[MS-WMSP]:
Windows Media HTTP Streaming Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.0		MCPP Milestone 2
04/03/2007	1.1		Monthly release
05/11/2007	1.2		Monthly release
06/01/2007	1.2.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
07/03/2007	1.2.2	Editorial	Revised and edited the technical content.
07/20/2007	1.2.3	Editorial	Revised and edited the technical content.
08/10/2007	1.2.4	Editorial	Revised and edited the technical content.
09/28/2007	1.3	Minor	Updated the technical content.
10/23/2007	1.4	Minor	Unused normative reference deleted.
11/30/2007	1.4.1	Editorial	Revised and edited the technical content.
01/25/2008	1.4.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This specification defines a Microsoft proprietary protocol referred to as the Windows Media HTTP Streaming Protocol. The Windows Media HTTP Streaming Protocol is a client/server–based protocol used to **stream** real-time data between the client (the receiver of **streaming** data) and server (the sender of streaming data).

This specification specifies the following:

- How messages are transported and message syntax in section 2.
- Protocol details including abstract data models, higher-layer triggered events, and message processing rules in section 3.
- Protocol examples in section 4.
- Security considerations for implementers in section 5.
- An appendix of Windows behavior in section 6.
- An index in section 7

1.1 Glossary

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

Globally Unique Identifier (GUID)
Little-Endian
Unicode

The following terms are specific to this document:

Advanced Systems Format (ASF): The file format used by Windows Media. See [ASF].

Content: Multimedia data. **Content** is always in **ASF** format. For example, a single ASF-format music file or a single ASF-format video file.

Non-Pipelined Mode: Mode of operation in which requests from the client must be sent on a TCP connection separate from the one being used by the server for **streaming content** to the client.

Pipelined Mode: Mode of operation in which requests from the client can be sent on the same TCP connection being used by the server for **streaming content** to the client.

Playlist: One or more **content** items that are **streamed** sequentially.

Session: The state maintained by the server when it is **streaming content** to a client. If a server-side **playlist** is used, the same **session** is used for all **content** in the **playlist**.

Stream: A sequence of **ASF** Data Packets (see [ASF]) that can be selected individually. For example, if a movie has an English and a Spanish soundtrack, each may be encoded in the **ASF** file as a separate **stream**. The video data would also be a separate **stream**.

Streaming: The act of transferring **content** from a sender to a receiver.

Striding: Enables fast-forward and rewind the output of a file. Files must be indexed to take advantage of these features. An index is a series of values representing positions in the file

(either presentation times, frame numbers, or SMTPE time codes) with corresponding offsets into the data section of the file for each.

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

1.2 References

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. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[ASF] Microsoft Corporation, "Advanced Systems Format Specification", December 2004, http://download.microsoft.com/download/7/9/0/790fecaa-f64a-4a5e-a430-0bccdab3f1b4/ASF_Specification.doc

If you have any trouble finding [ASF], please check here.

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol Specification", June 2007.

[MS-RTSP] Microsoft Corporation, "Real-Time Streaming Protocol (RTSP) Windows Media Extensions", July 2007.

[MS-WMLOG] Microsoft Corporation, "Windows Media Log Data Structure", July 2007.

[RFC1738] Berners-Lee, T., Masinter, L., and McCahill, M., "Uniform Resource Locators (URL)", RFC 1738, December 1994, http://www.ietf.org/rfc/rfc1738.txt

[RFC1945] Berners-Lee, T., Fielding, R., and Frystyk, H., "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996, http://www.ietf.org/rfc/rfc1945.txt

[RFC2109] Kristol, D., and Montulli, L., "HTTP State Management Mechanism", RFC 2109, February 1997, http://www.ietf.org/rfc/rfc2109.txt

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

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.ietf.org/rfc/rfc2617.txt

[RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", RFC 3629, November 2003, http://www.ietf.org/rfc/rfc3629.txt

[RFC4234] Crocker, D., Ed. and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

[RFC4559] Jaganathan, K., Zhu, L., and Brezak, J., "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006, http://www.ietf.org/rfc/rfc4559.txt

1.2.2 Informative References

[MS-MMSP] Microsoft Corporation, "Microsoft Media Server (MMS) Protocol Specification", June 2007.

