key" with results of serializing the key with name attribute (see section 2.2.5.3.1) set to "Key" and replacing "associated value" with results of serializing the associated value with name attribute (see section 2.2.5.3.1) set to "Value". Pairs can be processed and written in any order.

Example:

```
<!-- serialization of a dictionary created with the following pseudo code:
  d = new dictionary();  d.add("key1", 1);  d.add("key2", 2);  -->
```

```
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Collections.Hashtable</T>
    <T>System.Object</T>
  </TN>
  <DCT>
    <En><S N="Key">key2</S><I32 N="Value">2</I32></En>
    <En><S N="Key">key1</S><I32 N="Value">1</I32></En>
  </DCT>
</Obj>
```

2.2.5.2.7 Contents of Enums

Enums specify a value of an enumeration. An enumeration is a distinct type consisting of a set of named constants. Every enumeration type has an underlying type, which can be any integral type. The default underlying type of the enumeration elements is a 32-bit integer (see section 2.2.5.1.11). Enums never have adapted properties (see section 2.2.5.3.4.1).

XML Element: element corresponding to the primitive integer type (see section 2.2.5.1) that is underlying the enumeration type.

XML Contents: value of the enumeration converted to the underlying type.

Example:

```
<Obj RefId="0">
  <TN RefId="0">
    <T>System.ConsoleColor</T>
    <T>System.Enum</T>
    <T>System.ValueType</T>
    <T>System.Object</T>
  </TN>
  <ToString>Blue</ToString>
  <I32>9</I32>
</Obj>
```

2.2.5.2.8 Adapted Properties

This section describes how to serialize adapted properties (see section 2.2.5.3.4.1).

XML Element: <Props>

XML Contents: Results of serializing adapted properties of the Complex Object. Properties can be serialized in any order. Property names MUST be serialized using the attribute described in section 2.2.5.3.1.

Example:

```
<!-- serialization of an "point" object that has "X", "Y" and "IsEmpty" properties -->
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Drawing.Point</T>
    <T>System.ValueType</T>
    <T>System.Object</T>
  </TN>
```

```
  <ToString>{X=10,Y=20}</ToString>
  <Props>
    <B N="IsEmpty">false</B>
    <I32 N="X">10</I32>
    <I32 N="Y">20</I32>
  </Props>
</Obj>
```

2.2.5.2.9 Extended Properties

This section describes how to serialize extended properties (see section 2.2.5.3.4.2) and property sets (see section 2.2.5.3.4.3) of all Complex Objects.

XML Element: <MS>

XML Contents: Results of serializing values of extended properties and/or results of recursive serialization of property sets (resulting in a nested <MS> element). Properties and property sets can be serialized in any order. Property names and property set names MUST be serialized using the property name attribute described in section 2.2.5.3.1.

Example:

```
<!-- serialization of a point with 2 extended properties and with 1 property set that
contains 2 other extended properties -->
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Drawing.Point</T>
    <T>System.ValueType</T>
    <T>System.Object</T>
  </TN>
  <ToString>{X=10,Y=20}</ToString>
  <Props>
    <B N="IsEmpty">false</B>
    <I32 N="X">10</I32>
    <I32 N="Y">20</I32>
  </Props>
  <MS>
    <S N="Property1">This is an extended property</S>
    <S N="Property2">This is a second extended property</S>
    <MS N="PropertySet1">
      <S N="Property3">This is a third extended property</S>
      <S N="Property4">This is a forth extended property</S>
    </MS>
  </MS>
</Obj>
```

2.2.5.3 Miscellaneous

2.2.5.3.1 Property Name

If the serialized object was associated with a property, then the XML element representing the serialized object will have an N attribute that represents the name of that property. Property names MUST be encoded as described in section 2.2.5.3.2.

Example:

```
<!-- serialization of an "point" object that has "X", "Y" and "IsEmpty" properties -->
<Obj RefId="RefId-0">
  <TN RefId="RefId-0">
    <T>System.Drawing.Point</T>
    <T>System.ValueType</T>
    <T>System.Object</T>
  </TN>
  <ToString>{X=10,Y=20}</ToString>
  <Props>
    <B N="IsEmpty">false</B>
    <I32 N="X">10</I32>
    <I32 N="Y">20</I32>
  </Props>
</Obj>
```

2.2.5.3.2 Encoding Strings

Some strings require encoding before they can be used in XML output, to remove invalid surrogate pairs for example. In the sections that follow, the descriptions of strings which require encoding will explicitly cite this section; strings with descriptions that lack such a citation can be serialized without encoding them first.

This method translates some characters into escaped numeric entity encodings.

The escape character is "_". Control characters and surrogate characters are escaped as _xHHHH_, where HHHH string stands for the four-digit hexadecimal UCS-2 code for the character in most significant bit first order.

For example, the "Order\nDetails" is encoded as:

```
Order_x000A_Details
```

The underscore character only requires escaping when it is followed by a character sequence that, together with the underscore, can be misinterpreted as an escape sequence when **decoding** the name. For example, Order_Details is not encoded, but Order_x0020_ is encoded as Order_x005f_x0020_. No short forms are allowed. For example, the forms _x20_ and __ are not generated.

2.2.5.3.3 Lifetime of a Serializer/Deserializer Pair

The serialization used in the PowerShell remoting protocol makes certain assumptions about lifetime of a serializer/deserializer pair. These assumptions are used in managing uniqueness of object identifiers (section 2.2.5.2.1) and type identifiers (section 2.2.5.2.3) used by the serializer.

A new serializer/deserializer pair MUST be created and reused for each type of message data that is specified in section 2.2.2 and sent across the network.

2.2.5.3.4 Structure of Complex Objects

2.2.5.3.4.1 Adapted Properties

Adapted properties are name/value pairs exposed by the core definition of an object.

Example 1: A .NET object representing a point can have a property named X with an associated value equal to 123 and a property named Y with an associated value equal to 456.

Example 2: A WMI object representing a computer system can have a property named Model with an associated value equal to HP Compaq dc7800 Convertible Minitower.

2.2.5.3.4.2 Extended Properties

Extended properties are name/value pairs added to an object outside of the core definition of an object.

Example: A .NET object representing a point can have 2 adapted properties named X and Y. A pipeline executing on a PowerShell server can add extended properties to some instances of this object, for example, a property named Label with a value of My Location.

2.2.5.3.4.3 Property Sets

A property set is a named collection of properties.

2.2.5.3.4.4 ToString Value

A ToString value is an optional string representation of the object provided and used by the higher layer for display purposes. The PowerShell Remoting Protocol MUST transparently pass this value (or lack of the value) between the higher layers on the client and server without interpretation.

2.2.5.3.4.5 Type Names

An object can be associated with a list of type names. The list of type names is optional, and an object might not have any type names associated with it.

If a list of type names is associated with an object, the PowerShell Remoting Protocol MUST transparently pass it between the higher layers on the client and server without interpretation.

2.2.6 Encoding Host Parameters in Host Method Calls

The parameters of a host method call are encoded as follows:

- A list of parameters is constructed.
- Each element of the list is encoded using the rules described specified in 2.2.6.1. This depends on the type of the parameter.
- The list is then converted into UTF-8 encoded XML, equivalent to the XML created by serializing an object with extended properties (see section 2.2.5.2.9).

2.2.6.1 Encoding Individual Parameters

The following sections specify how individual parameters are encoded.

2.2.6.1.1 Any Serializable Type

Any type which can be serialized as described in 2.2.5 is not encoded.

2.2.6.1.2 CultureInfo

The *CultureInfo* parameter is encoded by calling the ToString() method on the object. See [ECMA-335] for the definition of ToString() of CultureInfo.

2.2.6.1.3 List

The *list* parameter is encoded by constructing a new list with the elements being encoded in UTF-8 XML format which is equivalent to an XML obtained by serializing an object (see section 2.2.5) with the following extended properties (see section 2.2.5.2.9).

Property Name: T.

Property Value: Type name of the element as defined in [ECMA-335]

Property Type: String

Property Name: V.

Property Value: Element encoded using rules described in section 2.2.6.1

Property Type: List (encoded as defined in section 2.2.5.2.6.3).

2.2.6.1.4 Array

Represents a (potentially multi-dimensional) array of elements.

An array is encoded in UTF-8 encoded XML, which is equivalent to the XML obtained by serializing a Complex Object (section 2.2.5.2) object (see section 2.2.5) with the following extended properties (see section 2.2.5.2.9).

Property Name: mae.

Property Value: Elements of the array are flattened into a List and ordered by first listing the deepest elements. For example for a 3-dimensional array where dimensions are 2,3,2, the order of elements is: a[0,0,0], a[0,0,1], a[0,1,0], a[0,1,1], a[0,2,0], a[0,2,1], a[1,0,0], a[1,0,1], a[1,1,0], a[1,1,1], a[1,2,0], a[1,2,1].

Property Type: List (see section 2.2.5.2.6.3).

Property Name: mal.

Property Value: Sizes of each of the dimensions of the array, from the topmost to the deepest dimension.

Property Type: List (see section 2.2.5.2.6.3) of Signed Ints (see section 2.2.5.1.11). The List MUST have at least one element.

2.2.6.1.5 Collection

The *collection* parameter is encoded like a list as defined in section 2.2.6.1.3

2.2.6.1.6 Dictionary

The *dictionary* paramater is encoded by constructing a new hash table with the following key/value pairs:

Key: Key in the dictionary, encoded using rules described in section 2.2.6.1

Value: Value corresponding to key in dictionary, encoded using rules described in section 2.2.6.1

2.2.6.1.7 Object Dictionary

The *object dictionary* parameter is encoded by constructing a new hash table with the following key/value pairs:

Key: Key in the dictionary, encoded using rules described in section 2.2.6.1

Value: UTF-8 encoded XML that is equivalent to the XML created by serializing an object with the following extended properties (see section 2.2.5.2.9).

Property Name: T.

Property Value: Type name of the element as defined in [ECMA-335]

Property Type: String

Property Name: V.

Property Value: Value corresponding to key in dictionary, encoded using rules described in section 2.2.6.1

Property Type: List (encoded as defined in section 2.2.5.2.6.3).

2.2.6.1.8 Other Object Types Used in a Host Call

The non-null properties of any other object types used in a host call, as defined in section 2.2.3, are encoded as extended properties (see section 2.2.5.2.9) in the following manner.

Property Name: Name of the object's property

Property Value: The value of the object's property encoded as described in section 2.2.6.1. and then encoded into UTF-8 XML as described in section 2.2.5

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

3.1.1.1 Global Data

Global client data MUST be initialized as described in section 3.1.3.

3.1.1.1.1 MS-WSMV ShellID to RunspacePool Table

The PowerShell client MUST maintain a global table that maps a Web Services Management Protocol Extensions for Windows Vista **[MS-WSMV]** shell to data associated with a RunspacePool (see section 3.1.1.2) to a RunspacePool (see section 3.1.1.2).

The key used in the table is the value of the ShellID selector received in the wxf:ResourceCreated message (see [MS-WSMV] section 3.1.4.5.2).

3.1.1.1.2 MS-WSMV CommandId to Pipeline Table

The PowerShell client MUST maintain a global table that maps a Web Services Management Protocol Extensions for Windows Vista [MS-WSMV] command to data associated with a pipeline (see section 3.1.1.3).

The key used in the table is the value of the commandId element received in the wxf:Command Response message (see [MS-WSMV] section 2.2.4.8).

3.1.1.1.3 Public Key Pair

The PowerShell client MUST have an RSA public key pair [PKCS1] (public key MUST be 2048-bit) that can be used in a key exchange (see sections 3.1.5.4.3, 3.1.5.4.4 and 3.1.5.4.5). The same public key pair MUST be used in all key exchanges.

The public key pair MUST be generated before the first PUBLIC_KEY message (see section 3.1.5.4.3) is sent from the client to the server. The client MAY generate the public key pair when the client starts running.

3.1.1.2 RunspacePool Data

3.1.1.2.1 GUID

Each RunspacePool has an associated GUID. The GUID is generated by the PowerShell client after the higher layer triggers the creation of a RunspacePool (section 3.1.4.1) and before the corresponding wxf:Create message is sent.

3.1.1.2.2 RunspacePool State

Each RunspacePool has an associated state. Section 2.2.3.4 specifies available states and describes the data types used to encode the states in the PowerShell remoting protocol messages.

For details about how a RunspacePool state transitions from its initial state of Opening to the state of Opened, see the RunspacePool creation process specified in section 3.1.4.1.

From the Opened state, a RunspacePool can reach either the Closed or Broken state specified in section 2.2.3.4.

A PowerShell client can close a RunspacePool by sending a wxf:Delete message (section 3.1.5.3.11). Before sending this message, the PowerShell client changes the RunspacePool state to Closing and stops any executing pipelines (section 3.1.4.4) using the pipeline table (section 3.1.1.2.6). If there is a successful response (section 3.2.5.3.12), then the PowerShell client changes the RunspacePool state to Closed; otherwise, the PowerShell client changes the state to Broken.

For details of how a PowerShell client can disconnect from a RunspacePool, see section 3.1.4.9. For details of how a PowerShell client can connect to a RunspacePool, see section 3.1.4.10.

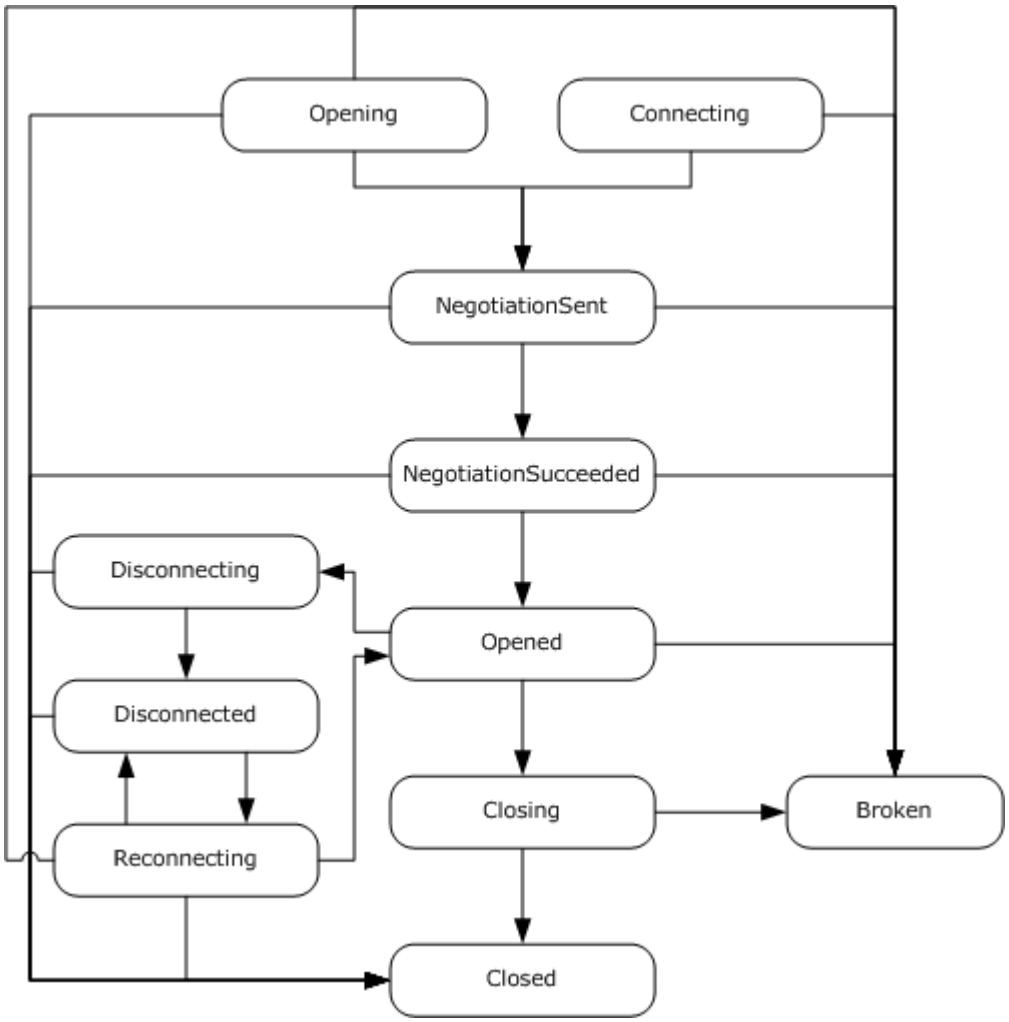


Figure 2: Client RunspacePool states and transitions

3.1.1.2.3 Defragmentation Data

The current state of **defragmentation** (see sections 2.2.4 and 3.1.5.1.2) for PSRP messages (section 2.2.1) sent by the PSRP server and targeted at the RunspacePool.

Defragmentation data consists of the following pieces of information:

- **LastObjectId**: contents of **ObjectId** field of the last received fragment. Initialized to 0.
- **LastFragmentId**: contents of **FragmentId** field of the last received fragment. Initialized to 0.
- **PartiallyDefragmentedPsrpMessage**: blob with merged **Data** fields from all fragments with **ObjectId** equal to the value of **LastObjectId**. Initialized to an empty blob.

3.1.1.2.4 MS-WSMV Shell

Each RunspacePool has an associated Web Services Management Protocol Extensions for Windows [MS-WSMV] shell which stores the following information:

- wsa:EndpointReference (section 3.1.5.3.2).
- ShellID selector (section 3.1.5.3.2).
- ResourceURI (section 3.1.5.3.3).

3.1.1.2.5 RunspacePool Information CI Table

The PowerShell client MUST maintain a table associating an integer identifier with the following outstanding messages sent to a RunspacePool:

- The SET_MAX_RUNSPACES message (as specified in section 2.2.2.6).
- The SET_MIN_RUNSPACES message (as specified in section 2.2.2.7).
- The GET_AVAILABLE_RUNSPACES message (as specified in section 2.2.2.11).

The table is used to unblock the higher layer when a RunspacePool response (see section 2.2.2.8) is received, and to route the response to the higher-layer.

3.1.1.2.6 Pipeline Table

Each RunspacePool maintains a table representing the pipelines that are currently executing using the RunspacePool.

3.1.1.2.7 Session Key

The PowerShell client MUST store and reuse the session key received from the server in the ENCRYPTED_SESSION_KEY message (section 2.2.2.4). There is no initialization—the key is created on demand.

3.1.1.2.8 SessionKeyTransferTimeoutms

The idle time-out, in milliseconds, between a PowerShell client sending the PUBLIC_KEY message (section 3.1.5.4.3) and the PowerShell client receiving the ENCRYPTED_SESSION_KEY message (section 3.1.5.4.4). This element SHOULD be initialized to 60000.

3.1.1.3 Pipeline Data

3.1.1.3.1 GUID

Each pipeline has an associated GUID. The GUID is generated by the PowerShell client after the higher layer triggers the execution of a pipeline (section 3.1.4.3) and before the corresponding wxf:Command message is sent.

3.1.1.3.2 Pipeline State

Each pipeline has an associated state. Section 2.2.3.5 specifies available states and describes the data type used to encode the state in the PowerShell remoting protocol messages.

For details about how pipeline state transition happens on the client side, see the steps involved in executing a pipeline specified in section 3.1.4.3.

A PowerShell client can stop an executing pipeline at any time by sending a wxf:Signal message (section 3.1.4.4). Before sending this message, the PowerShell client changes the pipeline state to Stopping. If there is a successful response to the wxf:Signal message (section 3.2.5.3.10), then the PowerShell client changes the pipeline state to Stopped; otherwise, the PowerShell client changes the state to Failed.

If a PowerShell server sends a State Information message (section 3.1.5.4.21) with a Failed state, then the PowerShell client MUST process this message and change the pipeline state to Failed accordingly.

When the pipeline state is changed to Completed or Stopped or Failed, the PowerShell client removes the pipeline from the corresponding RunspacePool's pipeline table (section 3.1.1.2.6) and the global pipeline table (section 3.1.1.1.2).

When the pipeline state is changed to Completed or Stopped or Failed, the PowerShell client MUST not send any more messages to the PowerShell server targeted to that particular pipeline.

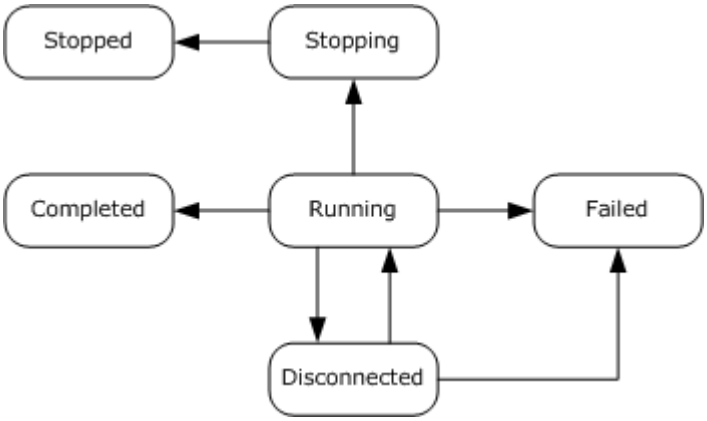


Figure 3: Client pipeline states and transitions

3.1.1.3.3 Defragmentation Data

The current state of defragmentation (see sections 2.2.4 and 3.1.5.1.2) for PSRP messages (section 2.2.1) sent by the PSRP server and targeted at the pipeline.

Defragmentation data for a pipeline contains exactly the same type information as defragmentation data for a RunspacePool (section 3.1.1.2.3).

3.1.1.3.4 MS-WSMV Command

Each pipeline has an associated Web Services Management Protocol Extensions for Windows Vista [MS-WSMV] command storing the following information:

- CommandId (as specified in section 3.1.5.3.4).

3.1.2 Timers

The PowerShell remoting protocol defines one timer in addition to those of the Web Services Management Protocol Extensions for Windows Vista [MS-WSMV].

The Session Key transfer timer MUST trigger closure of an associated RunspacePool if an ENCRYPTED_SESSION_KEY message (section 3.1.5.4.4) is not received from the server in the number of milliseconds specified by the SessionKeyTransferTimeoutms (section 3.1.1.2.8).

3.1.3 Initialization

Client Initialization

- The tables specified in sections 3.1.1.1.1 and 3.1.1.1.2 MUST be initialized to empty.
- The state of a newly created RunspacePool (section 3.1.1.2.2) MUST be initialized to Opening.
- The RunspacePool Information CI Table (section 3.1.1.2.5) MUST be initialized as empty.