[WMSSDK] Microsoft Corporation, "Windows Media Services 9.0 Series SDK", http://msdn2.microsoft.com/en-us/library/ms738748.aspx

1.3 Protocol Overview (Synopsis)

The Windows Media HTTP Streaming Protocol is used for transferring real-time multimedia data (that is, audio and video). It is a streaming protocol, which means that the protocol attempts to facilitate scenarios where the multimedia data is being transferred and rendered (that is, video displayed, audio played) simultaneously.

The protocol depends on HTTP for the transfer of all protocol messages, including the transfer of the multimedia data. In this specification, the entity that initiates the HTTP connection is referred to as the client, and the entity that responds to the HTTP connection is referred to as the server. The multimedia data flows from the server to the client.

The client can send feedback to the server in the form of HTTP-based request messages. Based on the feedback, the server may, for example, replace the currently transmitted video stream with a lower bit-rate version of the same video stream.

Unlike HTTP, which is a stateless protocol, the Windows Media HTTP Streaming Protocol does maintain state, referred to as **session** state.

It is the session state that allows a server to relate an HTTP request containing feedback information to an HTTP request used for the transfer of the multimedia data.

In its most common mode of operation, feedback from the client is sent to the server by using HTTP requests on TCP connections that are separate from the TCP connection that is used by the server to transfer the multimedia data to the client. This transfer is referred to as the normal, or **non-pipelined mode** of operation.

In another mode of operation, feedback from the client can be sent to the server by using HTTP requests on the same TCP connection that the server uses for transferring multimedia data to the client. This transfer is referred to as the **pipelined mode** of operation.

1.4 Relationship to Other Protocols

The Windows Media HTTP Streaming Protocol depends on HTTP 1.0, as specified in [RFC1945]. The pipelined mode of the protocol can only be used if the client, the server, and any intermediate HTTP proxy servers support the pipelining feature of HTTP 1.1, as specified in [RFC2616].

This protocol can be used instead of the Microsoft Media Server (MMS) Protocol as specified in [MS-MMSP]. This protocol can also be used instead of the Real-Time Streaming Protocol Extensions as

specified in [MS-RTSP]. However, it should be noted that although these two other protocols allow the multimedia data to be transmitted over either User Datagram Protocol (UDP) or TCP, the Windows Media HTTP Streaming Protocol only allows multimedia data to be transmitted over TCP (because HTTP always uses TCP).

1.5 Prerequisites/Preconditions

The Windows Media HTTP Streaming Protocol does not provide a mechanism for a client to discover the URL to the server. Thus, it is a prerequisite that the client obtain a URL to the server before this protocol can be used.

1.6 Applicability Statement

The Windows Media HTTP Streaming Protocol is suitable for streaming delivery of real-time multimedia data. The term streaming means that the data is transmitted at some fixed rate or at some rate that is related to the rate at which the data will be consumed (for example, displayed) by the receiver.

This protocol may also be appropriate if the client sends feedback to the server that may cause the multimedia data that is being transmitted to change or cause the transmission rate to change.

Furthermore, this protocol may be appropriate if the client must perform "trick-mode operations" on the multimedia data and prefers the server to execute trick modes on its behalf. The term "trick-mode operation" refers to operations like fast-forwarding and rewinding the data, pausing the transmission, or seeking a different position in the multimedia data stream.

If some, or all, of the above applies, but the multimedia data is to be transferred over UDP, it may be more appropriate to implement the Real-Time Streaming Protocol Extensions, as specified in [MS-RTSP].

If none of the above applies, it may be more appropriate to use HTTP 1.0, as specified in [RFC1945], to download the data, instead of implementing this protocol.

1.7 Versioning and Capability Negotiation

This specification covers versioning issues in the following areas:

Supported Transports: This protocol can be implemented on top of HTTP 1.0 and HTTP 1.1, as specified in section 2.1.

Protocol Versions: Clients specify the protocol version by using the User-Agent (section 2.2.1.8) header and by using the version11-enabled (section 2.2.1.4.30) token on the Pragma header. Servers specify the protocol version by using the Server (section 2.2.1.5) header.

Security and Authentication Methods: This protocol supports HTTP access authentication, as specified in HTTP 1.0 [RFC1945] section 11.

Localization: This specification does not specify any localization-dependent protocol behavior.

Capability Negotiation: This protocol performs explicit capability negotiation by using the following mechanisms:

- The features (section 2.2.1.4.8) token on the Pragma header.
- The Supported (section 2.2.1.7) header.
- The X-Accept-Authentication (section 2.2.1.9) header.

This protocol does not use operating system versioning. This is because operating systems typically include multiple-client implementations that have different capabilities. Furthermore, the client software components are frequently updated independently of the rest of the operating system. Instead, the protocol versioning mechanism relies on the version number of the Microsoft software product that is sending the request or response to be stated in the User-Agent and Server headers, respectively.

For an example of how this versioning mechanism works and how it affects which protocol messages are sent, see the description of the $M (Metadata) packet (section 2.2.3.6).

1.8 Vendor-Extensible Fields

The Windows Media HTTP Streaming Protocol uses HRESULTs as specified in [MS-ERREF]. Vendors are free to choose their own values as long as the C bit (0x20000000) is set, indicating it is a customer code.

1.9 Standards Assignments

There are no standards assignments for the Windows Media HTTP Streaming Protocol.

2 Messages

The following sections specify how Windows Media HTTP Streaming Protocol messages are transported and Windows Media HTTP Streaming Protocol message syntax.

2.1 Transport

The Windows Media HTTP Streaming Protocol uses HTTP 1.0, as specified in [RFC1945], or HTTP 1.1, as specified in [RFC2616], as the transport layer.

A TCP port has not been reserved for this protocol. TCP port 80 is commonly used because many HTTP proxy servers only forward HTTP traffic that uses port 80.

The protocol uses the Access Authentication functionality of the HTTP layer. The supported HTTP access authentication schemes are implementation-specific.<1> Clients SHOULD use the X-Accept-Authentication (section 2.2.1.9) header to specify the preferred list of authentication schemes. For more information about HTTP access authentication, see HTTP 1.0, as specified in [RFC1945], section 11.

2.2 Message Syntax

This section is organized as follows:

- Section 2.2.1 specifies the syntax for HTTP headers that are defined by this protocol.
- Section 2.2.2 specifies the types of requests that are defined by this protocol and how each request type is mapped to HTTP.
- Section 2.2.3 specifies the syntax for the binary packet format that is used in the payloads of the HTTP messages.
- Section 2.2.4 specifies the syntax of the Content Description List format.
- Section 2.2.5 specifies the syntax of the Remote Event format.

2.2.1 HTTP Header Fields

The Windows Media HTTP Streaming Protocol defines several new headers that do not exist in HTTP 1.0, as specified in [RFC1945], or HTTP 1.1, as specified in [RFC2616]. Some headers that are defined by these other specifications are further restrained by the Windows Media HTTP Streaming Protocol in how they can be used. The new headers, and the modified existing ones, are defined in this section.

Unless specified otherwise, the headers defined in this specification, and any tokens (also called tags or directives) that are used on those headers, are defined for use in both requests and responses.

If a client or server receives an HTTP header that is not defined in the subsequent sections (Cache-Control through X-StartupProfile), or if the header is not defined in the current context (for example, receiving a request-only header in a response), then the header MUST be interpreted in accordance with the HTTP protocol used, either HTTP 1.0, as specified in [RFC1945], or HTTP 1.1, as specified in [RFC2616].

If a client or server receives an HTTP header defined in the subsequent sections, and the header contains an unknown token, or the token is not defined in the current context (for example, receiving a request-only token in a response), then the token MUST be ignored.

These following sections define the syntax of the HTTP headers using the Augmented Backus-Naur Form (ABNF) syntax. ABNF syntax is defined in Augmented BNF for Syntax Specifications: ABNF, as specified in [RFC4234]. Any ABNF syntax rules that are not defined in Augmented BNF for Syntax Specifications: ABNF use the ABNF extensions that are defined in HTTP 1.0, as specified in [RFC1945], or HTTP 1.1, as specified in [RFC2616].