Pipeline Initialization

- The state of a newly created pipeline (section 3.1.1.3.2) MUST be initialized to Running.

3.1.4 Higher-Layer Triggered Events

The following sections describe how the higher-layer triggers various PowerShell remoting protocol events. For more information about how a PowerShell Remoting Protocol message is sent from the client to the server, see section 3.1.5.1.

3.1.4.1 Creating a RunspacePool

The higher-layer triggers the RunspacePool creation on the client. The following activities happen as part of the RunspacePool creation. During the RunspacePool creation time, the PowerShell client sends PowerShell messages to a PowerShell server and receives PowerShell messages back from the PowerShell server. The PowerShell client expects certain specific PowerShell messages from the server at each stage, as described later in this section. If PowerShell client does not receive expected messages at each stage, then the PowerShell client terminates the RunspacePool creation and notifies the higher-layer. If a wxf:Fault message is received at any stage, the PowerShell client reports the failure to the higher-layer, closes the RunspacePool as specified in section 3.1.5.3.13, and terminates the RunspacePool creation.

1. The PowerShell client creates a new RunspacePool, assigns a unique GUID to this RunspacePool as described in section 3.1.1.2.1, and initializes the RunspacePool state to Opening as described in section 3.1.1.2.2.

2. The PowerShell client constructs a SESSION_CAPABILITY message (as specified in section 2.2.2.1) and an INIT_RUNSPACEPOOL message (section 2.2.2.2). The PowerShell client then constructs fragmented messages for these PowerShell messages using the rules specified in section 3.1.5.1.1.

3. The PowerShell client MUST use wxf:Create (section 3.1.4.5.2) to create a RunspacePool on the server. While sending the wxf:Create message, the PowerShell client sends as many fragments as possible from step 2, along with the wxf:Create message, using the <open content> portion, as specified in section 3.1.5.3.1. If all fragments of the SESSION_CAPABILITY message have been sent, then the PowerShell client changes the RunspacePool state (section 3.1.1.2) to NegotiationSent; otherwise, the RunspacePool state change is delayed until step 6.

4. If the PowerShell client receives a wxf:ResourceCreated message, the PowerShell client stores the ShellID from the response (sections 3.1.1.1.1 and 3.1.1.2.4), as specified in section 3.1.5.3.1. If the PowerShell client receives a wxf:Fault message, the PowerShell client reports the failure to the higher-layer and terminates RunspacePool creation.

5. At this point, the PowerShell client has a ShellID associated with the remote RunspacePool and MUST send a wxf:Receive message (section 3.1.5.3.7) to the PowerShell server to start receiving data from the PowerShell server.

   After each received wxf:ReceiveResponse message, the PowerShell client MUST send another wxf:Receive if the RunspacePool is not in a Closed or Broken state.

6. If there are any fragments left in step 3, the remaining fragments MUST be sent using one or more wxf:Send messages (as specified in section 3.1.5.3.5). If the RunspacePool state (section 3.1.1.2) was not changed to NegotiationSent in step 3, then it is changed after sending the last fragment of the SESSION_CAPABILITY message.

7. The PowerShell client expects a SESSION_CAPABILITY message (section 2.2.2.1) from the server at this stage. If a SESSION_CAPABILITY message is received, then the PowerShell client hands over the Session Capability to the higher-layer.

8. The PowerShell client changes the RunspacePool state (section 3.1.1.2) to NegotiationSucceeded.

9. The PowerShell client expects an APPLICATION_PRIVATE_DATA message (section 2.2.2.13) from the server at this stage. If an APPLICATION_PRIVATE_DATA message is received, then the PowerShell client hands over the application private data to the higher-layer.

10. The PowerShell client expects the RUNSPACEPOOL_STATE message (section 2.2.2.9) from the server at this stage. If a RUNSPACEPOOL_STATE message is received, then the PowerShell client extracts the State from the message and changes the RunspacePool state (section 3.1.1.2) to Opened.

When the RunspacePool state is in Opened state, the higher-layer can trigger other events, such as Executing a pipeline (section 3.1.4.3) or Closing the RunspacePool (section 3.1.4.2).

3.1.4.2 Closing a RunspacePool

The higher layer can initiate the closing of a RunspacePool. If the state of a RunspacePool is not Opened, then the PowerShell client does nothing. Otherwise, the following activities happen as part of the RunspacePool closure:

1. The PowerShell client stops any currently executing pipelines (section 3.1.4.4).

2. The PowerShell client sends a wxf:Delete message (section 3.1.5.3.11) using the ShellID stored in 3.1.1.2.4.

3. PowerShell client expects a wxf:DeleteResponse (section 3.2.5.3.12) from the server at this state. If wxf:DeleteResponse is received, then the PowerShell client changes the RunspacePool state (section 3.1.1.2) to Closed. If a wxf:Fault message is received, then the PowerShell client changes the RunspacePool state (section 3.1.1.2) to Broken.

4. When the RunspacePool reaches a Closed or Broken state, the PowerShell client removes the RunspacePool instance from the global table (section 3.1.1.1.1).

3.1.4.3 Executing a Pipeline

The higher layer can initiate the execution of a pipeline on the PowerShell server at any time as long as the RunspacePool is in Opened (section 3.1.1.2) state. The following activities happen as part of the pipeline execution. During the pipeline creation time, the PowerShell client sends PowerShell messages to a PowerShell server and receives PowerShell messages back from the server. The PowerShell client expects specific PowerShell messages from the server at each stage as described later in this section. If the PowerShell client does not receive the expected messages at each stage, then the PowerShell client terminates the pipeline execution (section 3.1.4.3) and notifies the higher layer. If a wxf:Fault message is received at any stage, the PowerShell client reports the failure to the higher layer and stops the pipeline (as specified in section 3.1.5.3.13).

1. The PowerShell client creates a new pipeline, assigns a unique GUID to this pipeline (section 3.1.1.3.1), and initializes the pipeline state (section 3.1.1.3.2) to Running. The PowerShell client adds this pipeline instance to the RunspacePool's pipeline table (section 3.1.1.2.6). The PowerShell client constructs a CREATE_PIPELINE message (section 2.2.2.10) and sends it to server using wxf:Command (section 3.1.5.3.3) and (if needed) wxf:Send (section 3.1.5.3.5) messages.

2. After sending all fragments of a CREATE_PIPELINE message, the PowerShell client stores the CommandId (sections 3.1.1.3.4 and 3.1.1.1.2) and MUST send a wxf:Receive message to start receiving data from the pipeline on the server. After each received wxf:ReceiveResponse message, the PowerShell client MUST send another wxf:Receive message if the pipeline is not in a Completed or Stopped state.

3. At this stage, the PowerShell client interacts with the higher layer in three ways concurrently:

   - The PowerShell client reads input data (if any) from the higher layer, constructs a PIPELINE_INPUT message (section 3.1.5.4.17), and sends it to a PowerShell server. This process is repeated for all the input objects provided by the higher layer. When the higher layer signals that all input data has been provided, the PowerShell client MUST send an END_OF_PIPELINE_INPUT message (section 2.2.2.18).

   - The PowerShell client receives result messages from the PowerShell server and hands over the result data to the higher layer. Only the following result messages are expected at this stage: PIPELINE_OUTPUT (section 2.2.2.19), ERROR_RECORD (section 2.2.2.20), DEBUG_RECORD (section 2.2.2.22), VERBOSE_RECORD (section 2.2.2.23), WARNING_RECORD (section 2.2.2.24), PROGRESS_RECORD (section 2.2.2.25), PIPELINE_HOST_CALL (section 2.2.2.26), and PIPELINE_STATE (section 2.2.2.21). If the client receives any other message, then the client MUST stop the pipeline (section 3.1.4.3). When a PIPELINE_STATE message is received, then the PowerShell client stops sending input data and skips to step 4.

   - If the higher layer stops the pipeline 3.1.4.4, the PowerShell client does not execute steps 4 and 5.

4. If a PIPELINE_STATE message (section 3.1.5.4.21) is received, the PowerShell client changes the pipeline state (section 3.1.1.3.2) per the message received and notifies the higher-layer.

5. When the pipeline reaches Completed or Failed state, the PowerShell client removes the pipeline instance from the global table (section 3.1.1.1.2) and the RunspacePool's pipeline table (section 3.1.1.2.6).

3.1.4.4 Stopping a Pipeline

The higher-layer can choose to stop an executing pipeline. If the state of the pipeline is not Running, the PowerShell client ignores the request. Otherwise, the following activities happen as

part of stopping the pipeline. If a wxf:Fault message is received at any stage, the PowerShell client reports the failure to the higher-layer, removes the pipeline from the RunspacePool's pipeline table (section 3.1.1.2.6), removes the pipeline from the global pipeline table (section 3.1.1.1.2), and changes the pipeline State to Failed (section 3.1.1.3.2).

1. The PowerShell client waits for a wxf:CommandResponse message for the wxf:Command message (section 3.1.5.3.3) before proceeding with stopping the pipeline.

2. The PowerShell client changes the pipeline state (section 3.1.1.3.2) to Stopping and sends a wxf:Signal message (section 3.1.5.3.9) to stop the pipeline on the server.

3. The PowerShell client expects a wxf:SignalResponse message (section 3.2.5.3.10) at this stage. If a wxf:SignalResponse message is received, the PowerShell client changes the pipeline State (section 3.1.1.3.2) to Stopped, removes the pipeline from the RunspacePool's pipeline table (section 3.1.1.2.6), removes the pipeline from the global pipeline table (section 3.1.1.1.2), and notifies the higher-layer.

3.1.4.5 Getting Command Metadata

The higher layer triggers the sending of a GET_COMMAND_METADATA message (section 2.2.2.14) to get the metadata of commands (section 2.2.3.19) available in a RunspacePool. The RunspacePool MUST be in the Opened state (section 3.1.1.2). When sending this message and receiving responses from the server, the client uses similar data structures that are used for executing a pipeline (section 3.1.4.3).

The following activities happen as part of sending the GET_COMMAND_METADATA message and receiving responses from the server. The PowerShell client expects certain specific messages from the server at each stage, as described below. If the PowerShell client does not receive the expected messages at any stage, then the PowerShell client terminates the Getting command metadata higher-layer triggered action and notifies the higher layer. If a wxf:Fault message is received at any stage, the PowerShell client reports the failure to the higher layer and stops the Getting command metadata action in the same manner as described in section 3.1.5.3.13.

1. The PowerShell client creates a new pipeline data structure (section 3.1.1.3), assigns a unique GUID to this pipeline (section 3.1.1.3.1) and initializes the pipeline state (section 3.1.1.3.2) to Running.

   The PowerShell client adds this pipeline instance to the RunspacePool's pipeline table (section 3.1.1.2.6).

   The PowerShell client constructs a GET_COMMAND_METADATA message (section 2.2.2.14) and sends it to the PowerShell server.

2. If a wxf:CommandResponse message (section 3.1.5.3.4) is received, the PowerShell client stores the CommandId (sections 3.1.1.3.4 and 3.1.1.1.2) and sends a wxf:Receive message to start receiving data from the PowerShell server.

3. At this stage, the PowerShell client receives result messages from the PowerShell server and sends the result data to the higher-layer. Only the following result messages are expected at this stage. If the PowerShell client receives any other message, then the PowerShell client MUST stop the pipeline (section 3.1.4.4). The messages expected at this stage are the following: PIPELINE_OUTPUT (section 2.2.2.19) containing either CommandMetadataCount (first Output received, see section 2.2.3.21) or CommandMetadata (subsequent Output received, see section 2.2.3.22), ERROR_RECORD (section 2.2.2.20), DEBUG_RECORD (section 2.2.2.22), VERBOSE_RECORD (section 2.2.2.23), WARNING_RECORD (section 2.2.2.24),

PROGRESS_RECORD (section 2.2.2.25), PIPELINE_HOST_CALL (section 2.2.2.26) and PIPELINE_STATE (section 2.2.2.21).

The CommandMetadataCount (section 2.2.3.21) MUST be the first Output (section 2.2.2.19) message received and it specifies the number of subsequent CommandMetadata (section 2.2.3.22) Output messages received by the client. The client SHOULD process only this number of CommandMetadata Output messages.

When a PIPELINE_STATE message is received, or when the higher-layer stops the Getting command metadata action, the PowerShell client stops executing these steps.

4. If a PIPELINE_STATE message (section 3.1.5.4.21) is received, the PowerShell client changes the pipeline state (section 3.1.1.3.2) as per the message received and notifies the higher layer.

5. When the pipeline reaches the Completed or Failed state, the PowerShell client removes the pipeline instance from the global pipeline table (section 3.1.1.1.2) and the RunspacePool's pipeline table (section 3.1.1.2.6).

3.1.4.6 Setting the Minimum or Maximum Runspaces in a RunspacePool

The higher layer can initiate setting minimum or maximum (section 3.2.1.2.9) runspaces in a RunspacePool on the PowerShell server at any time as long as the RunspacePool is in an Opened (section 3.1.1.2.5) state. The following activities happen as part of setting the minimum or maximum runspaces in a RunspacePool:

1. The PowerShell client creates a new entry in the **RunspacePool CI Table** (section 3.1.1.2.5) and blocks the higher layer until step 4.

2. The PowerShell client constructs a SET_MAX_RUNSPACES (section 2.2.2.6) or SET_MIN_RUNSPACES (section 2.2.2.7) message and sends it (section 3.1.5.1.1) to the server using a wxf:Send message.

3. The PowerShell client waits to receive (section 3.1.5.1.2) a RUNSPACE_AVAILABILITY message (section 2.2.2.8) associated with the **RunspacePool CI Table** entry from step 1. This step assumes that the client has already sent out a wxf:Receive message for the RunspacePool as specified in section 3.1.4.1.

4. The PowerShell client removes the **RunspacePool CI Table** entry, unblocks the higher-layer, and communicates the result extracted from the SetMinMaxRunspacesResponse field of the received RUNSPACE_AVAILABILITY message.

3.1.4.7 Getting the Number of Available Runspaces in a RunspacePool

The higher layer can initiate getting the number of available (section 3.2.1.4.1) runspaces in a RunspacePool on the PowerShell server at any time as long as the RunspacePool is in an Opened (section 3.1.1.2) state. The following activities happen as part of getting the number of available runspaces in a RunspacePool:

1. The PowerShell client creates a new entry in the **RunspacePool CI Table** (section 3.1.1.2.5) and blocks the higher-layer until step 4.

2. The PowerShell client constructs a GET_AVAILABLE_RUNSPACES message (section 2.2.2.11) and sends it (section 3.1.5.1.1) to the server using a wxf:Send message.

3. The PowerShell client waits to receive (section 3.1.5.1.2) a RUNSPACE_AVAILABILITY message (section 2.2.2.8) associated with the **RunspacePool CI Table** entry from step 1. This step

assumes that the client has already sent out a wxf:Receive message for the RunspacePool as specified in section 3.1.4.1.

4. The PowerShell client removes the **RunspacePool CI Table** entry, unblocks the higher layer, and communicates the result extracted from the **SetMinMaxRunspacesResponse** field of the received RUNSPACE_AVAILABILITY message.

3.1.4.8 Initiating a Session Key Exchange

The higher layer can initiate a session key exchange at any time, so long as the RunspacePool is in an Opened state (section 3.1.1.2).

1. The PowerShell client ignores this higher-layer request if either of the following is true:

   - The session key (section 3.1.1.2.7) is already registered by the PowerShell client.
   - The session key exchange is already in progress.

   If this higher-layer request is ignored, then steps 2 and 3 of this procedure are skipped.

2. The PowerShell client constructs a PUBLIC_KEY message (section 2.2.2.3) and sends it to PowerShell server using a wxf:Send message (see section 3.1.5.1.1.

3. The PowerShell client waits to receive an ENCRYPTED_SESSION_KEY (as specified in section 2.2.2.4) from the PowerShell server (see section 3.1.5.1.2) and updates the abstract data (see section 3.1.1.2.7).

4. The PowerShell client notifies the higher-layer when the session key exchange is completed.

3.1.4.9 Disconnecting from a RunspacePool

In order for the server session to support Disconnect and Connect operations, the client MUST provide a wsmv:SessionId element ([MS-WSMV] (section 3.1.4.1.37)) in all wxf messages. This element is the unique identifier of a client session and will remain the same for all messages sent from that session. Server sessions supporting Disconnect and Connect operations distinguish requests from different client sessions based on this identifier.

The higher layer can initiate the process of disconnecting from a RunspacePool. Any active pipelines will automatically be disconnected once the RunspacePool is disconnected. If the RunspacePool is not in the Opened state, the PowerShell client ignores any requests to disconnect. Otherwise, the PowerShell client takes the following actions to process the disconnect request:

1. The PowerShell client waits for any ongoing send operation to complete by waiting for wxf:SendResponse messages (see section 3.1.5.3.6) from the server.

2. The PowerShell client sends a wxf:Disconnect message (see section 3.1.5.3.16) using the ShellID specified in section 3.1.1.2.4.

3. The PowerShell client receives a wxf:DisconnectResponse (see section 3.2.5.3.17) from the server. The PowerShell client changes the states of the RunspacePool (see section 3.1.1.2) and any associated pipelines to Disconnected. If the client receives a wxf:Fault message, it changes the RunspacePool state to Broken.

3.1.4.10 Connecting to a RunspacePool

After a client disconnects from a RunspacePool, that same RunspacePool can be reconnected to by the previous client session or by a new client session. When a previous client reconnects, the server

session recognizes it based on the client's wsmv:SessionId element (see [MS-WSMV] section 3.1.4.1.37). When a new client session connects to the RunspacePool, the client and server exchange messages to negotiate a new session identifier.

3.1.4.10.1 Discovering Disconnected RunspacePools and Associated Pipelines on a PowerShell Server

Before connecting to a RunspacePool on a PowerShell server, the client needs to obtain an identifier for that RunspacePool. Each RunspacePool instance is represented as a WSMan Shell instance. Clients can use the wxf:Enumerate request (as specified in [MS-WSMV]) to obtain a list of ShellID values, which are the RunspacePool identifiers.

The PowerShell client uses the identifier for the RunspacePool it intends to connect to in the wxf:Connect message that initiates the connection process. See section 3.1.4.10.3 for details.

Once a client has connected to a RunspacePool, it can enumerate the pipelines in the RunspacePool and connect to a particular pipeline to receive that pipeline's output. Each pipeline is represented as a WSMan Command instance. Clients again use the wxf:Enumerate request to obtain a list of CommandID values, which are the pipeline identifiers.

The PowerShell client sends another wxf:Connect message with the pipeline identifier to initiate a connection to that pipeline. See section 3.1.4.10.3 for details.

3.1.4.10.2 Connecting to a RunspacePool from a Previous Client Session

A client session that has previously disconnected from a remote RunspacePool can reconnect by using the wxf:Reconnect message (section 3.1.5.3.18). The client sends the same wsmv:SessionId (see [MS-WSMV] (section 3.1.4.1.37)) value that it used in the original connection to that RunspacePool.

1. The PowerShell client sends a wxf:Reconnect message, using the ShellID as specified in 3.1.1.2.4.

2. The PowerShell client receives a wxf:ReconnectResponse message (section 3.2.5.3.19) from the server. The PowerShell client changes the RunspacePool state (section 3.1.1.2) to Opened. If the client receives a wxf:Fault message, it instead changes the RunspacePool state to Broken.

3. If the PowerShell client received a wxf:ReconnectResponse message in the previous step, it MUST send a wxf:Receive message (section 3.1.5.3.7) to the PowerShell server to start receiving data from the PowerShell server.

4. The PowerShell client MUST send additional wxf:Receive messages in response to any further wxf:ReconnectResponse messages it receives from the server, as long as the RunspacePool is not in either the Closed or Broken state.

3.1.4.10.3 Connecting to a RunspacePool from a New Client Session

The following procedure specifies the sequence of interactions between a PowerShell client and server when a new client connects to a disconnected RunspacePool:

1. The PowerShell client discovers the ShellID value of the RunspacePool to connect to by issuing a wsm:Enumerate message as described in section 3.1.4.10.1.

2. The PowerShell client creates a new RunspacePool, assigns the ShellID to this RunspacePool, and initializes the RunspacePool state to Connecting (section 3.1.1.2.2).

3. The PowerShell client constructs a SESSION_CAPABILITY message (section 2.2.2.1) and a CONNECT_RUNSPACEPOOL message (section 2.2.2.2). The PowerShell client then constructs fragmented messages for these PowerShell messages as specified in section 3.1.5.1.1.

4. The PowerShell client MUST send a wxf:Connect message (section 3.1.5.3.14) to create a RunspacePool on the server. The PowerShell client sends all fragments from the preceding step along with the wxf:Connect message, using the open content portion of the wxf:Connect message. The PowerShell client changes the RunspacePool state to NegotiationSent.

5. The PowerShell client receives a wxf:ConnectResponse message along with a SESSION_CAPABILITY message from the server, then passes the Session Capability to the higher layer. If the PowerShell client receives a wxf:Fault message, the PowerShell client reports the failure to the higher layer and terminates the RunspacePool connection.

6. The PowerShell client changes the RunspacePool state to Opened and sends a wxf:Receive message to the server.

7. After each wxf:ReceiveResponse message the PowerShell client receives from the server, the client MUST send another wxf:Receive as long as the RunspacePool is not in either the Closed or Broken state.

8. The PowerShell client waits for APPLICATION_PRIVATE_DATA messages (section 2.2.2.13) from the server and passes any application private data it receives to the higher layer.

When the RunspacePool state is in the Opened state, the higher layer can trigger other events such as closing the RunspacePool (section 3.1.4.2) or executing a pipeline (section 3.1.4.3). Once the RunspacePool is connected, the Powershell client can connect to individual pipelines as follows:

1. The PowerShell client discovers the Command identifier for the pipeline to connect to (section 3.1.4.10.2).

2. The PowerShell client creates a new pipeline, assigns the Command identifier to this pipeline, and initializes the pipeline state to Running (section 3.1.1.3.2).

3. The PowerShell client sends a wxf:Connect message (section 3.1.5.3.14) using the above ShellID and Command identifier and waits for a wxf:ConnectResponse message (section 3.1.5.3.15).

4. When the PowerShell client receives the wxf:ConnectResponse message from the server, it sends a wxf:Receive message to start receiving data from the pipeline on the server.

5. After each received wxf:ReceiveResponse message, the PowerShell client MUST send another wxf:Receive message as long as the pipeline is not in either the Completed or Stopped state.

6. With the pipeline connected, the PowerShell client interacts with the higher layer in the three ways specified in section 3.1.4.

7. If the PowerShell client receives a PIPELINE_STATE message (section 3.1.5.4.21), the client changes the pipeline state (section 3.1.1.3.2) in accordance with the message and notifies the higher layer.

8. When the pipeline reaches either the Completed or Failed state, the PowerShell client removes the pipeline instance from the global table (section 3.1.1.1.2) and from the RunspacePool's pipeline table (section 3.1.1.2.6).

9. If a wxf:Fault message is received at any step in this procedure, the PowerShell client reports the failure to the higher layer.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 General Rules

The PowerShell Remoting Protocol MUST adhere to the message processing rules specified in [MS-WSMV] section 3.1.4.1.31, in addition to the following.

1. The PowerShell client uses wxf:Send (section 3.1.5.3.5), wxf:Create (section 3.1.5.3.3), and wxf:Command (section 3.1.5.3.3) messages to send PowerShell Remoting Protocol data to a PowerShell server's RunspacePool or pipeline. The PowerShell client MUST follow the rules described in section 3.1.5.1.1 while sending messages.

2. The PowerShell client receives data from the server as part of wxf:ReceiveResponse (section 3.2.5.3.8) message and constructs a PowerShell message as per the rules described in section 3.1.5.2. The PowerShell client decides whether a PowerShell message is targeted to a RunspacePool or pipeline as per the rules described in section 3.1.5.4 and 2.2.1.

3. Some messages apply only to RunspacePools, and are valid only when the RunspacePool is in certain states. The valid states for each message are listed in section 3.1.5.4. When a PowerShell client receives a message for a RunspacePool that is not in the correct state, the client MUST stop any executing pipelines (section 3.1.1.3.2) and close that RunspacePool (section 3.1.1.2.2).

4. Some messages apply to pipelines, and are valid only when the pipeline is in certain states. The valid states for each message are listed in section 3.1.5.4. When a PowerShell client receives a message for a pipeline that is not in the correct state, then the client MUST stop the pipelines (section 3.1.1.3.2).

5. When a PowerShell client's RunspacePool state reaches Closed or Broken state, the client MUST NOT process any message targeted for that particular RunspacePool and MUST NOT send any messages to the PowerShell server's RunspacePool, except for wxf:Delete message (section 3.1.5.3.11). If the PowerShell client receives any message from the server targeted to the RunspacePool in this state, then the PowerShell client MUST ignore that message.

6. When a PowerShell client's pipeline state reaches Completed or Stopped or Failed state, the PowerShell client MUST not process any message targeted for that particular pipeline and MUST not send any messages to the PowerShell server's pipeline, except for wxf:Signal message (section 3.1.5.3.9). If the client receives any message from the server targeted to the pipeline in this state, then the PowerShell client MUST ignore that message.

3.1.5.1.1 Rules for Sending Data

1. The PowerShell client MUST use one of wxf:Create, wxf:Command, or wxf:Send messages (as specified in [MS-WSMV]) to send PowerShell messages to the PowerShell server, depending on the circumstances. See section 3.1.5.3 for details.

2. When sending any PowerShell message (section 2.2), the message MUST first be fragmented into one or more fragments. See section 2.2.4 for the format of a fragment. The FragmentIDs MUST be numbered consecutively beginning with 0.

3. The fragments MUST be sent in ascending order of FragmentID, using either wxf:Create (section 3.1.5.3.3), wxf:Send (section 3.1.5.3.5) or wxf:Command (section 3.1.5.3.3).

4. If multiple fragments can fit into a single WS-MAN message, then the single WS-MAN message SHOULD include as many fragments as possible (see [MS-WSMV], section 3.1.4.1.7). The fragments MUST be embedded in the order that the PowerShell messages were generated.

5. When sending fragments using wxf:Create or wxf:Command, the fragments MUST be base64 encoded, as specified in sections 3.1.5.3.3 and 3.1.5.3.3.

6. When sending fragments using wxf:Send, the fragments MUST be sent with the Stream element (as specified in [MS-WSMV], section 2.2.4.40) set to either "stdin" or "pr". Fragments from RUNSPACEPOOL_HOST_RESPONSE and PIPELINE_HOST_RESPONSE messages (sections 3.1.5.4.16 and 3.1.5.4.27) SHOULD be sent using a "pr" stream. There can be multiple Stream elements in a Send Complex Type (as specified in [MS-WSMV], section 2.2.4.32). Multiple fragments can be concatenated and sent in a single Stream element. An individual fragment cannot be broken down and cannot span multiple Stream elements. The PowerShell Remoting Protocol does not encode fragments sent using wxf:Send messages, instead relying on the encoding being done by Web Services Management Protocol Extensions for Windows Vista (see [MS-WSMV], section 2.2.4.40 for allowed encodings).

3.1.5.1.2 Rules for Receiving Data

1. The PowerShell client receives data from the PowerShell server using the wxf:ReceiveResponse WS-MAN message. Each wxf:ReceiveResponse message contains one or more fragments. See section 2.2.4 for the format of a fragment.

2. When one of the WS-MAN messages with fragmented data is received, the PowerShell client extracts the **Blob** field of the fragment and appends the extracted data to the **PartiallyDefragmentedPsrpMessage** field of the targeted RunspacePool (section 3.1.1.2.3) or pipeline (section 3.1.1.3.3).

3. After an End Fragment packet is received (section 2.2.4), a whole PSRP Message (see section 2.2.1) is stored in **PartiallyDefragmentedPsrpMessage** and can be handled as described in section 3.1.5.4.

4. PowerShell clients SHOULD compare the **ObjectId** and **FragmentId** fields of each received fragment with the **LastObjectId** and **LastFragmentId** data stored in the ADM and then update the ADM. If at any point, it is determined that the fragments are not received in ascending order of **FragmentID** with the same **ObjectID**, the PowerShell client MUST close the appropriate RunspacePool (section 3.1.4.2) or stop the appropriate pipeline (section 3.1.4.4).

3.1.5.2 Sequencing Rules

The following is a typical sequence for creating a RunspacePool and executing a pipeline on a PowerShell server.

1. The PowerShell client MUST construct a RunspacePool and the RunspacePool MUST be in Opened state. Refer to section 3.1.4.1 for more details.

2. When a RunspacePool is in the Opened state, RunspacePool specific messages--such as Set Maximum Runspaces (section 3.1.5.4.6), Set Minimum Runspaces (section 3.1.5.4.7), and Get Available Runspaces (section 3.1.5.4.11-- can be sent to PowerShell server's RunspacePool. For more details about the exact messages that can be sent, see section 3.1.5.4.

3. When a RunspacePool is in the Opened state, the PowerShell client MAY send a pipeline message (section 3.1.5.4.10) to the PowerShell server to start executing a pipeline on the server. Refer to section 3.1.4.3 for more details about the PowerShell pipeline sequence.

4. When the RunspacePool is in Opened state, the PowerShell client MAY receive RunspacePool specific messages, such as the RUNSPACEPOOL_HOST_CALL message (section 3.1.5.4.15) and RUNSPACEPOOL_STATE message (section 3.1.5.4.9).

5. When a pipeline is in Running state and a success response message for wxf:Command is received (section 3.1.5.3.4), the PowerShell client MAY receive pipeline specific messages, such as the PIPELINE_OUTPUT message (section 3.1.5.4.19) and PIPELINE_HOST_CALL message (section 3.1.5.4.26). For more details about the exact messages that can be received, see section 3.1.5.4.

6. The PowerShell client MAY choose to stop a pipeline at any time using the wxf:Signal message (section 3.1.5.3.9), as long as the pipeline is in a Running state and a success response message for wxf:Command is received (section 3.1.5.3.4).

7. A PowerShell client MAY choose to close a RunspacePool and associated pipelines at any time, as long as the RunspacePool is in an Opened state.

8. When a RunspacePool is in a Closed state, that specific RunspacePool is not allowed for executing pipelines.

3.1.5.3 Rules for Processing WS-MAN Messages

3.1.5.3.1 Rules for the wxf:Create Message

The PowerShell client uses a wxf:Create message (as described in [MS-WSMV] section 3.1.4.5.2) to create a RunspacePool on the PowerShell server. Before sending this message, the PowerShell client creates a RunspacePool instance, assigns it a GUID (section 2.2.5.1.18), and initializes its state to Opening (section 3.1.1.2.2). The following information is supplied for the wxf:Create message.

Element	Value
Uri	Network URI to which to connect.
ResourceURI	Any string, per the rules specified in [MS-WSMV] section 3.1.4.5.2.<1>
OptionSet	An option set with the following options. Name = ProtocolVersion, MustComply=True, Value=2.1 or 2.2

The following information is supplied for the shell data type, as required by [MS-WSMV] section 2.2.4.37, in the wxf:Create message.

Element	Value
ShellId	Valid PowerShell remoting connection string of the form.Proto://computername:port/applicationname Where "proto" can be "http" or "https", "computername" is the name of the machine to which to connect, "port" is the port for connection, and "applicationname" can be WSMAN or any other application that supports the Web Services Management Protocol Extensions for Windows Vista [MS-WSMV].<2>
IdleTimeout	The client can specify any integer value. <3>
InputStreams	"stdin pr". "stdin" is used to send regular data. "pr" is used to send host response data (see sections 2.2.2.27 and 2.2.2.16).
OutputStreams	stdout
WorkingDirectory	Unused by the PowerShell Remoting Protocol.

Element	Value
Lifetime	Unused by the PowerShell Remoting Protocol.
Environment	Unused by the PowerShell Remoting Protocol.
<open content>	<creationXml> is described in the following section.

The generic description for <open content> is defined in [MS-WSMV] section 2.2.4.37.

The PowerShell client uses <open content> to send additional data, called creationXml data, that assists in creating a shell on the server. This creationXml can contain any data that is destined to the shell. Without this creationXml data, clients MUST use wxf:Send messages, described in section 3.1.5.3.5. To avoid multiple network calls, it is encouraged to send additionally using "creationXml". A SESSION_CAPABILITY message (section 2.2.2.1) MUST be the first message that is sent to a server from the client. Typically the SESSION_CAPABILITY message is broken down to only one fragment (see section 2.2.4), as is the INIT_RUNSPACEPOOL message, and both those messages are included in the creationXml. The creationXml MUST be of the following format.

```
<creationXml xmlns=http://schemas.microsoft.com/powershell>
 Base64-Encoded data
</creationXml>
```

As described in the preceding section, all the data that is sent as part of creationXml MUST be base64-encoded as described in [RFC3548].

If the wxf:Create message is successfully received and processed by the server, the server MUST send either a success or a failure message. In either case a response is sent from the server. A wxf:ResourceCreated message, described in [MS-WSMV] section 3.1.4.5.2, is sent to notify success. A wxf:Fault message, described in [MS-WSMV] section 2.2.4.43, is sent to notify failure.

The SESSION_CAPABILITY message (section 2.2.2.1) and INIT_RUNSPACEPOOL message (section 2.2.2.2) SHOULD be sent using wxf:Create message. The PowerShell client MUST NOT send any other PowerShell messages using a wxf:Create message.

3.1.5.3.2 Rules for the wxf:ResourceCreated Message

The PowerShell server sends a wxf:ResourceCreated message ([MS-WSMV], section 3.1.4.5.2) upon successful processing of a wxf:Create message (section 3.1.5.3.1).

The wsa:EndpointReference message encapsulated within the wxf:ResourceCreated message contains a reference to the newly created [MS-WSMV] Shell instance on the PowerShell server. The PowerShell client stores this wsa:EndPointReference for future use (section 3.1.1.2.4). The PowerShell client MUST use this address in all subsequent [MS-WSMV] messages to the shell instance, that is, wxf:Delete (section 3.1.5.3.11), wxf:Command (section 3.1.5.3.3), wxf:Signal (section 3.1.5.3.9), wxf:Send (section 3.1.5.3.5), and wxf:Receive (section 3.1.5.3.7).

The PowerShell client stores the value specified in the ShellID element of the wxf:ResourceCreated for future use (sections 3.1.1.1.1 and 3.1.1.2.4). The PowerShell client MUST use this ShellID in all subsequent [MS-WSMV] messages to the shell instance, that is, wxf:Delete (section 3.1.5.3.11), wxf:Command (section 3.1.5.3.3), wxf:Signal (section 3.1.5.3.9), wxf:Send (section 3.1.5.3.5), and wxf:Receive (section 3.1.5.3.7).

3.1.5.3.3 Rules for the wxf:Command Message

The PowerShell remoting protocol executes a pipeline on the remote RunspacePool (created using a remote shell as described in 3.1.5.3.1) by sending a wxf:Command message to the remote shell, as specified in [MS-WSMV] section 3.1.4.11. The header of the wxf:Command message MUST contain the following information.

Element	Value
ResourceURI	Any string, per the rules specified in [MS-WSMV] section 3.1.4.5.2. <4>
ShellID selector	The ShellID returned in the wxf:ResourceCreated message (see section 3.1.5.3.2).

The body of the message MUST contain a command line complex type as described in [MS-WSMV] section 2.2.4.7. The following information is supplied for the required values in the command line complex type.

Element	Value
Command	MUST be empty.
Arguments	The first fragment of the serialized pipeline. This first fragment MUST be base64-encoded before including the data in the Arguments element. The remaining fragments MUST be sent using the Send message to the command as described in [MS-WSMV] section 3.1.4.13.

If the wxf:Command message is successfully received and processed by the server, the server MUST send either a success or a failure message. In either case, a response is sent from the server. A wxf:CommandResponse message, described in [MS-WSMV] section 2.2.4.8, is sent to notify success. A wxf:Fault message, specified in [MS-WSMV] section 2.2.4.43, is sent to notify failure.

The PowerShell messages CREATE_PIPELINE (section 3.1.5.4.10) and GET_COMMAND_METADATA (section 2.2.2.14) MAY be sent using a wxf:Command message. The PowerShell client MUST NOT send any other PowerShell messages using a wxf:Command message.

3.1.5.3.4 Rules for the wxf:CommandResponse Message

The PowerShell server sends a wxf:CommandResponse message ([MS-WSMV] section 3.1.4.11) upon successful processing of wxf:Command (section 3.1.5.3.3). The PowerShell client stores the value specified in the CommandId element of the wxf:CommandResponse message for future reference (see section 3.1.1.3.4). The PowerShell client MUST use this CommandId for future communication with the pipeline, that is, wxf:Signal (section 3.1.5.3.9), wxf:Send (section 3.1.5.3.5), and wxf:Receive (section 3.1.5.3.7).

3.1.5.3.5 Rules for the wxf:Send Message

The wxf:Send message (as specified in [MS-WSMV] section 3.1.4.13) is used to send input to a pipeline or a RunspacePool. The following information is included in the message.

Element	Value
ResourceURI	The Resource URI of the RunspacePool to which this send message is targeted. For more information see section 3.1.5.3.3.
ShellID selector	The ShellID returned in the wxf:ResourceCreated message (see section 3.1.5.3.2).

The body of the send message MUST contain a send data type as described in [MS-WSMV] section 2.2.4.32. The data type MUST contain the following information.

Element	Value
Stream	Stdin - if messages are to be sent in the regular priority order. Pr - to send a PIPELINE_HOST_RESPONSE message (see sections 2.2.2.27 and RUNSPACEPOOL_HOST_RESPONSE message 2.2.2.16). The Name attribute of the stream element MUST be accordingly stdin or pr.

A wxf:Send message can be sent to a RunspacePool or pipeline. If the wxf:Send message is targeted to a pipeline it MUST contain the following attribute:

Element	Attribute	Value
Stream	CommandId	The CommandId returned in the wxf:CommandResponse message (see section 3.1.5.3.4). This attribute MUST NOT be specified if the wxf:Send message is targeted to a RunspacePool.

If the wxf:Send message is successfully received and processed by the server, the server MUST send either a success or a failure message. In either case a response is sent from the server. A wxf:SendResponse message, described in [MS-WSMV] section 2.2.4.33, is sent to notify success. A wxf:Fault message, described in [MS-WSMV] section 2.2.4.43, is sent to notify failure.

For any given RunspacePool or pipeline, there can only be one outstanding wxf:Send message targeted to that RunspacePool or pipeline. The PowerShell client MUST wait until the PowerShell server replies to the wxf:Send message with a wxf:SendResponse message or a wxf:Fault message before sending another wxf:Send message targeted to the same RunspacePool or pipeline.

Only the following PowerShell messages are allowed to be sent to the server using the wxf:Send message: SESSION_CAPABILITY (section 2.2.2.1), INIT_RUNSPACEPOOL (section 2.2.2.2), PUBLIC_KEY (section 3.1.5.4.3), SET_MAX_RUNSPACES (section 3.1.5.4.6), SET_MIN_RUNSPACES (section 3.1.5.4.7), CREATE_PIPELINE (section 3.1.5.4.10), GET_AVAILABLE_RUNSPACES (section 3.1.5.4.11), RUNSPACEPOOL_HOST_RESPONSE (section 3.1.5.4.16), PIPELINE_INPUT (section 3.1.5.4.17), END_OF_PIPELINE_INPUT (section 3.1.5.4.18), and PIPELINE_HOST_RESPONSE (section 3.1.5.4.27).

3.1.5.3.6 Rules for the wxf:SendResponse Message

The PowerShell client waits for a wxf:SendResponse message (see [MS-WSMV] section 3.1.4.13) to verify that the PowerShell server successfully processed the wxf:Send message (see section 3.1.5.3.5).

3.1.5.3.7 Rules for the wxf:Receive Message

The wxf:Receive message (as specified in [MS-WSMV] section 3.1.4.14) is used to notify a PowerShell server's RunspacePool or pipeline to send PowerShell messages to the client using the wxf:ReceiveResponse message (section 3.2.5.3.8). The following information is included in the message.

Element	Value
ResourceURI	The Resource URI of the RunspacePool to which this receive message is targeted. For

Element	Value
	more information, see section 3.1.5.3.3.
ShellID selector	The ShellID returned in the wxf:ResourceCreated message (section 3.1.5.3.2).

The body of the receive message MUST contain receive complex data type, as described in [MS-WSMV] section 2.2.4.26. The received complex data type MUST contain the following information.

Element	Value
DesiredStream	MUST contain a value of "stdout".

A wxf:Receive message can be sent to a RunspacePool or pipeline. If the wxf:Receive message is targeted to a pipeline it MUST contain the following attribute in the received complex data type.

Element	Attribute	Value
DesiredStream	CommandId	The CommandId returned in the wxf:CommandResponse message (see section 3.1.5.3.4). This attribute MUST NOT be specified if the wxf:Receive message is targeted to RunspacePool.

If the wxf:Receive message is successfully received and processed by the server, the server MUST send either a success or a failure message. In either case, a response is sent from the server. A wxf:ReceiveResponse message, described in [MS-WSMV] section 2.2.4.27, is sent to notify success. A wxf:Fault message, described in [MS-WSMV] section 2.2.4.43, is sent to notify failure.

Note that no PowerShell messages are sent using wxf:Receive.

3.1.5.3.8 Rules for the wxf:ReceiveResponse Message

The PowerShell server sends a wxf:ReceiveResponse message ([MS-WSMV] section 3.1.4.14) upon successful processing of wxf:Receive (section 3.1.5.3.7).

The wxf:ReceiveResponse message may contain data. The following table describes how to interpret this wxf:ReceiveResponse message.

Element	Attribute	Value
Stream	Name	This attribute MUST be stdout, as the PowerShell server can send data only in one stream. If another stream name is specified, then the message MUST be discarded.
Stream	CommandId	This attribute is present if the wxf:ReceiveResponse is meant for a pipeline in which case the value of the attribute identifies the pipeline to which this wxf:ReceiveResponse is targeted. If CommandId is not specified, then the wxf:ReceiveResponse is targeted to a RunspacePool.
CommandState	CommandId	The CommandState element is present if the wxf:ReceiveResponse is meant for a pipeline or a RunspacePool. The value of the CommandId attribute, if present, identifies the pipeline this wxf:ReceiveResponse is targeted to. This attribute MUST NOT be specified if the wxf:ReceiveResponse message is targeted to RunspacePool.

Element	Attribute	Value
		The CommandState may not be present in every wxf:ReceiveResponse message. When present, the value of the State attribute identifies the Command State.
CommandState	State	The CommandState element is present if the wxf:ReceiveResponse is meant for a pipeline or a RunspacePool. The value of this attribute identifies the state of the pipeline or a RunspacePool. A value of "http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandState/Done" specifies that this wxf:ReceiveResponse message is the last wxf:ReceiveResponse message from the server for that particular pipeline (as identified by CommandId) or for that particular RunspacePool (as identified by ShellId selector).

Finally the Stream element holds whatever data that the server sent. This data MUST first be interpreted as specified in [MS-WSMV] sections 2.2.4.27 and 3.1.4.1.31. When the data is interpreted this way, the converted data MUST be interpreted as described in section 3.1.5.1.2.

Upon receiving the wxf:ReceiveResponse message, the PowerShell client attempts to get the RunspacePool instance or pipeline instance, using the ShellID selector and the CommandId attribute specified in the wxf:ReceiveResponse message (section 3.2.5.3.8) and the RunspacePool and the pipeline tables (section 3.2.1.1). If a corresponding RunspacePool or pipeline instance is not found, then the PowerShell client ignores the message.

If a corresponding RunspacePool or pipeline instance is found, then PowerShell client extracts the data from wxf:ReceiveResponse message and processes the data as per the rules described in section 3.1.5.1.

3.1.5.3.9 Rules for the wxf:Signal Message

A wxf:Signal message can be sent either to a RunspacePool or pipeline (as specified in [MS-WSMV] section 3.1.4.12). The PowerShell client uses a signal to stop an executing pipeline on the server. The following information MUST be supplied to the message.

Element	Value
ResourceURI	The Resource URI of the RunspacePool to which this wxf:Signal message is targeted. For more information, see section 3.1.5.3.3.
ShellId	The ShellID returned in the wxf:ResourceCreated message (see section 3.1.5.3.2).

The message requires a signal complex data type in the body of the message as defined in [MS-WSMV] section 2.2.4.38. The following information is sent in the signal complex data type.