2.2.1.1 Cache-Control

The purpose of the Cache-Control header is to specify to clients, and any intermediate caches (for example, proxy servers), how to cache the **content**.

The syntax of the Cache-Control header is defined as follows:

```
CCdir    = "max-age" / "must-revalidate" /
           "no-store" / "private" / "proxy-revalidate" /
           "public" / " proxy-public" / "user-public" /
           "x-wms-event-subscription" / "x-wms-proxy-split" /
           "x-wms-content-size" / "x-wms-stream-type"

Cache-Control = "Cache-Control: " "no-cache" *12("," [SP] CCdir) CRLF
```

The optional directives defined by the CCdir ABNF syntax element above are only defined for use in responses to requests.<2>

Example:

```
Cache-Control: no-cache, x-wms-content-size=638066,
x-wms-event-subscription="remote-log"
```

The subsequent directives (max-age through x-wms-stream-type) are defined for the Cache-Control header when used in the response to a request.

2.2.1.1.1 max-age

This directive specifies how many seconds a cache is allowed to use the content without revalidating it with the server. For more information about max-age, including ABNF syntax, see HTTP 1.1, as specified in [RFC2616] section 14.9.

2.2.1.1.2 must-revalidate

This directive specifies that the cache MUST revalidate that the content is still fresh before streaming the content or playing it.

2.2.1.1.3 no-store

This directive specifies that the cache MUST NOT store the content on persistent storage.

2.2.1.1.4 private

This directive specifies that the content MUST NOT be shared with other users on the device on which the client software is running or by other proxy servers. The directive applies to both caches that are clients and caches that are proxy servers. If the cache is a proxy server, the proxy-public (section 2.2.1.1.7) directive overrides this directive. If the cache is not acting as a proxy server, the user-public (section 2.2.1.1.8) directive overrides this directive.

2.2.1.1.5 proxy-revalidate

This directive specifies that if the cache is a proxy server, then it MUST revalidate that the content is still fresh before streaming the content. This directive MUST be ignored by caches that are not acting as proxy servers.

2.2.1.1.6 public

This directive specifies that the content may be cached and may be shared with other users on the device on which the client software is running or shared with other proxy servers. The directive applies to both caches that are clients and caches that are proxy servers.

2.2.1.1.7 proxy-public

This directive specifies that if the cache is a proxy server, then the content MAY be shared with other clients or other proxy servers. If the cache is a proxy server, this directive overrides the private (section 2.2.1.1.4) directive.

2.2.1.1.8 user-public

This directive specifies that if the cache is not acting as a proxy server, then the content may be cached and may be shared with other users on the device on which the client software is running. If the cache is not acting as a proxy server, this directive overrides the private (section 2.2.1.1.4) directive.

2.2.1.1.9 x-wms-content-size

This directive specifies the approximate size of the content, in bytes, that will be required if the content is cached in its entirety.<3>

2.2.1.1.10 x-wms-event-subscription

This directive specifies a comma-separated list of remote event names that the server SHOULD receive. The list is enclosed in double-quotes. The SendEvent Request (section 2.2.2.9) is used to send the remote events to the server.<4>

The syntax of the directive is defined as follows:

```
log-event = ( "remote-open" / "remote-close" / "remote-log" )
Eventsub  = "x-wms-event-subscription="
            %x22 log-event *2( "," log-event ) %x22
```

2.2.1.1.11 x-wms-proxy-split

This directive indicates that the content may be "split", that is, forwarded to multiple clients in real time. This approach is typically used for "live" content for which caching is inappropriate or not allowed.<5>

2.2.1.1.12 x-wms-stream-type

This directive specifies a comma-separated list of properties that apply to the content. The list is enclosed in double quotes.

The "broadcast" property specifies that the content is broadcast or live (and thus may be suitable for splitting to multiple downstream clients). If the "broadcast" property is not specified, it implies that the content is on-demand (i.e., not broadcast or live.) The "**playlist**" property specifies that the content possibly consists of multiple entries from a server-side playlist.

The list of properties can be used by a caching proxy server to determine if the content is a suitable candidate for caching or splitting. The ability to determine if a candidate is suitable for caching or splitting is useful if the proxy server has sent a GetContentInfo request (section 2.2.2.2) because the features token on the Pragma header is not usually available in the response to a GetContentInfo request.<6>

The syntax of the directive is defined as follows:

```
stream-prop = ( "broadcast" / "playlist" )
StreamTypes = "x-wms-stream-type="
              %x22 stream-prop *( "," stream-prop ) %x22
```

2.2.1.2 Content-Type

The Content-Type header specifies the type of data that is included in the message payload (that is, the response to a GET request or the message body of a POST request).<7>

The syntax of the Content-Type header is defined as follows:

```
Ctype         = "application/octet-stream" /
                "application/vnd.ms.wms-hdr.asfv1" /
                "application/x-mms-framed" /
                "application/x-wms-getcontentinfo" /
                "application/x-wms-LogStats" /
                "application/x-wms-sendevent" /
                "text/plain"

Content-Type= "Content-Type: " Ctype [";charset=UTF-8"] CRLF
```

Example:

```
Content-Type: application/x-wms-LogStats;charset=UTF-8
```

2.2.1.2.1 application/octet-stream

This content-type specifies that the response consists of a sequence of bytes.

Clients MUST use contextual information (for example, knowledge about what kind of response is being expected) to determine how to interpret the response.

2.2.1.2.2 application/vnd.ms.wms-hdr.asfv1

This content-type specifies that the response consists of an **Advanced Systems Format (ASF)** header (as specified in [ASF]) encapsulated in a $H packet (section 2.2.3.5). The $H packet may be preceded by $P packets (section 2.2.3.7) and $M packets (section 2.2.3.6).

2.2.1.2.3 application/x-mms-framed

This content-type specifies that the response consists of a sequence of framed Microsoft Media Server (MMS) data packets, adhering to the syntax specified in section 2.2.3.

2.2.1.2.4 application/x-wms-getcontentinfo

This content-type is used in a POST request to query cache-control information from a server.<8> For more information, see GetContentInfo request (section 2.2.2.2).

2.2.1.2.5 application/x-wms-LogStats

This content-type specifies that the message body of the POST request contains a logging message in the XML format as specified in [MS-WMLOG]. This content-type is defined only for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<9>

2.2.1.2.6 application/x-wms-sendevent

This content-type specifies that the message body of the POST request contains a remote event message in the remote event format defined in section 2.2.5. This content-type is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<10>

2.2.1.2.7 text/plain

This content-type MAY be used when sending a logging message using a POST request. The logging message MUST be included in the log-line (section 2.2.1.4.10) token on the Pragma header and the message body of the POST request MUST be empty.<11>

2.2.1.3 Cookie

The syntax of the Cookie header MUST be as specified in [RFC2109].

This header is defined for use in requests sent to a server. Clients SHOULD share a single repository for RTSP cookies and HTTP cookies, and treat http:// and rtsp:// URLs as a single protocol.

This means that if a cookie is set for the URL rtsp://example.com/ using the RTSP protocol, and the client then sends an HTTP GET request for the URL http://example.com/, then the cookie SHOULD be included in the GET request, even though it was originally obtained through RTSP.

2.2.1.4 Pragma

The Windows Media HTTP Streaming Protocol uses the HTTP Pragma header field to communicate information specific to the operation of the protocol. The Pragma header consists of one or more comma-separated tokens, as specified in [RFC1945] section 10.12.

The following error conditions are treated in an implementation-specific manner:

- A Pragma header token which MUST or SHOULD be included by the sender is missing.
- A Pragma header token which MUST NOT or SHOULD NOT be included by the sender is received.
- A Pragma header token contains incorrect syntax, specifies an invalid value, or contains any other error.<12>

The Pragma header tokens are defined in the subsequent sections, AccelBW through xStopStrm.

2.2.1.4.1 AccelBW

When used in a request, this token specifies a transmission rate, in bits per second, that the client is requesting the server to use when transmitting the amount of multimedia data specified by the AccelDuration (section 2.2.1.4.2) token.

When used in a response, this token specifies the transmission rate, in bits per second, that the server is intending to use when transmitting the amount of multimedia data specified by the server's AccelDuration token.