Element	Value
Code	The value MUST be "powershell/signal/crtl_c".

A wxf:Signal message can be sent to a RunspacePool or pipeline. If the wxf:Signal message is targeted to a pipeline it MUST contain the following attribute in the Signal complex data type.

Element	Attribute	Value
Signal	CommandId	The CommandId returned in the wxf:CommandResponse message (see section

Element	Attribute	Value
		3.1.5.3.4). This attribute MUST NOT be specified if the wxf:Signal message is targeted to a RunspacePool.

If the wxf:Signal message is successfully received and processed by the server, the server MUST send either a success or a failure message. In either case a response is sent from the server. A wxf:SignalResponse message, described in [MS-WSMV] section 3.1.4.12, is sent to notify success. A wxf:Fault message, described in [MS-WSMV] section 2.2.4.43, is sent to notify failure.

Note that the PowerShell remoting protocol never sends a wxf:Signal message to a RunspacePool, although the underlying [MS-WSMV] protocol supports it. No PowerShell messages are sent using wxf:Signal.

3.1.5.3.10 Rules for the wxf:SignalResponse Message

The PowerShell client waits for a wxf:SignalResponse message (see [MS-WSMV] section 3.1.4.12) to verify that the PowerShell server successfully processed the wxf:Signal message (see section 3.1.5.3.10); otherwise, it discards the data from the wxf:SignalResponse message.

3.1.5.3.11 Rules for the wxf:Delete Message

To close a RunspacePool on the PowerShell server, the PowerShell client MUST initiate the close by sending a wxf:Delete message (as specified in [MS-WSMV] section 3.1.4.4.1). This message can be sent asynchronously to any outstanding messages on the RunspacePool, and therefore the RunspacePool will be forcibly closed. The following information is supplied in the delete message.

Element	Value
Uri	Network Uri to connect to.
ResourceURI	The Resource URI of the RunspacePool to which this wxf:Delete message is targeted. For more information, see section 3.1.5.3.3.
ShellId	The ShellID returned in the wxf:ResourceCreated message (see section 3.1.5.3.2).

If the wxf:Delete message is successfully received and processed by the server, the server MUST send either a success or a failure message. In either case, a response is sent from the server. A wxf:DeleteResponse message, described in [MS-WSMV] section 3.1.4.4.1, is sent to notify success. A wxf:Fault message, described in [MS-WSMV] section 2.2.4.43, is sent to notify failure.

Note that no PowerShell messages are sent using wxf:Delete.

3.1.5.3.12 Rules for the wxf:DeleteResponse Message

The PowerShell client waits for a wxf:DeleteResponse (see [MS-WSMV] section 3.1.4.4.1) message to verify that the PowerShell server successfully processed the wxf:Delete (see section 3.1.5.3.11) message; otherwise, it discards the data from the wxf:DeleteResponse message.

3.1.5.3.13 Rules for the wxf:Fault Message

If the PowerShell client receives a wxf:Fault message (as specified in [MS-WSMV] section 2.2.4.43) targeted to a RunspacePool, then the PowerShell client MUST change the RunspacePool state to

Broken, stop any executing pipelines (section 3.1.4.3) using the pipeline table (section 3.1.1.2.6), and send a wxf:Delete message (section 3.1.5.3.11).

If the PowerShell client receives a wxf:Fault message ([MS-WSMV] section 2.2.4.43) in response to a message targeted to a pipeline, then the PowerShell client MUST change the pipeline state to Failed and send a wxf:Signal message (section 3.1.5.3.9).

3.1.5.3.14 Rules for the wxf:Connect Message

The PowerShell client uses a wxf:Connect message (as specified in [MS-WSMV] section 3.1.4.17) to connect to a RunspacePool on the PowerShell server. Before sending this message, the PowerShell client discovers the RunspacePool identifiers available on the server by sending a wxf:Enumerate message (as specified in [MS-WSMV] section 3.1.4.8). The PowerShell client then creates its own RunspacePool instance, assigns it an identifier from the list of available RunspacePool identifiers, and initializes its state to Connecting (section 3.1.1.2.2). The client supplies the following information for the wxf:Connect message.

Element	Value
Uri	The network URI to connect to.
ResourceURI	Any string adhering to the rules specified in [MS-WSMV] section 3.1.4.5.2.<5>
OptionSet	An option set with the following options: Name = ProtocolVersion, MustComply=True, Value=2.1 or 2.2

Clients supply the following information in the wxf:Connect message for the shell data type, as specified in [MS-WSMV] section 2.2.4.12:

Element	Value
Resource	A valid PowerShell remoting connection string of the form "protocol://computername:port/applicationname", where protocol can be one of "http" or "https", computername is the name of the machine to connect to, port is the port number for the connection, and applicationname can be "WSMan" or any other application that supports the Web Services Management Protocol Extensions for Windows Vista.<6>
ShellID	The identifier of the RunspacePool the client intends to connect to.

<open content>	Additional data that assists in connecting to a shell on the server, if any, MUST be Base64 encoded (as specified in [RFC3548]) and packaged in a <connectXml> element with the following format:
	
	`<connectXml xmlns=http://schemas.microsoft.com/powershell>`
	`Base64-Encoded data`
	`</connectXml>`
	
	The <open content> MAY also contain any data that is useful to the shell.

The body of the message MAY contain the following optional element:

Element	Value
BufferMode	Either of the values "Drop" or "Block" to indicate the buffering mode the server will use when the shell is later disconnected.

If the wxf:Connect message is successfully received and processed by the server, the server MUST send either a success or a failure message. The server sends a wxf:ConnectResponse message, described in [MS-WSMV] section 3.1.4.17, to indicate success. The server sends a wxf:Fault message, described in [MS-WSMV] section 2.2.4.43, to indicate failure.

The PowerShell client MUST use a wxf:Connect message to send SESSION_CAPABILITY (section 2.2.2.1) and CONNECT_RUNSPACEPOOL (section 2.2.2.28) message data to the server. The PowerShell client MUST NOT send any other PowerShell message data using a wxf:Connect message.

The PowerShell client also uses the wxf:Connect message to connect to a specific pipeline associated with a RunspacePool. Once the RunspacePool is connected using the wxf:Connect Message, the PowerShell client MUST issue a separate wxf:Connect message to connect to a specific pipeline. The following additional information MUST be added to the body of the second wxf:Connect message:

Element	Value
CommandId	The identifier of the pipeline that the PowerShell client intends to connect to.

Once connected, the PowerShell client can use the wxf:Send and wxf:Receive messages to send input to and receive output from a pipeline.

3.1.5.3.15 Rules for the wxf:ConnectResponse Message

The PowerShell server sends a wxf:ConnectResponse message upon successful processing of a wxf:Connect message, as specified in [MS-WSMV] section 3.1.4.17.

The PowerShell server sends back its SESSION_CAPABILITY message as part of the wxf:ConnectResponse message. The PowerShell client should terminate the connection process and set the state of the RunspacePool to Broken if a SESSION_CAPABILITY message is not received as part of the wxf:ConnectResponse message.

3.1.5.3.16 Rules for the wxf:Disconnect Message

The PowerShell Remoting Protocol disconnects a remote RunspacePool, created using a remote shell as specified in section 3.1.5.3.1, by sending a wxf:Disconnect message to the remote shell as specified in [MS-WSMV] section 3.1.4.11. Once the server shell is disconnected, all command input and output streams associated with the shell are automatically suspended. The header of the wxf:Disconnect message MUST contain the following information.

Element	Value
ResourceURI	Any string that satisfies the rules specified in [MS-WSMV] section 3.1.4.5.2.<7>
ShellID selector	The ShellID returned in the wxf:ResourceCreated message (see section 3.1.5.3.2).

The body of the message MAY contain the following optional elements.

Element	Value
IdleTimeout	Any integer value, which will override the idle timeout value specified in the prior wxf:Create message.<8>
BufferMode	Either of the strings "Drop" or "Block", which indicate the buffering mode the server will use when the shell is disconnected.

If the wxf:Disconnect message is successfully received and processed by the server, the server MUST send either a success or a failure message. A wxf:DisconnectResponse message, as specified in [MS-WSMV] section 3.1.4.15, indicates success. A wxf:Fault message, specified in [MS-WSMV] section 2.2.4.43, indicates failure.

3.1.5.3.17 Rules for the wxf:DisconnectResponse Message

The PowerShell client waits for a wxf:DisconnectResponse message (see [MS-WSMV] section 3.1.4.15) to verify that the PowerShell server successfully processed the wxf:Disconnect message (see section 3.1.5.3.16).

3.1.5.3.18 Rules for the wxf:Reconnect Message

The PowerShell Remoting Protocol reconnects to a RunspacePool that has been previously disconnected by a wxf:Disconnect message (section 3.1.5.3.16), by sending a wxf:Reconnect message to the remote shell as specified in [MS-WSMV] section 3.1.4.16. The header of the wxf:Reconnect message MUST contain the following information.

Element	Value
ResourceURI	Any string that satisfies the rules specified in [MS-WSMV] section 3.1.4.5.2.<9>
ShellID selector	The ShellID returned in the wxf:ResourceCreated message (see section 3.1.5.3.2).

The body of the message MAY contain the following optional elements.

Element	Value
BufferMode	Either of the strings "Drop" or "Block", which indicate the buffering mode the server will use when the shell is later disconnected.

If the wxf:Reconnect message is successfully received and processed by the server, the server MUST send either a success or a failure message. A wxf:ReconnectResponse message, as specified in [MS-WSMV] section 3.1.4.16, indicates success. A wxf:Fault message, specified in [MS-WSMV] section 2.2.4.43, indicates failure.

A wxf:Reconnect message sent to a shell does not automatically reconnect any commands associated with it. A second wxf:Reconnect message with the following additional element in its body MUST be sent to reconnect to a particular command.

Element	Value
CommandID	The identifier of the command, returned as part of a wxf:CommandResponse message (section 3.1.5.3.4).

3.1.5.3.19 Rules for the wxf:ReconnectResponse Message

The PowerShell client waits for a wxf:ReconnectResponse message (see [MS-WSMV] section 3.1.4.16) to verify that the PowerShell server successfully processed the wxf:Reconnect message (see section 3.1.5.3.18).

3.1.5.4 Rules for Processing PowerShell Messages

See the general protocol rules described in section 3.1.5.1. The following sections describe the impact of various PowerShell Remoting Protocol messages (section 2.2) on a PowerShell client.

3.1.5.4.1 SESSION_CAPABILITY Message

The syntax of this message is specified in section 2.2.2.1.

3.1.5.4.1.1 Sending to the Server

The RunspacePool MUST be in an Opening state (section 3.1.1.2.2) when this message is sent.

The SESSION_CAPABILITY message MUST be the first message sent to the server. Fragments (see section 2.2.4) of this message can be sent either as part of the <creationXml> element discussed in section 3.1.5.3.1 or as part of input discussed in section 3.1.5.3.5. This message MUST be sent only once per RunspacePool from a PowerShell client client to a PowerShell server.

The SESSION_CAPABILITY message MUST have the following properties when it is sent to the server.

Name	Value to Send
protocolversion	MUST be 2.1 or 2.2.
PSVersion	MUST be 2.0
SerializationVersion	MUST be 1.1.0.1
TimeZone	SHOULD be the client's time zone.

When this message is sent, the PowerShell client changes the RunspacePool state to NegotiationSent (section 3.1.1.2.2).

3.1.5.4.1.2 Receiving from the Server

The PowerShell client MUST receive this message once per the RunspacePool from the server. The RunspacePool MUST be in NegotiationSent state when this message is received. The PowerShell client processes the message and validates the actual data received from the server with the expected data given in the following table.

Name	Expected value
protocolversion	2.1 or 2.2.
PSVersion	2.0
SerializationVersion	1.1.0.1

If expected versions are received from the server, the PowerShell client MUST change the RunspacePool state to NegotiationSucceeded (section 3.1.1.2.2). If server protocolversion is 2.1 or 2.2, then the client SHOULD also change the RunspacePool state to NegotiationSucceeded, but the client MAY also change the state to Broken in this situation. In all other cases, the client MUST change the RunspacePool state to Broken.

3.1.5.4.2 INIT_RUNSPACEPOOL Message

The syntax of this message is specified in section 2.2.2.2.

The RunspacePool MUST be in an Opening or NegotiationSucceeded state (section 3.1.1.2.2) when this message is sent.

This message MUST be sent only once per RunspacePool from a PowerShell client to a PowerShell server.

3.1.5.4.3 PUBLIC_KEY Message

The syntax of this message is specified in section 2.2.2.3. The message's public key, exponent, and modulus fields MUST be from the client's **Public Key Pair** (see section 3.1.1.1.3).

The RunspacePool MUST be in Opened state (section 3.1.1.2.2) when this message is sent. This message MUST be sent from a PowerShell client to a PowerShell server 1) in response to a public key request received from the server (see section 3.1.5.4.5), and 2) when the higher layer requests a Session Key exchange prior to sending secure strings from the client to the server (see section 3.1.4.8).

This message MUST be sent only once from a PowerShell client to a PowerShell server for one RunspacePool.

The Session Key Transfer timer (section 3.1.1.2.8) MUST be started by the PowerShell Remoting Protocol when it sends a PUBLIC_KEY message. There MUST be a unique timer for each PUBLIC_KEY message. Upon receipt of an ENCRYPTED_SESSION_KEY message (section 2.2.2.4) for that PUBLIC_KEY message, the timer MUST be canceled.

The Session Key Transfer timer MUST expire after the number of milliseconds given by the SessionKeyTransferTimeoutms (section 3.1.1.2.8). Upon expiration of this timer, the PowerShell Remoting Protocol MUST close the associated RunspacePool as described in section 3.1.4.1.

3.1.5.4.4 ENCRYPTED_SESSION_KEY Message

The syntax of this message is specified in section 2.2.2.4.

This message is targeted to the RunspacePool. When this message is received, the PowerShell client extracts the session key (section 2.2.2.4) from the message, decrypts it using the global private key (see section 3.1.1.1.3), and stores it in the RunspacePool's Session Key structure (section 3.1.1.2.7).

When this message is received, the RunspacePool MUST be in the Opened state.

3.1.5.4.5 PUBLIC_KEY_REQUEST Message

The syntax of this message is specified in section 2.2.2.5.

This is message is targeted to RunspacePool. A PowerShell server sends this message to get a PowerShell client's Public Key. After receiving this message, the client MUST send a PUBLIC_KEY message (section 3.1.5.4.3) to the server.

When this message is received, RunspacePool MUST be in Opened state.

3.1.5.4.6 SET_MAX_RUNSPACES Message

The syntax of this message is specified in section 2.2.2.6.

The RunspacePool MUST be in Opened state (section 3.1.1.2.2) when this message is sent.

This message MUST be sent to the PowerShell server's RunspacePool. Before sending this message, the PowerShell client MUST construct a unique integer identifier to represent the message and store it in the RunspacePool's CI table (section 3.1.1.2.5). In response to this message, the PowerShell server will send a RUNSPACE_AVAILABILITY message (section 2.2.2.8), which the PowerShell client will use to update the RunspacePool's CI table (section 3.1.1.2.5) by removing the appropriate integer identifier.

3.1.5.4.7 SET_MIN_RUNSPACES Message

The syntax of this message is specified in section 2.2.2.7.

The RunspacePool MUST be in Opened state (section 3.1.1.2.2) when this message is sent.

This message MUST be sent to the PowerShell server's RunspacePool. Before sending this message, the PowerShell client MUST construct a unique integer identifier to represent the message and store it in the RunspacePool's CI table (section 3.1.1.2.5). In response to this message, the PowerShell server will send a RUNSPACE_AVAILABILITY (section 2.2.2.8), which the PowerShell client will use to update the RunspacePool CI table (section 3.1.1.2.5) by removing the appropriate integer identifier.

3.1.5.4.8 RUNSPACE_AVAILABILITY Message

The syntax of this message is specified in section 2.2.2.8.

This message is targeted to the RunspacePool. The PowerShell server sends this message as a response to SET_MAX_RUNSPACES message (section 2.2.2.6), SET_MIN_RUNSPACES message (section 2.2.2.7), or GET_AVAILABLE_RUNSPACES message (section 2.2.2.11).

When this message is received, the PowerShell client extracts the integer identifier from the message and updates the RunspacePool's CI table (section 3.1.1.2.5) by removing the appropriate integer identifier.

When this message is received, the RunspacePool MUST be in an Opened state.

3.1.5.4.9 RUNSPACEPOOL_STATE Message

The syntax of this message is specified in section 2.2.2.9.

This is message is targeted to RunspacePool. When this message is received, the PowerShell client extracts the state information (section 2.2.3.4) from the message and updates the RunspacePool state (section 3.1.1.2.2) accordingly.

This message can be received at any time as long as the RunspacePool is not in Closed or Broken state. If this message is received when the RunspacePool is in Closed or Broken state, then this message is ignored by the PowerShell client.

3.1.5.4.10 CREATE_PIPELINE Message

The syntax of this message is specified in section 2.2.2.10.

This message MAY be sent from a PowerShell client to a PowerShell server when the RunspacePool state (section 3.1.1.2.2) is Opened. The PowerShell client sends this message to execute a pipeline on the PowerShell server.

PowerShell client constructs a GUID to represent the pipeline, initializes the pipeline state (section 3.1.1.3.2) to Running, constructs the message (section 2.2.2.10), and sends it to the server.

For more details about how a PowerShell client executes a pipeline on a PowerShell server refer to section 3.1.4.2.

3.1.5.4.11 GET_AVAILABLE_RUNSPACES Message

The syntax of this message is specified in section 2.2.2.11.

The RunspacePool MUST be in an Opened state (section 3.2.1.2.2) when this message is sent.

This message MUST be sent to the PowerShell server's RunspacePool. Before sending this message, the PowerShell client MUST construct a unique integer identifier to represent the message and store it in the RunspacePool's CI table (section 3.1.1.2.5). In response to this message, the PowerShell server will send a RUNSPACE_AVAILABILITY (section 2.2.2.8) which the PowerShell client will use to update RunspacePool CI table (section 3.1.1.2.5) by removing the appropriate integer identifier.

3.1.5.4.12 USER_EVENT Message

The syntax of this message is specified in section 2.2.2.12.

This message is targeted to a PowerShell client's RunspacePool. The PowerShell server sends this message to notify a PowerShell client about a server-side event. Note that the PowerShell Remoting Protocol does not generate or interpret any events; it merely provides a mechanism for higher layers on the PowerShell client to be notified when new events are reported by the PowerShell server.

The PowerShell client's RunspacePool MUST be in an Opened state while processing this message.

3.1.5.4.13 APPLICATION_PRIVATE_DATA Message

The syntax of this message is specified in section 2.2.2.13.

This message is targeted to a PowerShell client's RunspacePool. The PowerShell server sends this message to notify a PowerShell client about server-side higher-layer specific application data.

PowerShell client's RunspacePool MUST be in a NegotiationSucceeded state (section 3.1.1.2.2) while processing this message.

3.1.5.4.14 GET_COMMAND_METADATA Message

The syntax of this message is specified in section 2.2.2.14.

The RunspacePool MUST be in an Opened state (section 3.1.1.2.2) when this message is sent. The PowerShell client sends this message to get command metadata from the server. When sending this PowerShell message and receiving responses from the server, the client uses similar data structures that are used for executing a pipeline (section 3.1.4.3).

The PowerShell client constructs a GUID to represent the pipeline, initializes the pipeline state (section 3.1.1.3.2) to Running, constructs the message (section 2.2.2.14), and sends it to the PowerShell server.

For more details on how a PowerShell client gets command metadata from a PowerShell server, see to section 3.1.4.5.

3.1.5.4.15 RUNSPACEPOOL_HOST_CALL Message

The syntax of this message is specified in section 2.2.2.15.

A PowerShell client's RunspacePool MUST be in an Opened or NegotiationSucceeded state (section 3.1.1.2.2) while processing this message.

This message is received by a PowerShell client from a PowerShell server as part of a wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to a RunspacePool. A PowerShell server sends this message to make a method call on a PowerShell client host.

The PowerShell client interprets the method and parameter information as described in section 2.2.6 and hands over the data to the higher-layer for its response. The PowerShell client collects the response from the higher-layer, if any, and sends a RUNSPACEPOOL_HOST_RESPONSE message (section 2.2.2.16) to the PowerShell server.

3.1.5.4.16 RUNSPACEPOOL_HOST_RESPONSE Message

The syntax of this message is specified in section 2.2.2.16.

This message MUST be sent if there is a response from the higher layer for a corresponding RUNSPACEPOOL_HOST_CALL message (section 3.1.5.4.15).

The RunspacePool MUST be in an Opened or NegotiationSucceeded state (section 3.1.1.2.2) when this message is sent.

While constructing this message, the PowerShell client MUST extract the "ci" (call id) value from the RUNSPACEPOOL_HOST_CALL message associated with the RunspacePool (section 3.1.5.4.15) and use the same value in the "ci" portion of the message.

If a response could not be constructed, the PowerShell client MUST close the RunspacePool as described in section 3.1.4.2.

3.1.5.4.17 PIPELINE_INPUT Message

The syntax of this message is specified in section 2.2.2.17.

The pipeline MUST be in Running state (section 3.1.1.3.2) and a successful response to the wxf:Command (section 3.2.5.3.4) message MUST have been received when this message is sent.

For more details on how a PowerShell client executes pipeline on a PowerShell server see section 3.1.4.3.

3.1.5.4.18 END_OF_PIPELINE_INPUT Message

The syntax of this message is specified in section 2.2.2.18.

This message MUST be sent only if the pipeline state (3.1.1.3.2) is Running.

For more details about how a PowerShell client executes a pipeline on a PowerShell server, see section 3.1.4.3.

3.1.5.4.19 PIPELINE_OUTPUT Message

The syntax of this message is specified in section 2.2.2.19.

This message is received by a PowerShell client from a PowerShell server as part of wxf:ReceiveResponse (section 3.2.5.3.8) targeted to a pipeline. A PowerShell server sends this message to notify a PowerShell client about a pipeline's Output data.

The PowerShell client's pipeline MUST be in a Running state while processing this message.

It is up to the PowerShell client to process this message and transmit the data to higher-layers.

For more details about how a PowerShell client executes a pipeline on a PowerShell server, see section 3.1.4.3.

3.1.5.4.20 ERROR_RECORD Message

The syntax of this message is specified in section 2.2.2.20.