A server is allowed to specify a value for AccelBW that is less than or equal to the value that the client requested, but it MUST NOT specify a larger value than the one requested by the client. If the client version (as specified by the client in the User-Agent header) is greater than or equal to 8.0, but less than 9.0, the value of the AccelBW token sent by the server MUST NOT exceed 1,048,576.

The AccelBW token is defined only for servers with a version greater than or equal to 5.0 (as specified by the server in the Server header) and only for clients with a version greater than or equal to 8.0 (as specified by the client in the User-Agent header).<13>

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the AccelBW token is defined as follows:

```
AccelBW          = "AccelBW=" 1*10DIGIT
```

Although not required, the AccelBW, AccelDuration, and LinkBW tokens are typically grouped together as shown in the following example.

Example:

```
Pragma: LinkBW=2147483647, AccelBW=1048576, AccelDuration=5000
```

2.2.1.4.2 AccelDuration

When used in a request, this token specifies an amount of multimedia data, in millisecond units, that the client is requesting the server to transmit at the bandwidth specified by the AccelBW (section 2.2.1.4.1) token.

When used in a response, this token specifies the amount of multimedia data, in milliseconds, that the server is intending to transmit at the bandwidth specified by the server's AccelBW token.

A server is allowed to specify a value for AccelDuration that is less than or equal to the value that the client requested, but MUST NOT specify a larger value than the one requested by the client.

The AccelDuration token is defined only for servers with a version greater than or equal to 5.0 (as specified by the server in the Server header), and only for clients with a version greater than or equal to 8.0 (as specified by the client in the User-Agent header).<14>

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the AccelDuration token is defined as follows:

```
AccelDuration    = "AccelDuration=" 1*10DIGIT
```

Although not required, the AccelBW, AccelDuration, and LinkBW tokens are typically grouped together as shown in the following example:

```
Pragma: LinkBW=2147483647, AccelBW=1048576, AccelDuration=5000
```

2.2.1.4.3 BurstBW

When used in a request, this token specifies a transmission rate, in bits per second, that the client is requesting the server to use when transmitting the amount of multimedia data specified by the BurstDuration (section 2.2.1.4.4) token. The client that is sending the request is normally an intermediate device, which is relaying a request on behalf of another client. The AccelBW (section 2.2.1.4.1) token, if specified, is the one provided by the original client. The BurstBW token specifies the transmission rate requested by the intermediate device.

When used in a response, this token specifies the transmission rate, in bits per second, that the server is intending to use when transmitting the amount of multimedia data specified by the server's BurstDuration token.

A server is allowed to specify a value for BurstBW that is less than or equal to the value that the client requested, but it MUST NOT specify a larger value than the one requested by the client.

The BurstBW token is defined only for servers with a version greater than or equal to 5.0 (as specified by the server in the Server header), and only for clients with a version greater than or equal to 9.0 (as specified by the client in the User-Agent header).<15>

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the BurstBW token is defined as follows:

```
BurstBW        = "BurstBW=" 1*10DIGIT
```

Although not required, the BurstBW and BurstDuration tokens are typically grouped together as shown in the following example:

```
Pragma: BurstBW=1048576, BurstDuration=5000
```

2.2.1.4.4 BurstDuration

When used in a request, this token specifies an amount of multimedia data, in milliseconds, that the client is requesting the server to transmit at the bandwidth specified by the BurstBW (section 2.2.1.4.3) token. The client that is sending the request is normally an intermediate device, which is relaying a request on behalf of another client. The AccelDuration (section 2.2.1.4.2) token, if specified, is the one provided by the original client. The BurstDuration token specifies the amount of data requested by the intermediate device.

When used in a response, this token specifies the amount of multimedia data, in milliseconds, that the server is intending to transmit at the bandwidth specified by the server's BurstBW token.

A server is allowed to specify a value for BurstDuration that is less than or equal to the value that the client requested, but it MUST NOT specify a larger value than the one requested by the client.

The BurstDuration token is defined only for servers with a version greater than or equal to 5.0 (as specified by the server in the Server header) and only for clients with a version greater than or equal to 9.0 (as specified by the client in the User-Agent header).<16>

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the BurstDuration token is defined as follows:

```
BurstDuration    = "BurstDuration=" 1*10DIGIT
```

Although not required, the BurstBW and BurstDuration tokens are typically grouped together as shown in the following example:

```
Pragma: BurstBW=1048576, BurstDuration=5000
```

2.2.1.4.5 client-id

This token specifies the server session to which the client request should be applied.

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the client-id token is defined as follows:

```
client-id        = "client-id=" 1*10DIGIT
```

Example:

```
Pragma: client-id=1234
```

2.2.1.4.6 client-lag

This token indicates to the server the amount of time, in milliseconds, by which the client may be lagging behind the server as a result of predictive streaming playlist changes. The server SHOULD then slow down the pushing of headers to the client by several milliseconds to ensure that the client does not fall too far behind.

Lag is defined as the amount of data queued on the client that is not intended for buffering. For example, if the client still has 5 seconds left to render from a playlist entry but the server has already streamed the first 7 seconds of the next playlist entry, and the client's network buffer is 3 seconds, then the lag is 5 + 7 - 3 = 9 seconds. Under normal circumstances, the lag is expected to be zero but implementation-specific issues can result in a non-zero lag.

This token is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header). The server MAY ignore the value specified by this token.<17>

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the client-lag token is defined as follows:

```
client-lag      = "client-lag=" 1*10DIGIT
```

Example:

```
Pragma: client-lag=250
```

2.2.1.4.7 expect-new-header

This token is used as a notification from the server to the client that the $M (section 2.2.3.6) and $H (section 2.2.3.5) packets, which are sent as the first packets in the response to the Play request, will be immediately followed by a $C packet (section 2.2.3.2) and another $M and $H packet.

This token can have a value of 0 or 1. If the server is intending to transmit a stream-change notification ($C packet) in response to a Play request, the value of this token MUST be set to 1; otherwise, the token SHOULD be omitted. Alternatively, instead of omitting the token, its value MAY be specified as 0.

If this token is omitted or its value is 0, the client SHOULD assume that the second packet that the server will transmit in response to a Play request will not be a $C packet.

If this token is specified and its value is 1, the client MUST assume that the second packet that the server will transmit in response to a Play request will be a $C packet.

This token is only defined for use in responses sent to a client.

The syntax of the expect-new-header token is defined as follows:

```
expect-new-header    = "expect-new-header=" ("0" / "1")
```

Example:

```
Pragma: expect-new-header=1
```

2.2.1.4.8 features

This token specifies a list of properties and capabilities that are applicable to the current content or playlist entry. When included in a server response, the features token MUST include at least one feature token.

This token is only defined for use in responses sent to a client.

The syntax of the features token is defined as follows:

```
wm-feat     = "broadcast" / "last" / "live" / "playlist" /
              "reliable" / "seekable" / "skipbackward" /
              "skipforward" / "stridable"
features    = "features=" %x22 [wm-feat *8("," wm-feat)] %x22
```

Example:

```
Pragma: features="seekable, stridable"
```

2.2.1.4.8.1 broadcast

Indicates to a client that the content is being broadcast.

If the playlist (section 2.2.1.4.8.4) feature token is specified, and the client version is less than 9.0 (as specified by the client in the User-Agent header), then the broadcast feature token MUST also be specified.

2.2.1.4.8.2 last

Indicates that the content is the last entry in a server-side playlist.<18>

2.2.1.4.8.3 live

Indicates to a client that the content is live content from an encoder.

2.2.1.4.8.4 playlist

Indicates to a client that the content is an entry, from possibly multiple entries, in a server-side playlist.<19>

2.2.1.4.8.5 reliable

Indicates that the content SHOULD be delivered over a reliable data communications transport mechanism, such as TCP.

2.2.1.4.8.6 seekable

Indicates that the server supports seeking within the content using the stream-time (section 2.2.1.4.28) token and the stream-offset (section 2.2.1.4.25) token.

If the playlist feature token is specified, and the client version is less than 9.0 (as specified by the client in the User-Agent header), then the seekable feature token MUST NOT be specified.

2.2.1.4.8.7 skipbackward

Indicates that the server supports skipping to the previous entry in the server-side playlist by using the pl-offset (section 2.2.1.4.20) token.