This message is received by a PowerShell client from the PowerShell server as part of wxf:ReceiveResponse (section 3.2.5.3.8) targeted to a pipeline. The PowerShell server sends this message to notify a PowerShell client about a pipeline's Error data.

A PowerShell client's pipeline MUST be in a Running state while processing this message.

It is up to a PowerShell client to process this message and transmit the data to the higher-layers.

For more details about how a PowerShell client executes a pipeline on a PowerShell server, see section 3.1.4.3.

3.1.5.4.21 PIPELINE_STATE Message

The syntax of this message is specified in section 2.2.2.21.

A PowerShell server sends this message to a RunspacePool or pipeline on the PowerShell client.

The PowerShell client SHOULD ignore PIPELINE_STATE messages targeted to RunspacePools.

If this message is targeted to a pipeline, the PowerShell server sends it message to notify a PowerShell client about the pipeline's state. Once this message is received, the PowerShell client extracts the state information (section 2.2.3.4) from the message and updates the pipeline state (section 3.1.1.3.2) accordingly. Once a pipeline reaches a Completed or Failed or Stopped state (section 3.1.1.3.2), the PowerShell client MUST remove the pipeline from the corresponding RunspacePool's pipeline table (section 3.1.1.2.6) and the global pipeline table (section 3.1.1.1.2).

If this message is targeted to a pipeline, the PowerShell client's pipeline MUST be in a Running state (section 3.1.1.3.2) while processing this message. If the pipeline is not in a Running state, then the PowerShell client SHOULD ignore this message.

The details of how a PowerShell client executes a pipeline on a PowerShell server are specified in section 3.1.4.2.

3.1.5.4.22 DEBUG_RECORD Message

The syntax of this message is specified in section 2.2.2.22.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's Debug data.

The PowerShell client's pipeline MUST be in a Running state while processing this message.

It is up to the PowerShell client to process this message and transmit the data to higher-layers.

For more details about how a PowerShell client executes pipeline on a PowerShell server see section 3.1.4.3.

3.1.5.4.23 VERBOSE_RECORD Message

The syntax of this message is specified in section 2.2.2.23.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's verbose data.

The PowerShell client's pipeline MUST be in a Running state while processing this message.

It is up to the PowerShell client to process this message and transmit the data to the higher-layer.

For more details about how a PowerShell client executes a pipeline on a PowerShell server, see section 3.1.4.3.

3.1.5.4.24 WARNING_RECORD Message

The syntax of this message is specified in section 2.2.2.24.

The PowerShell server sends this message to notify a client about a pipeline's Warning data.

The PowerShell client's pipeline MUST be in a Running state while processing this message.

It is up to the PowerShell client to process this message and transmit the data to higher-layers.

For more details about how a PowerShell client executes the pipeline on a PowerShell server, see section 3.1.4.3.

3.1.5.4.25 PROGRESS_RECORD Message

The syntax of this message is specified in section 2.2.2.25.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's Progress data.

The PowerShell client's pipeline MUST be in a Running state while processing this message.

It is up to the PowerShell client to process this message and transmit the data to higher-layers.

For more details about how a PowerShell client executes a pipeline on a PowerShell server, see section 3.1.4.3.

3.1.5.4.26 PIPELINE_HOST_CALL Message

The syntax of this message is specified in section 2.2.2.26.

The PowerShell server sends this message to make a method call on PowerShell client's host.

The PowerShell client's pipeline MUST be in a Running state (section 3.1.1.2.2) and a successful response to wxf:Command (section 3.2.5.3.4) message MUST be received while processing this message.

The PowerShell client interprets the method and parameter information, as described in section 2.2.6, and transmits the data to higher-layer for its response. The PowerShell client collects the response from the higher-layer, if any, and sends a PIPELINE_HOST_RESPONSE message (section 2.2.2.27) to the server.

3.1.5.4.27 PIPELINE_HOST_RESPONSE Message

The syntax of this message is specified in section 2.2.2.27.

This message is targeted to a pipeline on the server. This message MUST be sent if there is a response from a higher-layer for a corresponding PIPELINE_HOST_CALL message (section 3.1.5.4.26).

The pipeline MUST be in a Running state (section 3.1.1.3.2) and a successful response to wxf:Command (section 3.2.1.2.10) message is received when this message is sent.

While constructing this message, the PowerShell client MUST extract the "ci" (call id) value from the corresponding Host Method call associated with the pipeline message and use the same value in the "ci" portion of the message.

If a response could not be constructed, the PowerShell client MUST stop the pipeline as described in section 3.1.4.4.

3.1.5.4.28 CONNECT_RUNSPACEPOOL Message

The syntax of this message is specified in section 2.2.2.28. The RunspacePool MUST be in Connecting state (section 3.1.1.2.2) when this message is sent. This message MUST be sent only once per RunspacePool from a PowerShell client to a PowerShell server.

3.1.5.4.29 RUNSPACEPOOL_INIT_DATA Message

The syntax of this message is specified in section 2.2.2.29. This message is targeted to the RunspacePool. The PowerShell server sends this message as a response to a CONNECT_RUNSPACEPOOL message (section 2.2.2.28). When this message is received, the PowerShell client extracts and updates the RunspacePool information. When this message is received, the RunspacePool MUST be in the Opened state.

3.1.6 Timer Events

The Session Key Transfer timer (section 3.1.1.2.8) MUST be started by the PowerShell remoting protocol when it sends a PUBLIC_KEY message (section 3.1.5.4.3). There MUST be a unique timer for each PUBLIC_KEY message. Upon receipt of an ENCRYPTED_SESSION_KEY message (section 3.1.5.4.4) for that PUBLIC_KEY message, the timer MUST be canceled.

The Session Key Transfer timer MUST expire after the number of milliseconds given by the SessionKeyTransferTimeoutms (section 3.1.1.2.8). Upon expiration of this timer, the PowerShell remoting protocol MUST close the associated RunspacePool, as described in section 3.1.4.2.

3.1.7 Other Local Events

If there are any errors while processing a RunspacePool message, then that RunspacePool MUST be Closed as specified in section 3.1.1.2.2.

If there are any errors while processing a pipeline message, then that pipeline MUST be stopped as specified in section 3.1.1.3.2.

3.2 Server Details

3.2.1 Abstract Data Model

3.2.1.1 Global Data

Global server data MUST be initialized as specified in section 3.2.3.

3.2.1.1.1 MS-WSMV ShellID to RunspacePool Table

The PowerShell server MUST maintain a global table that maps a Web Services Management Protocol Extensions for Windows Vista [MS-WSMV] shell to data associated with a RunspacePool (see section 3.2.1.2).

The key used in the table is the value of the ShellID selector sent back in the wxf:ResourceCreated message (see [MS-WSMV] section 3.1.4.5.2).

3.2.1.1.2 MS-WSMV CommandId to Pipeline Table

The PowerShell server MUST maintain a global table that maps a Web Services Management Protocol Extensions for Windows Vista [MS-WSMV] command to data associated with a pipeline (see section 3.2.1.3).

The key used in the table is the value of the CommandId element sent back in the wxf:CommandResponse message (see [MS-WSMV] section 2.2.4.8).

3.2.1.2 RunspacePool Data

3.2.1.2.1 GUID

Each RunspacePool has an associated GUID. The GUID is initialized to the RPID (see section 2.2.1) used in the SESSION_CAPABILITY message (see section 2.2.2.1) associated with the RunspacePool.

3.2.1.2.2 RunspacePool State

Each RunspacePool has an associated state. The state of a newly created RunspacePool MUST be initialized to: BeforeOpen.

Section 2.2.3.4 lists available states and describes the data type used to encode the state in PowerShell remoting protocol messages.

Sections 3.2.5.4.1 and 3.2.5.4.2 describe how RunspacePool state transitions from the BeforeOpen state to the NegotiationSucceeded state, and then to the Opened state. From the Opened state, a RunspacePool can reach either the Closed or Broken state, mentioned in section 2.2.3.4.

A PowerShell client can close a RunspacePool by sending wxf:Delete message (section 3.1.5.3.11). When a PowerShell server receives this message, the PowerShell server MUST stop all the running pipeline, change the RunspacePool state to Closed, and send a wxf:DeleteResponse (section 3.2.5.3.1).

The PowerShell server can change the RunspacePool state from Opened to Broken at any time if the PowerShell server determines that something is wrong with the RunspacePool (such as a Network

connection getting lost or a corrupted RunspacePool). Before changing the state to Broken, the PowerShell server MUST stop all the running pipelines. After changing the RunspacePool state to Broken, the PowerShell server MUST send a RUNSPACEPOOL_STATE message (section 3.2.5.4.9) with a Broken state to the PowerShell client if there is a pending wxf:Receive message (see section 3.2.5.3.7).

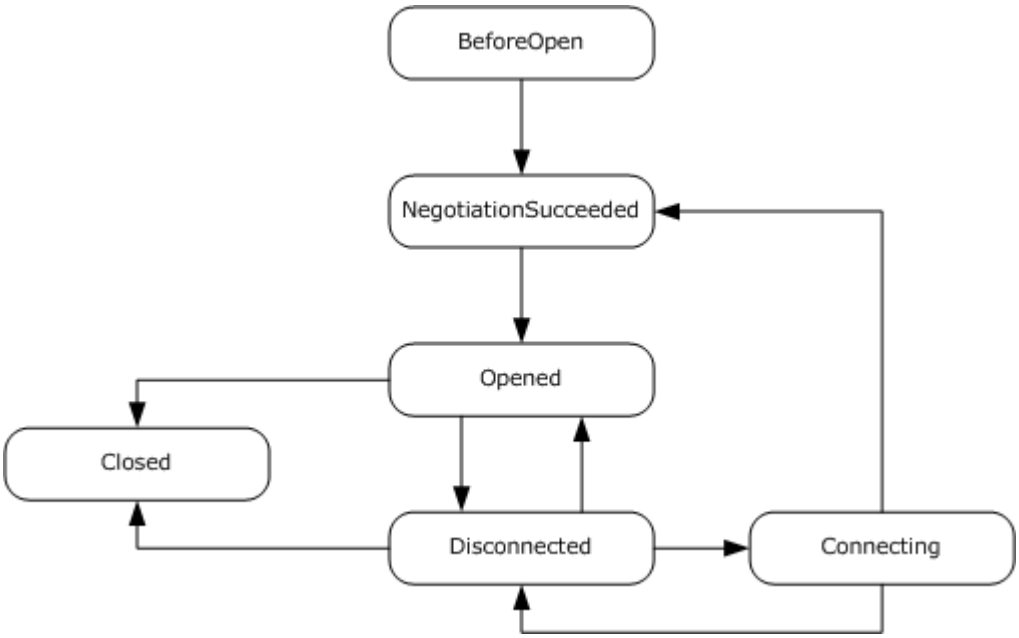


Figure 4: Server RunspacePool states and transitions

3.2.1.2.3 Defragmentation Data

The current state of defragmentation (see sections 2.2.4 and 3.2.5.1.2 for PSRP messages (section 2.2.1) sent by the PSRP server and targeted at the RunspacePool.

Server-side defragmentation data for a RunspacePool includes exactly the same type of information as client-side defragmentation data (section 3.1.1.2.3).

3.2.1.2.4 Queue of Outgoing Messages

The PowerShell server MUST maintain a first in, first out (FIFO) queue of messages (see section 2.2.1) ready to be sent to the particular RunspacePool on the PowerShell client. The element at the beginning of the queue can be a whole message or a suffix of a message (when the prefix has already been fragmented and sent to the client); all other elements of the queue are whole messages. See section 3.2.5.1.1 for details on how the queue is used.

The queue is initialized to be empty.

3.2.1.2.5 HostInfo

The PowerShell server MUST store the HostInfo received in the INIT_RUNSPACEPOOL message (see section 2.2.2.2) and make it available to commands executed in pipelines that use a host associated with a RunspacePool, instead of using a separate host associated with a pipeline.

3.2.1.2.6 Host calls CI Table

The PowerShell server MUST maintain a table associating an integer identifier with outstanding RUNSPACEPOOL_HOST_CALL messages (section 2.2.2.15) originating from the higher-layer.

The table is used to map the PowerShell server requests to corresponding PowerShell client responses with the "ci" property of RUNSPACEPOOL_HOST_RESPONSE messages (see section 2.2.2.16).

3.2.1.2.7 Session Key

The PowerShell server MUST store and reuse the session key generated and sent by the PowerShell server in the ENCRYPTED_SESSION_KEY message (section 2.2.2.4).

3.2.1.2.8 Public Key

The PowerShell server MUST store public key generated and sent by the PowerShell client in the PUBLIC_KEY message (section 2.2.2.3).

3.2.1.2.9 Minimum and Maximum Number of Runspaces in the Pool

Each RunspacePool has an associated minimum and maximum number of runspaces to be present in the pool of runspaces. Minimum and maximum are initialized to the values requested by the PowerShell client in INIT_RUNSPACEPOOL messages (see section 2.2.2.2).

The number of runspaces in the RunspacePool MUST be within the limits expressed by the minimum and maximum numbers.

3.2.1.2.10 Runspace Table

A PowerShell server MUST maintain a table with information about each runspace associated with a RunspacePool. Information associated with each runspace is described in section 3.2.1.4.

The table of runspace availability is initialized to any number of runspaces within the constraints from section 3.2.1.2.9.

At any time, the PowerShell server MAY remove a runspace in an available state (see section 3.2.1.4.1) from the pool (for example, to conserve resources) as long as the constraints from section 3.2.1.2.9 are not violated.

3.2.1.2.11 Pending pipelines queue

The PowerShell server MUST maintain a queue with pending requests to run a pipeline.

When a CREATE_PIPELINE message (see section 2.2.2.10) comes at a time when all runspaces in a RunspacePool are busy (see section 3.2.1.4.1), the request is put into the pending pipelines queue.

Later, when a runspace in the RunspacePool becomes available, the RunspacePool MUST pick the first pipeline from the pending pipelines queue and execute the pipeline using the runspace.

3.2.1.3 Pipeline Data

3.2.1.3.1 GUID

Each pipeline has an associated GUID. The GUID is initialized to the PID (see section 2.2.1) used in the first received PowerShell Remoting Protocol message associated with the pipeline.

3.2.1.3.2 Pipeline State

Each pipeline has an associated state.

Section 2.2.3.5 lists available states and describes the data type used to encode the state in PowerShell remoting protocol messages.

For details about how a pipeline state transitions from NotStarted to Running, see section 3.2.5.4.10.

The PowerShell server can change the pipeline state from Running to Failed at any time if it determines that there is something wrong with the pipeline (such as a network connection getting lost, a corrupted RunspacePool is in bad state, or a pipeline failed while executing). After changing the pipeline state to Failed, the PowerShell server MUST send a PIPELINE_STATE message (section 3.2.5.4.21) with a Failed state to the PowerShell client.

When the pipeline state is changed to Completed, Stopped, or Failed, the PowerShell server MUST not send any PowerShell Remoting Protocol layer messages to the PowerShell client targeted to that particular pipeline.

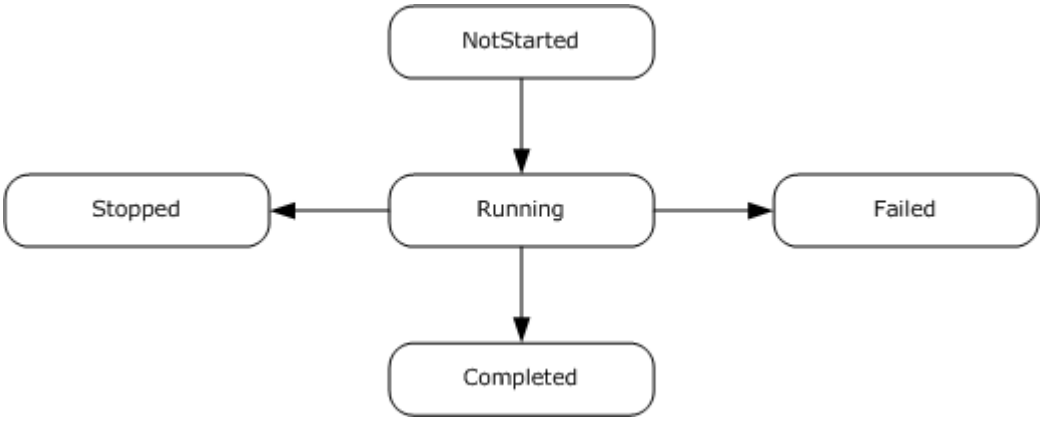


Figure 5: Server pipeline states and transitions

3.2.1.3.3 Defragmentation Data

The current state of defragmentation (see sections 2.2.4 and 3.2.5.1.2 for PSRP messages (section 2.2.1) sent by the PSRP server and targeted at the pipeline.

Server-side defragmentation data for a pipeline includes exactly the same type information as client-side defragmentation data (section 3.1.1.3.3).

3.2.1.3.4 Queue of Outgoing Messages

The PowerShell server MUST maintain a first in, first out (FIFO) queue of messages (see section 2.2.1) ready to be sent to the particular pipeline on the PowerShell client. The element at the beginning of the queue can be a whole message or a suffix of a message (when the prefix has already been fragmented and sent to the client); all other elements of the queue are whole messages. See section 3.2.5.1.1for details on how the queue is used.

The queue is initialized to be empty.

3.2.1.3.5 HostInfo

The PowerShell server MUST store the HostInfo received in the CREATE_PIPELINE message (see section 2.2.2.10) and make it available to commands executed in pipelines that use a separate host associated with a pipeline, instead of using a host associated with a RunspacePool.

3.2.1.3.6 Host Calls CI Table

The PowerShell server MUST maintain a table associating an integer identifier with outstanding PIPELINE_HOST_CALL message (section 2.2.2.26) originating from the higher layer.

The table is used to map the PowerShell server requests to corresponding client responses by the "c" property of PIPELINE_HOST_RESPONSE messages (see section 2.2.2.27).

3.2.1.4 Runspace Data

3.2.1.4.1 Runspace State

A runspace in a RunspacePool can be in any of the following states:

1. available: ready to run new pipelines.

2. busy: already running a pipeline.

Runspace state is initialized to "available" in newly created runspaces.

3.2.1.4.2 Currently Running Pipeline

A runspace in a busy state (see section 3.2.1.4.1) runs exactly one pipeline. A runspace MUST store a key associated with the pipeline (from the global table of pipelines, see section 3.2.1.1.2) this runspace is currently running.

When initialized, newly created runspaces do not have a currently running pipeline.

3.2.2 Timers

None.

3.2.3 Initialization

Server Initialization

- The tables described in sections 3.2.1.1.1 and 3.2.1.1.2 MUST be initialized to empty.

RunspacePool Initialization

- The Session Key (section 3.2.1.2.7) MUST be initialized to none.
- The Host calls CI Table (section 3.2.1.2.6) MUST be initialized to be empty.
- The Pending pipelines queue (section 3.2.1.2.11) MUST be initialized to be empty.
- The Public Key (section 3.2.1.2.8) MUST be initialized to an empty value.

Pipeline Initialization

- The state of a newly created pipeline (section 3.2.1.3.2) MUST be initialized to NotStarted.
- The Hosts calls CI Table (section 3.2.1.3.6) MUST be initialized to be empty.

3.2.4 Higher-Layer Triggered Events

1. When a RunspacePool is in an Opened state, the higher layer can trigger the following events on the server:
   - Reporting of user event to the client – see section 3.2.5.4.12.
   - Performing a host method call targeted at a RunspacePool (see section 3.2.5.4.15 and 3.2.5.4.16).
   - Initiating a session key exchange (see section 3.1.4.8).
     - Note: the PowerShell server MUST NOT start a session key exchange if another session key exchange is already in progress or has already been completed.
     - Note: the PowerShell server MUST notify the higher layer when a session key exchange is completed (see section 3.2.5.4.4), so that the higher layer can register when it can start to use secure strings in the higher-layer objects sent to the client.
2. When a Pipeline is in a Running state, the higher layer can trigger the following events on the server:
   - Emitting output from the Pipeline and reporting them to the client (see section 3.2.5.4.19).
   - Emitting non-terminating errors from the Pipeline and reporting them to the client – see section 3.2.5.4.20.
   - Changing the pipeline state (see section 3.2.5.4.21).
   - Emitting debug, warning or verbose messages from the Pipeline and reporting them to the client (see sections 3.2.5.4.22, 3.2.5.4.24 and 3.2.5.4.24).
   - Reporting progress of the pipeline (see section 3.2.5.4.25).
   - Performing a host method call targeted at the Pipeline (see sections 3.2.5.4.26 and 3.2.5.4.27).

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 General Rules

The message processing rules specified in [MS-WSMV] section 3.1.4, are applicable here as well.

1. The PowerShell server uses wxf:ReceiveResponse (section 3.2.5.3.8) messages to send data to a PowerShell client's RunspacePool or pipeline. While sending messages, the PowerShell server MUST follow the rules specified in section 3.2.5.1.1.

2. The PowerShell server receives data from the client as part of wxf:Send (section 3.2.5.3.5), wxf:Create (section 3.2.5.3.12), or wxf:Command (section 3.2.5.3.3) messages and constructs a PowerShell message, as per the rules specified in section 3.2.5.1.2. The PowerShell server determines whether a PowerShell message is targeted to a RunspacePool or pipeline, as per the rules specified in section 3.2.5.4 and section 2.2.1.

3. Some messages apply only to a RunspacePool, and are valid only when the RunspacePool is in certain states. The valid states for each message are listed in section 3.2.5.4. When a PowerShell server receives a message for a RunspacePool that is not in the correct state, the server MUST send a wxf:Fault message ([MS-WSMV] section 2.2.4.43) to the PowerShell client as a response to any pending wxf:Receive (section 3.1.5.3.7) messages, close the RunspacePool as specified in section 3.2.1.2.2, and discard any incoming messages for that specific RunspacePool.

4. Some messages apply only to a pipeline, and are valid only when the pipeline is in certain states. The valid states for each message are listed in section 3.2.5.4. When a PowerShell server receives a message for a pipeline that is not in the correct state, then the server MUST send a wxf:Fault message ([MS-WSMV] section 2.2.4.43) to the client as a response to any pending wxf:Receive (section 3.1.5.3.7) messages, stop the pipeline as specified in section 3.2.1.3.2, and discard the incoming messages for that specific pipeline.

5. If a PowerShell server receives a message that does not target any existing pipeline or RunspacePool, as per the data specified in section 3.2.1, then the PowerShell server MUST send a wxf:Fault message ([MS-WSMV] section 2.2.4.43) to the PowerShell client and ignore the message.

3.2.5.1.1 Rules for Sending Data

1. A PowerShell server MUST use the wxf:ReceiveResponse WS-MAN message (section 3.2.5.3.8) to send PowerShell messages to a PowerShell client.

2. When sending a PowerShell Remoting Protocol message (section 2.2.1), the message MUST first be placed at the end of the appropriate queue of outgoing messages (see sections 3.2.1.2.4 and 3.2.1.3.4), which will store the message until a wxf:Receive message comes from the PowerShell client

3. When the wxf:Receive message arrives from the PowerShell client, the PowerShell server MUST dequeue the entire PowerShell Remoting Protocol message or a suffix of a PowerShell Remoting Protocol message from the beginning of the appropriate queue (see sections 3.2.1.2.4 and 3.2.1.3.4), and fragment it (see section 2.2.4). If the appropriate queue is empty, then the PowerShell server MUST block and wait for new items to be added to the queue (while following rules for keeping the connection alive specified in [MS-WSMV], section 3.1.4.14). The **FragmentID** fields for a particular PowerShell Remoting Protocol message MUST be numbered consecutively beginning with 0, and the fragments MUST be sent in ascending order of the FragementID using wxf:ReceiveResponse (section 3.2.5.3.8).

4. If multiple fragments from step 3 can fit into a single WS-MAN message, then the single WS-MAN message SHOULD include as many fragments as possible (see [MS-WSMV], section 3.1.4.1.7). If any fragments did not fit into the wxf:ReceiveResponse message, then the suffix of the message associated with those fragments MUST be put back at the beginning of the appropriate queue (see section 2.2.4) to be processed when the next wxf:Receive message comes from the client. If more data could fit into the wxf:ReceiveResponse message and the queue is still not empty, then

the server SHOULD go back to the previous step to generate more fragments for the wxf:ReceiveResponse message.

3.2.5.1.2 Rules for Receiving Data

1. The PowerShell server receives data from the PowerShell client using wxf:Create, wxf:Command, or wxf:Send MS-WSMV message. Each MS-WSMV message contains one or more fragments. See section 2.2.4 for the format of a fragment.

2. When one of the MS-WSMV messages with fragmented data is received, the PowerShell server extracts the **Blob** field of the fragment and appends the extracted data to the **PartiallyDefragmentedPsrpMessage** field of the targeted RunspacePool (section 3.2.1.2.3) or pipeline (section 3.2.1.3.3). If the data is received using wxf:Create (section 3.2.5.3.1) or wxf:Command (section 3.2.5.3.3), the appropriate data MUST be decoded using base-64 format.

3. After an EndFragment packet is received, a whole PSRP message (see section 2.2.1) is stored in the **PartiallyDefragmentedPsrpMessage** field and can be handled as described in section 3.2.5.4.

4. The PowerShell server should compare the **ObjectId** and **FragmentId** fields of each received fragment with the **LastObjectId** and **LastFragmentId** data stored in the ADM and then update the ADM. If at any point it is determined that the fragments are not received in ascending order of **FragmentID** with the same **ObjectID**, the PowerShell MUST close the appropriate RunspacePool or stop the appropriate pipeline.

3.2.5.2 Sequencing Rules

The following is a typical sequence of activity for a PowerShell server's RunspacePool and pipeline

1. The PowerShell server creates a RunspacePool and the RunspacePool gets into the Opened state. Refer to sections 3.2.5.4.1 and 3.2.5.4.2 for more details.

2. When a RunspacePool is in an Opened state, RunspacePool-specific messages such as SET_MAX_RUNSPACES (section 3.2.5.4.6), SET_MIN_RUNSPACES (section 3.2.5.4.7), and GET_AVAILABLE_RUNSPACES (section 3.2.5.4.11) may be received by the PowerShell server's RunspacePool. For more details about which messages can be received, see section 3.2.5.4.

3. When a RunspacePool is in an Opened state, a PowerShell client may send a CREATE_PIPELINE (section 3.2.5.4.10) to the PowerShell server to start executing a pipeline on the server. The PowerShell server creates a pipeline and changes the pipeline state to Running.

4. When the RunspacePool is in Opened state, a PowerShell server may send RunspacePool-specific messages, such as RUNSPACEPOOL_HOST_CALL (section 3.2.5.4.15) and RUNSPACEPOOL_STATE (section 3.2.5.4.9).

5. When a pipeline is in the Running state, a PowerShell server may send pipeline-specific messages, such as PIPELINE_OUTPUT (section 3.2.5.4.19) and PIPELINE_HOST_CALL (section 3.2.5.4.26). For more details about the exact messages that can be received, see section 3.2.5.4.

6. The PowerShell server may choose to stop or fail a pipeline at any time (section 3.2.1.3.2) as long as the pipeline is in a Running state. After changing the state, the PowerShell server MUST send a PIPELINE_STATE message (section 3.2.5.4.21) with the appropriate state information to the PowerShell client.

7. A PowerShell server may choose to close a RunspacePool and associated pipelines at any time, as long as the RunspacePool is in an Opened state. After changing the state, the PowerShell server MUST send a RUNSPACEPOOL_STATE message (section 3.2.5.4.9) with appropriate state information to the PowerShell client.

8. When a RunspacePool is in a Closed state, that specific RunspacePool is not allowed for executing pipelines.

3.2.5.3 Rules for Processing WS-Man Messages

Transportation using WS-MAN is as specified in section 3.1.5.3.

A PowerShell server SHOULD participate in this protocol sequence by sending response messages as described in the following subsections.

3.2.5.3.1 Rules for the wxf:Create message

A PowerShell client uses the wxf:Create message to create a RunspacePool on the PowerShell server, as specified in section 3.1.5.3.3.

Upon receiving this message, the PowerShell server validates the option with the name "protocolversion" and compares the value of this option against version "2.1" (taken from the table in section 3.1.5.3.1). The PowerShell server MUST send a wxf:Fault message as described in section 3.2.5.3.2 when any of the following conditions are true:

- The "protocolversion" option is missing;
- The major version number of the "protocolversion" option is not equal to 2.

If the validation as described earlier is successful, the PowerShell server creates a RunspacePool instance, initializes its state to BeforeOpen (section 3.2.1.2.2), and adds it into the MS-WSMV shell to a RunspacePool table (section 3.2.1.1.1). The PowerShell server MUST then send a wxf:ResourceCreated message (section 3.2.5.3.2).

The wxf:Create message MAY contain "creationXml" data, as described in section 3.1.5.3.3. If "creationXml" data is present in the message, the data will be in Base-64 encoded format. The PowerShell server decodes this Base-64 data and processes the message per the rules described in section 3.2.5.1. If the rules specified in section 3.2.5.1 result in a wxf:Fault message, then the PowerShell server MUST change the RunspacePool state to Broken.

3.2.5.3.2 Rules for the wxf:ResourceCreated Message

A PowerShell client uses the wxf:Create message to create a RunspacePool on the PowerShell server, as specified in section 3.1.5.3.1. A PowerShell server implementation MUST process the wxf:Create message and send either a success response (using the wxf:ResourceCreated message specified in [MS-WSMV], section 3.1.4.5.2) or a failure response (using the wxf:Fault message, specified in [MS-WSMV] section 2.2.4.43). The PowerShell server MUST use wxf:ReceiveResponse messages to send any data (section 3.2.5.3.8) targeted to that RunspacePool.

The wxf:Create message sent by clients MUST contain an option with the name "protocolversion" and the value "2.1" or "2.2". If the server does not accept a client's protocol version, then the server MUST send an error message to the client using a wxf:Fault message as specified in [MS-WSMV], section 2.2.4.43.

The following information MUST be included in the wxf:Fault message.

Element	Value
Code	2152991685
Machine	A string that SHOULD specify the machine name where this fault occurred.
Message	A string message in the following format. `<PSProtocolVersionError ServerProtocolVersion="x.y" ServerBuildVersion="a.b.cdef.g">detailed error message </PSProtocolVersionError>` Where "x.y" represents the server's ProtocolVersion (currently 2.1), and "a.b.cdef.g" represents the server's build number (such as 7.0.7000.0).

A server MUST be compatible with minor version changes; in other words, a server could accept a client's packet even if the protocol version was specified as "2.1".

Upon successful processing of a wxf:Create message, the PowerShell remoting protocol MUST create a Shell instance, store it in [MS-WSMV] shell to a RunspacePool table (section 3.2.1.1.1), and return a reference to it as wsa:EndPointReference as specified in [WSAddressing] and constrained by [DMTF-DSP0226].

The wsa:EndpointReference encapsulated within the wxf:ResourceCreated (as specified in [MS-WSMV] section 3.1.4.5.2) contains a reference to the newly created Shell instance. This address is used in all subsequent messages to the Shell instance; that is, it is used in wxf:Delete (section 3.1.5.3.11), wxf:Command (section 3.1.5.3.3), wxf:Signal (section 3.1.5.3.9), wxf:Send (section 3.1.5.3.5), and wxf:Receive (section 3.1.5.3.7) messages. The following list describes the additional normative constraints on the wsa:EndpointReference message.

- **ReferenceParametersp**: This required element identifies the created Shell instance.
- **ResourceURI**: The value of ResourceURI is implementation-specific.<10>
- **SelectorSet**: The value of the Name attribute of the Selector element MUST contain the GUID identifying the new Shell.

3.2.5.3.3 Rules for the wxf:Command Message

A PowerShell client uses the wxf:Command message to execute a pipeline on the PowerShell server, as described in section 3.1.5.3.3.

Upon receiving this message, the PowerShell server attempts to get the RunspacePool instance, using the ShellID specified in the wxf:Command message, from the [MS-WSMV] shell to the RunspacePool table (section 3.2.1.1.1). If a RunspacePool instance is not found in the table, or if the RunspacePool is not in an Opened state, then the PowerShell server MUST send a wxf:Fault message.

If a corresponding RunspacePool instance is found, then PowerShell creates a pipeline instance and initializes its state to NotStarted. The PowerShell server then adds the pipeline instance into [MS-WSMV] command to pipeline table (section 3.2.1.1.2). If a RunspacePool instance is not found, then the PowerShell server MUST send a wxf:Fault message.

The wxf:Command message MAY contain "Arguments", as described in section 3.1.5.3.3. If Arguments data is present in the message, the data will be in Base-64 encoded format. The PowerShell server decodes this Base-64 data and processes the message as per the rules described in section 3.2.5.1.

Upon successfully processing a wxf:Command message, the PowerShell server MUST send a wxf:CommandResponse message (section 3.2.5.3.4).

3.2.5.3.4 Rules for the wxf:CommandResponse Message

A PowerShell client initiates a pipeline invocation using the message structure specified in section 3.1.5.3.3. A PowerShell server implementation MUST process this message and send a response, if successful, using a wxf:CommandResponse message, as specified in [MS-WSMV] section 2.2.4.8.

3.2.5.3.5 Rules for the wxf:Send Message

A PowerShell client uses the wxf:Send message to send data to a RunspacePool or pipeline on the PowerShell server, as described in section 3.1.5.3.5.

Upon receiving this message, the PowerShell server attempts to get the RunspacePool instance or pipeline instance, using the ShellID and the CommandID specified in the wxf:Send message (section 3.1.5.3.5) and the RunspacePool and the pipeline tables (section 3.2.1.1). If a corresponding RunspacePool or pipeline instance is not found, then the PowerShell server MUST send a wxf:Fault message.

If a corresponding RunspacePool or pipeline instance is found, then PowerShell server extracts the data from wxf:Send message and processes the data as per the rules described in section 3.2.5.1.

Upon successfully processing the message, PowerShell server MUST send a wxf:SendResponse message (section 3.2.5.3.6).

Only the following PowerShell messages are allowed to be sent to the server using the wxf:Send message: SESSION_CAPABILITY (section 3.1.5.4.1), INIT_RUNSPACEPOOL (section 3.1.5.4.2), PUBLIC_KEY (section 3.1.5.4.3), SET_MAX_RUNSPACES (section 3.1.5.4.6), SET_MIN_RUNSPACES (section 3.1.5.4.7), CREATE_PIPELINE (section 3.1.5.4.10), GET_AVAILABLE_RUNSPACES (section 3.1.5.4.11), RUNSPACEPOOL_HOST_RESPONSE (section 3.1.5.4.16), PIPELINE_INPUT (section 3.1.5.4.17), END_OF_PIPELINE_INPUT (section 3.1.5.4.18), PIPELINE_HOST_RESPONSE (section 3.1.5.4.27).

3.2.5.3.6 Rules for the wxf:SendResponse Message

A PowerShell client sends data to a RunspacePool or a pipeline instance on the server, as specified in section 3.1.5.3.5. A PowerShell server implementation MUST process the wxf:Send message and send a response message, if successful, using wxf:SendResponse message, as specified in [MS-WSMV] section 3.1.4.13.

3.2.5.3.7 Rules for the wxf:Receive Message

A PowerShell server implementation MUST process a wxf:Receive message by sending back a wxf:ReceiveResponse message, as specified in section 3.2.5.3.8.

3.2.5.3.8 Rules for the wxf:ReceiveResponse Message

When a PowerShell client is ready to receive output it sends a wxf:Receive request, as specified in section 3.1.5.3.7. A PowerShell server implementation MUST process this wxf:Receive message and send a response message using a wxf:ReceiveResponse message, as specified in [MS-WSMV], section 3.1.4.14. A PowerShell server implementation MUST send the wxf:ReceiveResponse message only after it receives a wxf:Receive message from the PowerShell client for the corresponding RunspacePool or pipeline.

A PowerShell server implementation MUST use the stream name "stdout" to send data to the client. A PowerShell client expects data from the PowerShell server in this stream only.

The following information MUST be included in the Stream element of the message.

Element	Value
Name	Stdout
CommandId	This attribute MUST be identical to that sent in the wxf:CommandResponse for the executed pipeline message, as specified in 3.2.5.3.4. This attribute MUST NOT be specified if the wxf:ReceiveResponse message is targeted to the RunspacePool.

The body of the Stream element MUST contain the actual data. The data MUST be in the form as described in Messages (section 2).The following information SHOULD be included in the CommandState element of the message if the message is meant for a pipeline.

Element	Value
CommandId	This attribute MUST NOT be specified if the wxf:ReceiveResponse message is targeted to the RunspacePool. If present, this attribute MUST be identical to that sent in the wxf:CommandResponse for the executed pipeline message, as specified in section 3.2.5.3.4. This element may or may not be present in every wxf:ReceiveResponse message. If present, the value in the State attribute identifies the Command State.
State	The value of the attribute identifies the state of the wxf:Command. This Element may or may not present in every wxf:ReceiveResponse packet. A value of http://schemas.microsoft.com/wbem/wsman/1/windows/shell/CommandState/Done specifies that this wxf:ReceiveResponse packet is the final wxf:ReceiveResponse message from the PowerShell server for that particular pipeline (as identified by CommandId) or for that particular RunspacePool (as identified by ShellId selector).

As described earlier, the wxf:ReceiveResponse messages MUST NOT be sent for a particular RunspacePool or pipeline when a CommandState/Done state message is sent.

The PowerShell server uses wxf:ReceiveResponse messages to send PowerShell Remoting Protocol messages to PowerShell clients, if any, as per the rules described in section 3.2.5.4.

3.2.5.3.9 Rules for the wxf:Signal Message

A PowerShell client uses the wxf:Signal message to stop an executing pipeline on the PowerShell server, as described in section 3.1.5.3.9.

The PowerShell server MUST process this message only if the message is targeted to a pipeline instance and the value of the <Code> element MUST be "powershell/signal/crtl_c" as described in section 3.1.5.3.9. If these constraints are not met, then the PowerShell server MUST send a wxf:Fault message to the PowerShell client.

If validation is successful, then the PowerShell server tries to get the pipeline instance, using the ShellID and the CommandID specified in the message, from the pipeline table (section 3.2.1.1.2). If a corresponding pipeline instance is not found, then the PowerShell server MUST send a wxf:Fault message.

If a corresponding pipeline instance is found, then the PowerShell server stops the pipeline from further execution (section 3.2.1.3.2), sends a PIPELINE_STATE message (section 3.2.5.4.21) with a state of Stopped, and sends a wxf:SignalResponse message (section 3.2.5.3.10).

3.2.5.3.10 Rules for the wxf:SignalResponse Message

A PowerShell client sends a wxf:Signal request to stop an executing a pipeline on the PowerShell server, as specified in section 3.1.5.3.9. A PowerShell server implementation MUST process this message and send a response message, if successful, using the wxf:SignalResponse message, as specified in [MS-WSMV] section 3.1.4.12.

3.2.5.3.11 Rules for the wxf:Delete Message

A PowerShell client uses the wxf:Delete message to close a RunspacePool on the PowerShell server as described in section 3.1.5.3.11.

The PowerShell server tries to get the RunspacePool instance, using the ShellID specified in the message, from the RunspacePool table (section 3.2.1.1.1). If a corresponding RunspacePool instance is not found, then the PowerShell server MUST send a wxf:Fault message.

If a corresponding RunspacePool instance is found, then the PowerShell server closes the RunspacePool (section 3.2.1.2.2) and sends a wxf:DeleteResponse message (section 3.2.5.3.12). Before a server closes a RunspacePool, it SHOULD stop all the pipelines currently executing inside that RunspacePool.

3.2.5.3.12 Rules for the wxf:DeleteResponse Message

A PowerShell client sends a wxf:Delete message to close the associated RunspacePool and any active pipelines in the RunspacePool, as specified in section 3.1.5.3.11.

A PowerShell server implementation MUST process this message and send a response message, if successful, using wxf:DeleteResponse as described in [MS-WSMV] section 3.1.4.4.1.

3.2.5.3.13 Rules for the wxf:Fault Message

The PowerShell server uses the wxf:Fault message (as specified in [MS-WSMV] section 2.2.4.43) to inform the PowerShell client about a failure related to processing any of the WS-Man Messages received from the PowerShell client and described above.

3.2.5.3.14 Rules for the wxf:Connect Message

A PowerShell client uses the wxf:Connect message to connect to an existing RunspacePool on the PowerShell server, as specified in section 3.1.5.3.14.

Upon receiving this message, the PowerShell server compares the "protocolversion" option against the value "2.2" (see section 3.1.5.3.1). The PowerShell server MUST send a wxf:Fault message as specified in section 3.1.5.3.2 when either of the following conditions is true:

- The "protocolversion" option is missing.
- The "protocolversion" option is present, but the major version number of the value it contains is not equal to 2.

The wxf:Connect message will contain "connectXml" data, as specified in section 3.1.5.3.14. The PowerShell server decodes this Base-64 data and processes the message per the rules described in section 3.2.5.1. The server expects this payload to contain a SESSION_CAPABILITY message

followed by a CONNECT_RUNSPACEPOOL message. If the expected payload is not found, the server MUST send a wxf:Fault message to the client.

3.2.5.3.15 Rules for the wxf:ConnectResponse Message

A PowerShell client uses the wxf:Connect message to create a RunspacePool or a pipeline on the PowerShell server, as specified in section 3.1.5.3.14. A PowerShell server implementation MUST process the wxf:Connect message and send either a wxf:ConnectResponse message (as specified in [MS-WSMV] section 3.1.4.5.2) to indicate success, or a wxf:Fault message (as specified in [MS-WSMV] section 2.2.4.43) to indicate failure. The PowerShell server MUST use wxf:ConnectResponse messages to send server SESSION_CAPABILITY messages back to the client.

When connecting to a RunspacePool, the wxf:Connect message sent by clients MUST contain an option with the name "protocolversion" and the value "2.2". If the server does not accept a client's protocol version, then the server MUST send an error message to the client using a wxf:Fault message (as specified in [MS-WSMV] section 2.2.4.43).

The following information MUST be included in the wxf:Fault message.

Element	Value
Code	The value 2152991685.
Machine	A string that SHOULD specify the machine name where the fault occurred.
Message	A string message in the following format: "<PSProtocolVersionError ServerProtocolVersion="x.y" ServerBuildVersion="a.b.cdef.g"> message </PSProtocolVersionError>" Where "x.y" represents the server's ProtocolVersion (currently 2.1), and "a.b.cdef.g" represents the server's build number (such as 7.0.7000.0) and "message" is an unstructured text that SHOULD describe the cause of the fault in detail.

3.2.5.3.16 Rules for the wxf:Disconnect Message

A PowerShell client uses the wxf:Disconnect message to disconnect a RunspacePool on the PowerShell server as described in section 3.1.5.3.16.

The PowerShell server attempts to obtain the RunspacePool instance from the RunspacePool table, using the ShellID specified in the message (see section 3.2.1.1.1). If a corresponding RunspacePool instance is not found, then the PowerShell server MUST send a wxf:Fault message.

If a corresponding RunspacePool instance is found, then the PowerShell server disconnects the RunspacePool (as specified in section 3.2.1.2.2) and sends a wxf:DisconnectResponse message (as specified in section 3.2.5.3.17). The server can later be reconnected to using the wxf:Reconnect message. Once disconnected, the server will reject all requests related to that RunspacePool until it is once again reconnected.

3.2.5.3.17 Rules for the wxf:DisconnectResponse Message

A PowerShell client sends a wxf:Disconnect message to disconnect the associated RunspacePool, as specified in section 3.1.5.3.16. A PowerShell server implementation MUST process this message and send a response message, if successful, using wxf:DisconnectResponse as described in [MS-WSMV] section 3.1.4.15.

3.2.5.3.18 Rules for the wxf:Reconnect Message

A PowerShell client uses the wxf:Reconnect message to reconnect to a disconnected RunspacePool on the PowerShell server, as specified in section 3.1.5.3.18.

The PowerShell server attempts to obtain the RunspacePool instance from the RunspacePool table, using the ShellID specified in the message (see section 3.2.1.1.1). If a corresponding RunspacePool instance is not found, then the PowerShell server MUST send a wxf:Fault message.

If a corresponding RunspacePool instance is found, then the PowerShell server reconnects the RunspacePool (as specified in section 3.2.1.2.2) and sends a wxf:ReconnectResponse message (as specified in section 3.2.5.3.19).

3.2.5.3.19 Rules for the wxf:ReconnectResponse Message

A PowerShell client sends a wxf:Reconnect message to reconnect to the associated RunspacePool, as specified in section 3.1.5.3.18. A PowerShell server implementation MUST process this message and send a response message, if successful, using wxf:ReconnectResponse as specified in [MS-WSMV] section 3.1.4.16.

3.2.5.4 Rules for Processes PowerShell Messages

See the general protocol rules described in section 3.2.5.1.The following sections describe the impact of various PowerShell Remoting Protocol messages (section 2.2) on a PowerShell server.

3.2.5.4.1 SESSION_CAPABILITY Message

The syntax of this message is specified in section 3.1.5.4.1.

3.2.5.4.1.1 Receiving from the Client

The server waits immediately after it is started for the SESSION_CAPABILITY message. It uses this message to determine the client's capabilities.

When this message is processed, the RunspacePool MUST be in the BeforeOpen state (section 3.2.1.2.2).

The PowerShell server processes the message and validates the actual data received from the client with the expected data given in the following table.

Name	Expected value
protocolversion	Major version = 2. Any minor version numbers.
PSVersion	Major version = 2. Any minor version numbers.
SerializationVersion	Major version = 1. Any minor version numbers.

If expected versions are received from the client, the PowerShell server changes the RunspacePool state to NegotiationSucceeded (section 3.2.1.2.2). Otherwise the server MUST change the RunspacePool state to Broken.

If the state changed to NegotiationSucceeded, then the PowerShell server extracts the RPID from the PowerShell Remoting Protocol message (section 2.2.1) and stores it as the GUID (section 3.2.1.2.1) of the RunspacePool.

3.2.5.4.1.2 Sending to the Client

If the expected versions have not been received from the PowerShell client (section 3.2.5.4.1.1) and the SESSION_CAPABILITY message is received through wxf:Send message (section 3.2.5.3.5), then the PowerShell server MUST send a wxf:Fault message to the PowerShell client.

If expected versions have been received from the client (section 3.2.5.4.1.1), then the PowerShell server MUST send a SESSION_CAPABILITY message in response to a PowerShell client SESSION_CAPABILITY message.

The PowerShell server sends a response to the PowerShell client with its SESSION_CAPABILITY message (section 2.2.2.1) using the wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the RunspacePool. The **RPID** field (section 2.2.1) of the SESSION_CAPABILITY message sent by the PowerShell server MUST be zeroed out.

The SESSION_CAPABILITY message MUST have the following properties when it is sent to the client.

Name	Value to send
protocolversion	MUST be 2.0 when client sent protocolversion=2.0; otherwise, MUST be 2.1 or 2.2.
PSVersion	MUST be 2.0.
SerializationVersion	MUST be 1.1.0.1.
TimeZone	The TimeZone property MUST be omitted.

3.2.5.4.2 INIT_RUNSPACEPOOL Message

The syntax of this message is specified in section 3.1.5.4.2.

When this message is processed, the RunspacePool's state (section 3.2.1.2.2) MUST be in the NegotiationSucceeded state.

The PowerShell server gathers application private data from higher layers, constructs an APPLICATION_PRIVATE_DATA message (section 3.2.5.4.15) and sends it to client using a wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the RunspacePool.

The PowerShell server changes the RunspacePool state (section 3.2.1.2.2) to Opened and sends a RUNSPACEPOOL_STATE message (section 3.2.5.4.9) with Opened state to the PowerShell client using wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to RunspacePool.

For more information on how a RunspacePool is created on the PowerShell server, see section 3.1.4.1.

3.2.5.4.3 PUBLIC_KEY Message

The syntax of this message is specified in section 2.2.2.3.

The RunspacePool MUST be in an Opened state (section 3.2.1.2.2) while processing this message.

When this message is received, the PowerShell server extracts the public key from the message and stores it in the RunspacePool's public key (section 3.2.1.2.8).

PowerShell server generates a session key (section 3.2.1.2.7), if one is not already generated, and sends the session key as part of an ENCRYPTED_SESSION_KEY message (section 3.2.5.4.4) to the

PowerShell client using a wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the RunspacePool.

3.2.5.4.4 ENCRYPTED_SESSION_KEY Message

The syntax of this message is specified in section 2.2.2.4. The **RPID** field (as specified in section 2.2.1) of this message MUST be zeroed out.

The PowerShell server MUST send this message to the client as a response to the PUBLIC_KEY message (section 3.2.5.4.3).

The PowerShell server MUST generate a session key (section 3.1.1.2.7), if one is not already generated, and send the session key as part of an ENCRYPTED_SESSION_KEY message to the PowerShell client using a wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to RunspacePool.

When this message is sent, the RunspacePool MUST be in an Opened state.

3.2.5.4.5 PUBLIC_KEY_REQUEST Message

The syntax of this message is specified in section 2.2.2.5. The **RPID** field (as specified in section 2.2.1) of this message MUST be zeroed out.

The PowerShell server MUST send this message to the PowerShell client's RunspacePool if the PowerShell server is trying to send secured data, and if the Session Key (section 3.2.1.2.7) is not available yet. See section 3.2.5.1.1 for more details.

The PowerShell server sends this message to get the PowerShell client's Public Key.

When this message is sent, the RunspacePool MUST be in an Opened state.

3.2.5.4.6 SET_MAX_RUNSPACES Message

The syntax of this message is specified in section 2.2.2.6.

The RunspacePool MUST be in an Opened state (section 3.2.1.2.2) while processing this message.

The PowerShell server MUST extract the "ci" (call id) value from the message and use it for sending a response using RUNSPACE_AVAILABILITY message (section 2.2.2.8).

In response to this message, the PowerShell server MUST update the Maximum number of Runspaces in the RunspacePool (section 3.2.1.2.9) value, unblock any pipelines blocked in the pending pipelines queue (section 3.2.1.2.11), and send a RUNSPACE_AVAILABILITY message (section 2.2.2.8) with an appropriate Boolean value using a wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to a RunspacePool.

The PowerShell server MUST NOT stop any executing pipelines because of this message.

3.2.5.4.7 SET_MIN_RUNSPACES Message

The syntax of this message is specified in section 2.2.2.7.

The RunspacePool MUST be in the Opened state (section 3.2.1.2.2) while processing this message.

The PowerShell server MUST extract the "ci" (call id) value from the message and use it for sending a response using RUNSPACE_AVAILABILITY message (section 2.2.2.8).

In response to this message, the PowerShell server MUST update the Minimum number of Runspaces in the RunspacePool (section 3.2.1.2.9) value and send a RUNSPACE_AVAILABILITY message (section 2.2.2.8) with the appropriate Boolean value using the wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the RunspacePool.

3.2.5.4.8 RUNSPACE_AVAILABILITY Message

The syntax of this message is specified in section 3.1.5.4.8.

The PowerShell server MUST send this message as a response to SET_MAX_RUNSPACES message (section 2.2.2.6), SET_MIN_RUNSPACES message (section 2.2.2.7), or GET_AVAILABLE_RUNSPACES message (section 2.2.2.11) using a wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to a RunspacePool.

When this message is sent, the RunspacePool MUST be in an Opened state.

While constructing this message, the PowerShell server MUST extract the "ci" (call id) value from the corresponding SET_MAX_RUNSPACES message (section 2.2.2.6), SET_MIN_RUNSPACES message (section 2.2.2.7) or GET_AVAILABLE_RUNSPACES message (section 2.2.2.11) and use the same value in the "ci" portion of the message.

3.2.5.4.9 RUNSPACEPOOL_STATE Message

The syntax of this message is specified in section 2.2.2.9.

This message MUST be sent when the RunspacePool's state (section 3.2.1.2.2) changes to Opened or Broken.

This message MAY be sent when the RunspacePool’s state (section 3.2.1.2.2) changes to Closed.

3.2.5.4.10 CREATE_PIPELINE Message

The syntax of this message is specified in section 2.2.2.10.

The RunspacePool MUST be in the Opened state (section 3.2.1.2.2) while processing this message.

The PowerShell client sends this message to execute a pipeline on the server.

The PowerShell server extracts the PID from the message (section 2.2.1) and stores it as the GUID (section 3.2.1.3.1) of the pipeline.

The PowerShell client MAY send a pipeline object (section 2.2.3.11) as multiple fragments. If this is the case, the PowerShell client will send the first fragment using a wxf:Command message (section 3.1.5.3.3) and the rest of the fragments using a wxf:Send message (section 3.1.5.3.5). The PowerShell server MUST collect all the fragments, construct the PowerShell message using the rules described in section 3.2.5.1.2 and only then start executing the pipeline.

Before executing the pipeline, the PowerShell server initializes the pipeline state (section 3.2.1.3.2) to Running.

If a Runspace in the RunspacePool is available (section 3.2.1.2.10), the RunspacePool assigns one of the free Runspaces to the pipeline for execution. If a Runspace is not available, the RunspacePool adds the pipeline to the pending pipelines queue (section 3.2.1.2.11). When a Runspace in the RunspacePool is free, the RunspacePool picks up the first pipeline from the pipelines queue and invokes the pipeline after setting the state of the runspace to Busy (section 3.2.1.4.1) and storing the key associated with this pipeline (section 3.2.1.4.2).

3.2.5.4.11 GET_AVAILABLE_RUNSPACES Message

The syntax of this message is specified in section 2.2.2.11.

The RunspacePool MUST be in an Opened state (section 3.2.1.2.2) while processing this message.

The PowerShell server MUST extract the "ci" (call id) value from the message and use it for sending a response using the RUNSPACE_AVAILABILITY message (section 2.2.2.8).

In response to this message, the PowerShell server MUST get the available number of Runspaces in the RunspacePool (section 3.2.1.2.10) and sends a RUNSPACE_AVAILABILITY message (section 2.2.2.8) with appropriate integer value using wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the RunspacePool.

3.2.5.4.12 USER_EVENT Message

The syntax of this message is specified in section 2.2.2.12. Note that the PowerShell Remoting Protocol does not generate or interpret any events; it merely provides a mechanism for higher layers on the PowerShell client to be notified when new events are reported by the PowerShell server.

The RunspacePool MUST be in the Opened state (section 3.2.1.2.2) while sending this message.

The PowerShell server sends this message to notify a PowerShell client about a higher-layer server-side event.

3.2.5.4.13 APPLICATION_PRIVATE_DATA Message

The syntax of this message is specified in section 2.2.2.13.

This message MUST be sent to a PowerShell client when the RunspacePool state (section 3.2.1.2.2) is NegotiationSucceeded and the PowerShell server receives an INIT_RUNSPACEPOOL message (section 3.2.5.4.2) from the PowerShell client.

The PowerShell server sends this message to notify a PowerShell client about the server-side higher-layer specific application data.

3.2.5.4.14 GET_COMMAND_METADATA Message

The syntax of this message is specified in section 2.2.2.14.

While sending responses to the client, the server MUST use the same PowerShell messages that are used for the pipeline: PIPELINE_OUTPUT (section 3.2.5.4.19), ERROR_RECORD (section 3.2.5.4.20), DEBUG_RECORD (section 3.2.5.4.22), VERBOSE_RECORD (section 3.2.5.4.23), WARNING_RECORD (section 3.2.5.4.24), and PROGRESS_RECORD (section 3.2.5.4.25).

The server SHOULD perform the following steps upon receiving this message:

1. Extract the PID from the message (section 2.2.1) and use the same PID while sending responses back to the client.

2. The server MUST create a collection of command metadata which SHOULD be populated by collecting the available commands metadata in the RunspacePool from the higher layer by extracting the extended properties Name, CommandType, Namespace and ArgumentList from the GET_COMMAND_METADATA message (section 2.2.2.14) and passing them to the higher layer.

3. Once all the commands metadata is collected, the server MUST first construct a CommandMetadataCount (section 2.2.3.21) object using the collected number of commands metadata and send it to the client using the PIPELINE_OUTPUT message (section 3.2.5.4.19). For each and every command metadata in the collection, the server MUST construct a CommandMetadata (section 2.2.3.22) object and send it to client using the PIPELINE_OUTPUT message (section 3.2.5.4.19).

4. Once all the CommandMetadata (section 2.2.3.22) objects are sent, the server MUST send a PIPELINE_STATE message (section 3.2.5.4.21) with Completed state to the client.

If, for any reason, the server has to close a RunspacePool in the middle of performing these steps, the server SHOULD send a PIPELINE_STATE message (section 3.2.5.4.21) with Stopped state to the client and stop performing the next steps.

3.2.5.4.15 RUNSPACEPOOL_HOST_CALL Message

The syntax of this message is specified in section 2.2.2.15.

The RunspacePool MUST be in an Opened or NegotiationSucceeded state (section 3.2.1.2.2) while sending this message to a PowerShell client.

If a response is expected from the Host method call, the PowerShell server MUST construct a unique integer identifier to represent the message and store it in the Host calls CI table (section 3.2.1.2.6). The PowerShell server constructs the message using the integer identifier and MUST send the message using a wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the RunspacePool.

The PowerShell server sends this message to make a method call on the PowerShell client's Host.

3.2.5.4.16 RUNSPACEPOOL_HOST_RESPONSE Message

The syntax of this message is specified in section 2.2.2.16.

This message will be received by the PowerShell server when the RunspacePool state (section 3.2.1.2.2) is Opened or NegotiationSucceeded and a RUNSPACEPOOL_HOST_CALL message (section 3.2.5.4.15) has been sent.

The PowerShell server SHOULD extract the "ci" (call id) from the message and remove the corresponding integer identifier from the Host calls CI table (section 3.2.1.2.6).

3.2.5.4.17 PIPELINE_INPUT Message

The syntax of this message is specified in section 2.2.2.17.

This message is targeted to a pipeline. While processing this message, the pipeline MUST be in a Running state (section 3.2.1.2.2) and a successful response to wxf:Command (section 3.2.5.3.4) MUST already be sent.

The PowerShell server MUST process the message and send the contents as input to the pipeline executing in the higher layer on the server.

3.2.5.4.18 END_OF_PIPELINE_INPUT Message

The syntax of this message is specified in section 2.2.2.18.

This message is targeted to a pipeline. While processing this message, the pipeline MUST be in a Running state (section 3.2.1.3.2).

This message signifies that the PowerShell client is not sending any more input to the pipeline after this message.

3.2.5.4.19 PIPELINE_OUTPUT Message

The syntax of this message is specified in section 2.2.2.19.

The pipeline MUST be in a Running state (section 3.2.1.3.2) when this message is sent.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's Output data.

3.2.5.4.20 ERROR_RECORD Message

The syntax of this message is specified in section 2.2.2.20.

The pipeline MUST be in a Running state (section 3.2.1.3.2) when this message is sent.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's Error data.

3.2.5.4.21 PIPELINE_STATE Message

The syntax of this message is specified in section 2.2.2.21.

The PowerShell server MUST send this message whenever pipeline state (section 3.2.1.3.2) reaches Completed or Failed or Stopped. At the same time, the server must set the state of the runspace to Available (section 3.2.1.4.1) and clear the pipeline associated with the runspace (section 3.2.1.4.2).

The PowerShell server sends this message to notify a PowerShell client about a pipeline's state.

If the pipeline state (section 3.2.1.3.2) is Completed or Failed or Stopped, the PowerShell server MUST not send any PowerShell Remoting Protocol layer messages using wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the pipeline after sending the PIPELINE_STATE message (section 2.2.2.21.

3.2.5.4.22 DEBUG_RECORD Message

The syntax of this message is specified in section 2.2.2.22.

The pipeline MUST be in a Running state (section 3.2.1.3.2) when this message is sent.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's Debug data.

3.2.5.4.23 VERBOSE_RECORD Message

The syntax of this message is specified in section 2.2.2.23.

The pipeline MUST be in a Running state (section 3.2.1.3.2) when this message is sent.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's Verbose data.

3.2.5.4.24 WARNING_RECORD Message

The syntax of this message is specified in section 2.2.2.24.

The pipeline MUST be in a Running state (section 3.2.1.3.2) when this message is sent.

The PowerShell server sends this message to notify a PowerShell client about the pipeline's Warning data.

3.2.5.4.25 PROGRESS_RECORD Message

The syntax of this message is specified in section 2.2.2.25.

The pipeline MUST be in a Running state (section 3.2.1.3.2) when this message is sent.

The PowerShell server sends this message to notify a PowerShell client about a pipeline's Warning data.

3.2.5.4.26 PIPELINE_HOST_CALL Message

The syntax of this message is specified in section 2.2.2.26.

The pipeline MUST be in Running state (section 3.2.1.3.2) when this message is sent.

If a response is expected from the Host method call, the server MUST construct a unique integer identifier to represent the message, to be sent, and store it in the Host calls CI table (section 3.2.1.3.6). The PowerShell server constructs the message using the integer identifier and MUST send the message using wxf:ReceiveResponse message (section 3.2.5.3.8) targeted to the pipeline.

The PowerShell server sends this message to make a method call on a PowerShell client's Host.

3.2.5.4.27 PIPELINE_HOST_RESPONSE Message

The syntax of this message is specified in section 2.2.2.27.

This message will be received by a PowerShell server when the pipeline state (section 3.2.1.3.2) is running and a PIPELINE_HOST_CALL message (section 3.2.5.4.26) is sent.

This message will be sent to a PowerShell server's pipeline using a wxf:Send message (section 3.1.5.3.5) targeted to the pipeline.

The PowerShell server MUST extract the "ci" (call id) from the message and remove the corresponding integer identifier from the Host calls CI table (section 3.2.1.3.6).

3.2.5.4.28 CONNECT_RUNSPACEPOOL Message

The syntax of this message is specified in section 2.2.2.28.

When this message is processed, the RunspacePool MUST be in the NegotiationSucceeded state (see section 3.2.1.2.2). The PowerShell server gathers application private data from higher layers, constructs an APPLICATION_PRIVATE_DATA message (see section 3.2.5.4.15), and sends the APPLICATION_PRIVATE_DATA message to the client using a wxf:ReceiveResponse message (see section 3.2.5.3.8) targeted to the RunspacePool.

The PowerShell server changes the RunspacePool state to Opened and sends a RUNSPACEPOOL_INIT_DATA message (section 3.2.5.4.29) to the PowerShell client using a wxf:ConnectResponse message (see section 3.2.5.3.8) targeted to RunspacePool.

For more information on how a RunspacePool is connected to on the PowerShell server, see section 3.1.4.10.

3.2.5.4.29 RUNSPACEPOOL_INIT_DATA Message

The syntax of this message is specified in section 2.2.2.29.

The PowerShell server MUST send this message as a response to a CONNECT_RUNSPACEPOOL message (see section 2.2.2.28) targeted to a RunspacePool. When this message is sent, the RunspacePool MUST be in the Opened state.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

The PowerShell server SHOULD provide a mechanism that the higher-layer can use to notify the PowerShell server about higher-layer events. When the PowerShell server receives a higher-layer event notification from the higher-layer, it MUST send an USER_EVENT message (section 2.2.2.12) to the PowerShell client.

4 Protocol Examples

4.1 Sequence Diagrams

4.1.1 Creating a RunspacePool

The typical sequence, with respect to the PowerShell remoting protocol, for creating a successful RunspacePool on the PowerShell server is shown in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The PowerShell client initializes the RunspacePool state (section 3.1.1.2) to Opening. The PowerShell client connects with the PowerShell server using wxf:Create message (section 3.1.5.3.3).	>	The PowerShell server processes the message and validates the ProtocolVersion, InputStreams and OutputStreams specified in the message (section 3.1.5.3.3). The PowerShell server adds an entry in the RunspacePool table to map the [MS-WSMV] shell to the RunspacePool (section 3.2.1.1.1). The PowerShell server initializes the RunspacePool state (section 3.2.1.2.2) to BeforeOpen.
2		<	The PowerShell server sends a success message (section 3.2.5.3.2) if validation is successful.
3	The PowerShell client sends a Session Capability using wxf:Send message (section 3.1.5.3.5). The PowerShell client sends a wxf:Receive message (section 3.1.5.3.7) to the PowerShell server to start receiving data from the PowerShell server. After each received wxf:ReceiveResponse message (section 3.2.5.3.8), the PowerShell client sends another wxf:Receive message until the RunspacePool is not in a Closed or Broken state. The PowerShell client changes the RunspacePool state (section 3.1.1.2) to NegotiationSent.	>	The PowerShell server processes the message and validates the data as per the negotiation algorithm (section 3.2.5.4.1). The PowerShell server extracts the RPID from the message (section 2.2.1) and stores this value as RunspacePool's GUID (section 3.2.1.2.7). The PowerShell server changes the RunspacePool state (section 3.2.1.2.2) to NegotiationSucceeded.
4		<	The PowerShell server sends its Session Capability (section 3.2.5.4.1) using wxf:ReceiveResponse (section 3.2.5.3.8).
5	The PowerShell client processes the PowerShell server's Session Capability object and if successful (as described in section 3.2.5.4.1): The PowerShell client changes the	>	The PowerShell server processes the INIT_RUNSPACEPOOL message.

Step	PowerShell client	Direction	PowerShell server
	RunspacePool state (section 3.1.1.2) to NegotiationSucceeded. The PowerShell client sends a INIT_RUNSPACEPOOL message (section 2.2.2.2) using wxf:Send message (section 3.1.5.3.5).		
6		<	The PowerShell server sends ApplicationPrivateData (section 3.2.5.4.13) using wxf:ReceiveResponse (section 3.2.5.3.8).
7	The PowerShell client receives ApplicationPrivateData and hands it over to higher layers.		
8		<	The PowerShell server changes the RunspacePool state (section 3.2.1.2.2) to Opened. The PowerShell server constructs the Opened RUNSPACEPOOL_STATE message (section 2.2.2.9) and sends it to the PowerShell client using wxf:ReceiveResponse (section 3.2.5.3.8).
8	The PowerShell client changes the RunspacePool state (section 3.1.1.2) to Opened.		

4.1.2 Connecting to a RunspacePool

The typical PowerShell Remoting Protocol sequence for successfully connecting to an existing RunspacePool on the PowerShell server is shown in the following table.

Step	PowerShell client	Direction	PowerShell server
1	The PowerShell client initializes the RunspacePool state to Connecting (see section 3.2.1.2.2). The PowerShell client connects with the PowerShell server using a wxf:Connect message (see section 3.1.5.3.14) that includes the SESSION_CAPABILITY and CONNECT_RUNSPACEPOOL messages.	>	The PowerShell server processes the message and validates the ProtocolVersion (see section 3.1.5.3.14).The PowerShell server processes the CONNECT_RUNSPACEPOOL message.The PowerShell server sets the RunspacePool state (section 3.2.1.2.2) to Connecting.
2		<	The PowerShell server sends a success message (see section 3.2.5.3.15) along with the server's negotiation information and the RUNSPACEPOOL_INIT_DATA message, if validation succeeds.

Step	PowerShell client	Direction	PowerShell server
3	The PowerShell client processes the PowerShell server's Session Capability object and, if successful (as described in section 3.2.5.4.1), takes the following actions: ▪ The PowerShell client changes the RunspacePool state (see section 3.2.1.2.2) to NegotiationSucceeded. ▪ The PowerShell client processes the RUNSPACEPOOL_INIT_DATA message and changes the RunspacePool to Opened. ▪ The PowerShell client issues a wxf:Receive request to the server.	>	
4		<	The PowerShell server uses a wxf:ReceiveResponse message (see section 3.2.5.3.8) to send an APPLICATION_PRIVATE_DATA message (see section 3.2.5.4.13). The PowerShell server changes the RunspacePool state to Opened.
5	The PowerShell client receives the APPLICATION_PRIVATE_DATA message and passes it to the higher layer.		

4.1.3 Creating and Invoking a Pipeline

The typical sequence, with respect to the PowerShell remoting protocol, for creating and invoking a successful pipeline on the PowerShell server is shown in the following table:

Step	PowerShell Client	Direction	PowerShell server
1	The RunspacePool MUST be in the Opened state on the PowerShell client (section 4.2.1). The PowerShell client constructs a CREATE_PIPELINE message (section 2.2.2.10). The PowerShell client Fragments the message into multiple fragments as needed (section 2.2.4). The PowerShell client initializes the pipeline state (section 3.1.1.3.2) to Running. The PowerShell client sends the first fragment to the PowerShell server using wxf:Command message (section 3.1.5.3.3).	>	The PowerShell server extracts the PID from the message (section 2.2.1) and stores this value as pipeline's GUID (section 3.2.1.3.1). The PowerShell server initializes the pipeline state (section 3.2.1.3.2) to NotStarted. The PowerShell server processes and validates the message.

Step	PowerShell Client	Direction	PowerShell server
2		<	The PowerShell server sends a success message (section 3.2.5.3.4) if validation is successful.
3	If the pipeline message is fragmented into multiple fragments, then rest of the fragments (starting from second fragment) are sent individually using wxf:Send message (section 3.1.5.3.5).	>	
		<	The PowerShell server collects all the fragments until the end fragment (section 2.2.4) is received. The each wxf:Send message received from the client, the PowerShell server sends a wxf:SendResponse message (see section 3.2.5.3.6) to the client. The PowerShell server processes all the fragments and understands the pipeline to execute. If a runspace in the RunspacePool is available (section 3.2.1.2.10), the RunspacePool MUST assign the runspace to the pipeline for execution.
4		<	The PowerShell server changes the pipeline state (section 3.2.1.3.2) to Running. Note this state change information is not sent to the PowerShell client. The PowerShell server starts executing the pipeline.
5	The PowerShell client sends a wxf:Receive message to start receiving data from the pipeline on the PowerShell server. After each received wxf:ReceiveResponse message (see section 3.2.5.3.8), the PowerShell client sends another wxf:Receive message until the server indicates that the pipeline is completed (step 8). Sending of input (steps 6 and 7) can happen in parallel.	>	
		<	The PowerShell server sends the pipeline result messages (if any): PIPELINE_OUTPUT (section 2.2.2.19), ERROR_RECORD (section 2.2.2.20), DEBUG_RECORD (section 2.2.2.22), VERBOSE_RECORD (section 2.2.2.23), WARNING_RECORD (section 2.2.2.24) and PROGRESS_RECORD (section 2.2.2.25) using wxf:ReceiveResponse (section 3.2.5.3.8).
6	The PowerShell client MAY send any	>	The PowerShell server processes the

Step	PowerShell Client	Direction	PowerShell server
	PIPELINE_INPUT messages (section 2.2.2.17) to the pipeline using wxf:Send message (section 3.1.5.3.5) if the CREATE_PIPELINE message indicated that the pipeline takes input.		message and dispatches the input to pipeline execution.
7	If the CREATE_PIPELINE message indicated that the pipeline takes input, then the client MUST send an END_OF_PIPELINE_INPUT message (section 2.2.2.18) using wxf:Send message (section 3.1.5.3.5) after sending all (possibly zero) the PIPELINE_INPUT messages to the server.	>	The PowerShell server processes the END_OF_PIPELINE_INPUT message and notifies the pipeline execution that no more Input is expected.
8		<	Once the pipeline execution is complete: ▪ The PowerShell server removes the entry for this pipeline in the RunspacePool's pending pipelines queue (section 3.2.1.2.11). ▪ The PowerShell server changes the pipeline state (section 3.2.1.3.2) to Completed. ▪ The PowerShell server constructs the Completed PIPELINE_STATE message (section 2.2.2.21) and sends it to the PowerShell client using wxf:ReceiveResponse (section 3.2.5.3.8).
9	The PowerShell client changes the pipeline state (section 3.1.1.3.2) to Completed.		

4.1.4 Stopping a Pipeline

The typical sequence, with respect to the PowerShell remoting protocol, for stopping a running pipeline on the PowerShell server is shown in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The PowerShell client MUST construct a pipeline and that pipeline MUST be in the Running state. The PowerShell client MUST receive a success message for the wxf:Command message (section 3.1.5.3.3). The PowerShell client sends a wxf:Receive message (if the pipeline currently has no pending wxf:Receive messages) to start receiving data from the pipeline on the PowerShell	>	

Step	PowerShell client	Direction	PowerShell server
	server.		
2	The PowerShell client changes the pipeline state (section 3.1.1.3.2) to Stopping. The PowerShell client sends a wxf:Signal message (section 3.1.5.3.9) to stop the pipeline on the PowerShell server.	>	The PowerShell server changes the pipeline state (section 3.2.1.3.2) to Stopping. The PowerShell server SHOULD stop the currently executing pipeline.
3		<	The PowerShell server removes the entry for this pipeline in the RunspacePool's pending pipelines queue (section 3.2.1.2.11). The PowerShell server changes the pipeline state (section 3.2.1.3.2) to Stopped. The PowerShell server constructs the Stopped PIPELINE_STATE message (section 2.2.2.21) and sends it to the PowerShell client using wxf:ReceiveResponse (section 3.2.5.3.8).
4	The PowerShell client changes the pipeline state (section 3.1.1.3.2) to Stopped.	<	The PowerShell server sends a success message for wxf:Signal (section 3.2.5.3.10).

4.1.5 Client-Initiated Transfer of Session Key

The PowerShell Remoting Protocol allows the PowerShell client and the PowerShell server to exchange a session key (section 2.2.2.4). The typical sequence, with respect to the PowerShell remoting protocol, for transferring a session key (section 2.2.2.4) from the PowerShell server to the PowerShell client, when the PowerShell client initiates the transfer, is described in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The RunspacePool MUST be in the Opened state on the PowerShell client.		
2	The PowerShell client constructs a PUBLIC_KEY message (section 2.2.2.3) and sends it using a wxf:Send message (section 3.1.5.3.5) targeted to the RunspacePool. The PowerShell client starts Session Key Transfer timer (section 3.1.2).	>	The PowerShell server stores the Public Key (section 3.2.1.2.8). The PowerShell server generates a Session Key (section 3.2.1.2.7), if not already generated.
3	The PowerShell client sends a wxf:Receive message (see section 3.1.5.3.7) to the PowerShell server, if none is pending for this RunspacePool.	>	
4		<	For each wxf:Send message received from the client, the PowerShell server sends a wxf:SendResponse message (see

Step	PowerShell client	Direction	PowerShell server
			section 3.2.5.3.6) to the client.
5	The PowerShell client processes the ENCRYPTED_SESSION_KEY message (section 2.2.2.4), cancels the Session Key Transfer timer (section 3.1.6) and stores the Session Key (section 3.1.1.2.7) for future use.	<	The PowerShell server constructs an Encrypted Session Key (section 2.2.2.4) and sends it to the PowerShell client using wxf:ReceiveResponse (section 3.2.5.3.8).
6	From this point on, the PowerShell client uses the stored Session Key (section 3.1.1.2.7) for sending secure data (section 2.2.5.1.24) to the PowerShell server.		From this point on, the PowerShell server uses the Session Key (section 3.1.1.2.7) for sending secure data (section 2.2.5.1.24) to the PowerShell client.

4.1.6 Server-Initiated Transfer of Session Key

The PowerShell remoting protocol allows the PowerShell client and the PowerShell server to exchange a session key (section 2.2.2.4). The typical sequence, with respect to the PowerShell remoting protocol, for transferring a session key (section 2.2.2.4) from the PowerShell server to the PowerShell client, when the PowerShell server initiates the transfer, is described in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The RunspacePool MUST be in the Opened state on the PowerShell client.		The RunspacePool MUST be in the Opened state on the PowerShell server (section 4.1.1). The Public Key (section 3.2.1.2.8) MUST be empty.
2	The PowerShell client sends a wxf:Receive message (section 3.1.5.3.7) to the PowerShell server, if none is pending for this RunspacePool.	>	
3		<	The PowerShell server constructs a PUBLIC_KEY_REQUEST message (section 2.2.2.5) and sends it to the PowerShell client using wxf:ReceiveResponse (section 3.2.5.3.8).
4	The PowerShell client constructs a PUBLIC_KEY message (section 2.2.2.3) and sends it using wxf:Send message (section 3.1.5.3.5) targeted to the RunspacePool. The PowerShell client starts Session Key Transfer timer (section 3.1.2).	>	The PowerShell server stores the Public Key (section 3.2.1.2.8). The PowerShell server generates a Session Key (section 3.2.1.2.7), if not already generated.
5	The PowerShell client sends a wxf:Receive message (section 3.1.5.3.7) to the PowerShell server, if none is pending for this RunspacePool.	>	

Step	PowerShell client	Direction	PowerShell server
6		<	For each wxf:Send message received from the client, the PowerShell server sends a wxf:SendResponse message (see section 3.2.5.3.6) to the client.
7	The PowerShell client processes the ENCRYPTED_SESSION_KEY message (section 2.2.2.4), cancels the Session Key Transfer timer (section 3.1.2) and stores the Session Key (section 3.1.1.2.7) for future use.	<	The PowerShell server constructs an Encrypted Session Key (section 2.2.2.4) and sends it to the PowerShell client using wxf:ReceiveResponse (section 3.2.5.3.8).
8	From this point on, the PowerShell client uses the stored Session Key (section 3.1.1.2.7) for sending secure data (section 2.2.5.1.24) to the PowerShell server.		From this point on, the PowerShell server uses the Session Key (section 3.2.1.2.7) for sending secure data (section 2.2.5.1.24) to the PowerShell client.

4.1.7 Changing Maximum Runspaces Count of the Server's RunspacePool

The typical sequence, with respect to the PowerShell Remoting Protocol, for changing the maximum Runspaces count of the PowerShell server's RunspacePool is described in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The RunspacePool MUST be in the Opened state on the PowerShell client.		
2	The PowerShell client constructs an integer identifier to represent the message, to be sent, and stores it in the RunspacePool's CI table (section 3.1.1.2.5). The PowerShell client constructs a SET_MAX_RUNSPACES message (section 2.2.2.6) and sends it using a wxf:Send message (section 3.1.5.3.5) targeted to the RunspacePool.	>	The PowerShell server changes the Maximum number of runspaces as per the guidelines specified in section 3.2.1.2.9.
3	The PowerShell client sends a wxf:Receive message (see section 3.1.5.3.7) to the PowerShell server, if none is pending for this RunspacePool, to start receiving data from the PowerShell server.	>	
4	The PowerShell client processes the RUNSPACE_AVAILABILITY message (section 2.2.2.8) and removes the corresponding entry from the RunspacePool's CI table (section 3.1.1.2.5).	<	The PowerShell server sends a wxf:SendResponse message (see section 3.2.5.3.6) to the client. The PowerShell server constructs a RUNSPACE_AVAILABILITY message (section 2.2.2.8) and sends it to the PowerShell client using a wxf:ReceiveResponse message (section

Step	PowerShell client	Direction	PowerShell server
			3.2.5.3.8).
5	The PowerShell client MAY send the response to the higher layer as required.		

4.1.8 Changing Minimum Runspaces Count of the Server's RunspacePool

The typical sequence, with respect to the PowerShell Remoting Protocol, for changing the minimum runspaces count of the PowerShell server's RunspacePool is described in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The RunspacePool MUST be in the Opened state on the PowerShell client.		
2	The PowerShell client constructs an integer identifier to represent the message, to be sent, and stores it in the RunspacePool's CI table (section 3.1.1.2.5). The PowerShell client constructs a SET_MIN_RUNSPACES message (section 2.2.2.7) and sends it using wxf:Send message (section 3.1.5.3.5) targeted to the RunspacePool.	>	The PowerShell server changes the Minimum number of runspaces as per the guidelines specified in section 3.2.1.2.9.
3	The PowerShell client sends a wxf:Receive message (see section 3.1.5.3.7) to the PowerShell server, if none is pending for this RunspacePool.	>	
4	The PowerShell client processes the RUNSPACE_AVAILABILITY message (section 2.2.2.8) and removes the corresponding entry from the RunspacePool's CI table (section 3.1.1.2.5).	<	The PowerShell server sends a wxf:SendResponse message (section 3.2.5.3.6) to the client. The PowerShell server constructs a RUNSPACE_AVAILABILITY message (section 2.2.2.8) and sends it to the PowerShell client using a wxf:ReceiveResponse message (section 3.2.5.3.8).
5	The PowerShell client MAY send the response to the higher layer as required.		

4.1.9 Getting Available Runspaces of the Server's RunspacePool

The typical sequence, with respect to the PowerShell Remoting Protocol for getting the available Runspaces count of the PowerShell server's RunspacePool is described in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The RunspacePool MUST be in the Opened state on the PowerShell client.		

Step	PowerShell client	Direction	PowerShell server
2	The PowerShell client constructs an integer identifier to represent the message to be sent, and stores it in the RunspacePool's CI table (section 3.1.1.2.5). The PowerShell client constructs a GET_AVAILABLE_RUNSPACES message (section 2.2.2.11) and sends it using wxf:Send message (section 3.1.5.3.11) targeted to the RunspacePool.	>	
3	The PowerShell client sends a wxf:Receive message (section 3.1.5.3.7) to the PowerShell server if none is currently pending for this RunspacePool.	>	
4	The PowerShell client processes the RUNSPACE_AVAILABILITY message (section 2.2.2.8) and removes the corresponding entry from the RunspacePool's CI table (section 3.1.1.2.5).	<	The PowerShell server gets the available number of runspaces (section 3.2.1.2.10). The PowerShell server sends a wxf:SendResponse message (section 3.2.5.3.6) to the client. The PowerShell server constructs a RUNSPACE_AVAILABILITY message (section 2.2.2.8) and sends it to the PowerShell client using wxf:ReceiveResponse message (section 3.2.5.3.8).
5	The PowerShell client MAY send the response to the higher-layer as needed.		

4.1.10 Host method calls targeted to Client's Pipeline

The PowerShell Remoting Protocol allows a PowerShell server to initiate method calls on the PowerShell client's host. The typical sequence, with respect to the PowerShell remoting protocol, for initiating the host method call is described in the following table:

Step	PowerShell client	Direction	PowerShell server
1			The PowerShell server's pipeline MUST be in the Running state.
2	The PowerShell client processes the Host Method call associated with the PIPELINE_HOST_CALL message (section 2.2.2.26) and extracts the method to execute (section 2.2.3.17). The PowerShell client hands over the message to the higher layer for further processing.	<	If a response is expected from the Host method call, the PowerShell server MUST construct an integer identifier to represent the message, to be sent, and stores it in the Host calls CI table (section 3.2.1.2.6). The PowerShell server constructs a PIPELINE_HOST_CALL message (section 2.2.2.26) and sends it to the PowerShell client using wxf:ReceiveResponse message

Step	PowerShell client	Direction	PowerShell server
			(section 3.2.5.3.8). If a response is expected from the Host method call, the PowerShell server MUST pause executing the pipeline until a response for the Host method is received.
3	If a response (or return value) is expected from the Host method (section 2.2.3.17), the PowerShell client MUST construct a PIPELINE_HOST_RESPONSE message (section 3.2.5.4.27) and send it to the PowerShell server using wxf:Send message (section 3.1.5.3.5) targeted to the pipeline.	>	The PowerShell server processes the message and removes the corresponding entry from the Host calls CI table (section 3.2.1.2.6). The PowerShell server extracts the response portion of the PIPELINE_HOST_RESPONSE message, hands it over to the pipeline and resumes executing the pipeline.

If a response is not expected from the Host Method call, then:

- The CI table is not updated on the server in Step 2.
- The server does not pause the execution of the pipeline in Step 2.
- Step 3 is skipped.

4.1.11 Getting the Metadata of Remote Commands

The typical sequence, with respect to the PowerShell Remoting Protocol, for getting the metadata of commands available on the PowerShell server is shown in the following table:

Step	PowerShell client	Direction	PowerShell server
1	The RunspacePool MUST be in the Opened state on the client (section 4.2.1). The PowerShell client constructs a GET_COMMAND_METADATA message (section 2.2.2.14). The PowerShell client fragments the message into multiple fragments as needed (section 2.2.4). The PowerShell client initializes the pipeline state (section 3.1.1.3.2) to Running. The PowerShell client sends the first fragment to the server using the wxf:Command message (section 3.1.5.3.3).	>	The PowerShell server extracts the RPID and PID from the message (section 2.2.1) and uses the same values while sending responses to the PowerShell client.
2		<	The PowerShell server sends a success message (section 3.2.5.3.4) if validation is successful.
3	If the message is fragmented into multiple fragments, then the rest of	>	The PowerShell server collects all the fragments until the end fragment (section

Step	PowerShell client	Direction	PowerShell server
	the fragments (starting from the second fragment) are sent individually using the wxf:Send message (section 3.1.5.3.5).		2.2.4) is received. The PowerShell server validates the GET_COMMAND_METADATA message (section 3.2.5.4.14).
4			The PowerShell server collects the available commands metadata in the RunspacePool from the higher layer. While interacting with the higher layer, the PowerShell server extracts the extended properties Name, CommandType, Namespace and ArgumentList from the GET_COMMAND_METADATA message (section 2.2.2.14) message and passes them to the higher layer. The higher layer SHOULD interpret the values of these properties as per the guidelines specified in section 2.2.2.14.
5	The PowerShell client sends a wxf:Receive message to start receiving data from the server.	>	
6		<	Once all the commands metadata is collected, the PowerShell server first constructs a CommandMetadataCount (section 2.2.3.21) object using the collected number of commands metadata and sends it to the PowerShell client. For each and every command metadata collected, the PowerShell server constructs a CommandMetadata (section 2.2.3.22) object and sends it to the PowerShell client.
7	The PowerShell client sends a wxf:Receive message to start receiving data from the server. This step is repeated until server indicates that the Getting command metadata is completed (step 8).	>	
8		<	Once all CommandMetadata (see section 2.2.3.22) objects have been sent, the PowerShell server sends the Completed PIPELINE_STATE message (section 3.2.5.4.21).
9	The PowerShell client changes the pipeline state (section 3.1.1.3.2) to Completed and notifies the higher layer.		

4.2 Transport Message Examples

The following examples show how to represent transport-specific data blocks.

ObjectId: A value of 1 is represented as follows.

```
Byte 0: 0
Byte 1: 0
Byte 2: 0
Byte 3: 0
Byte 4: 0
Byte 5: 0
Byte 6: 0
Byte 7: 1
```

FragmentId: A value of 1 is represented as follows.

```
Byte 0: 0
Byte 1: 0
Byte 2: 0
Byte 3: 0
Byte 4: 0
Byte 5: 0
Byte 6: 0
Byte 7: 1
```

5 Security

PSRP clients should provide reasonable security when working with potentially malicious servers. In particular:

- If host method calls result in interaction with a user, then the implementation of the client should inform the user that the interaction (for example, a request for credentials) originated from a remote server.
- If host method calls (for example, calls to the **GetBufferContents** method) can result in an unintended information disclosure, then it may be better to return an exception ("me" property) rather than return the actual data ("mr" property).

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 3.1.5.3.1: Windows implementations specify the following value:

```
http://schemas.microsoft.com/PowerShell/Microsoft.PowerShell
```

<2> Section 3.1.5.3.1: A typical Windows implementations will specify the following values:

```
Proto -> http or https
Port -> 5985 (when Proto is http) or 5986 (when proto is https)
ApplicationName -> WSMan
```

<3> Section 3.1.5.3.1: A typical Windows implementations will specify a value of 240000.

<4> Section 3.1.5.3.3: Windows implementations specify the following value: "http://schemas.microsoft.com/PowerShell/Microsoft.PowerShell"

<5> Section 3.1.5.3.14: Windows implementations specify the following value:

"http://schemas.microsoft.com/PowerShell/Microsoft.PowerShell"

<6> Section 3.1.5.3.14: A typical Windows implementation supplies the applicationname "WSMan", and uses port number of 5985 when the protocol is "http" or port number of 5986 when the protocol is "https".

<7> Section 3.1.5.3.16: Implementations on Windows use the resource URI "http://schemas.microsoft.com/PowerShell/Microsoft.PowerShell".

<8> Section 3.1.5.3.16: Implementations on Windows specify the value 240000.

<9> Section 3.1.5.3.18: Implementations on Windows use the resource URI "http://schemas.microsoft.com/PowerShell/Microsoft.PowerShell".

<10> Section 3.2.5.3.2: In Windows default implementations, the value of Resource URI is "http://schemas.microsoft.com/powershell/Microsoft.PowerShell".

7 Change Tracking

This section identifies changes that were made to the [MS-PSRP] protocol document between the November 2013 and February 2014 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.

- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.3.19 CommandType	57081 Removed table of command types and values.	Y	Product behavior note removed.
2.2.3.22 CommandMetadata	57081 Removed reference to the ResolvedCommandName property name.	Y	Content updated.
3.1.5.3.15 Rules for the wxf:ConnectResponse Message	70924 Updated title to match content.	N	Content updated.

8 Index

I

K

L

M

N

O

P

R

S

T

V

W