This feature token is only defined for clients with a version greater than or equal to 9.0 (as specified by the client in the User-Agent header).<20>

2.2.1.4.8.8 skipforward

Indicates that the server supports skipping to the next entry in the server-side playlist by using the pl-offset (section 2.2.1.4.20) token.

This feature token is only defined for clients with a version greater than or equal to 9.0 (as specified by the client in the User-Agent header).<21>

2.2.1.4.8.9 stridable

Indicates that the server supports fast-forward or rewind of the content using the rate (section 2.2.1.4.22) token on the Pragma header.

2.2.1.4.9 LinkBW

This token MAY be sent by the client to inform the server of the connection bandwidth, in bits per second.

This token is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<22>

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the LinkBW token is defined as follows:

```
LinkBW       = "LinkBW=" 1*10DIGIT
```

Although not required, the AccelBW (section 2.2.1.4.1) token, the AccelDuration (section 2.2.1.4.2) token, and the LinkBW token are typically grouped together as shown in the following example.

Example:

```
Pragma: LinkBW=2147483647, AccelBW=1048576, AccelDuration=5000
```

2.2.1.4.10 log-line

This token specifies logging information for the server. If a client sends the log-line token, it MUST be included in a Pragma header that does not include any other tokens.

The syntax of the log-line token is defined as follows:

```
w3c-token   = *VCHAR
LogLine     = "log-line=" *( w3c-token SP ) w3c-token
```

The w3c-token MUST adhere to the "legacy" logging format specified in [MS-WMLOG].

2.2.1.4.11 max-duration

This token specifies the amount of data that the server should stream, expressed as a time interval in milliseconds. If the most significant bit of the value is 1, the remaining 31 bits are treated as an absolute duration. If the most significant bit is 0, the remaining 31 bits are treated as the duration relative to the stream-time (section 2.2.1.4.28) token.

The value MUST be an integer in the range from 0 through 4,294,967,295.

If the value is 0 or the value specifies an amount of data that exceeds the largest amount of data that can be streamed for the current content or playlist, then the remaining content or playlist MUST be streamed until its end is reached.

This token is only defined for use in requests sent to a server. This token MAY be ignored by the server.<23>

The syntax of the max-duration token is defined as follows:

```
max-duration    = "max-duration=" 1*10DIGIT
```

Example for absolute duration:

```
Pragma: max-duration=1000
```

2.2.1.4.12 no-cache

This token specifies that HTTP caches are not allowed to use a cached response. The Pragma general-header field is used to include implementation-specific directives, such as the no-cache token, as specified in HTTP 1.0 [RFC1945] section 10.12.

The syntax of the no-cache token is defined as follows:

```
no-cache = "no-cache"
```

Example:

```
Pragma: no-cache
```

2.2.1.4.13 packet-num

This token specifies the Advanced Systems Format (ASF), as specified in [ASF], packet number at which playlist should begin.

The value MUST be an integer in the range from 0 through 4,294,967,295. If the value is equal to 4,294,967,295, then the server MUST NOT use the packet-num token and MUST use either the stream-time or the stream-offset tokens to determine playback offset.

This token is only defined for use in requests sent to a server.

The syntax of the packet-num token is defined as follows:

```
packet-num    = "packet-num=" 1*10DIGIT
```

Example:

```
Pragma: packet-num=0
```

2.2.1.4.14 packet-pair-experiment

This token is used by clients to request that the server start a packet-pair experiment. It is also used by servers to indicate whether they will comply with the request.

This token MUST have a value of either 0 or 1.

If this token is not specified, a value of 0 MUST be assumed.

This token is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<24>

The syntax of the packet-pair-experiment token is defined as follows:

```
packet-pair-experiment = "packet-pair-experiment="
                         ("0" / "1")
```

Example:

```
Pragma: packet-pair-experiment=1
```

2.2.1.4.15 pipeline-experiment

This token is used by a client to specify that it supports Pipeline requests (section 2.2.2.5). A server uses the token to specify whether it accepts pipeline requests.

This token MUST have a value of either 0 or 1.

To determine if the server accepts pipeline requests, a client SHOULD include this token and specify its value as 1. The server SHOULD respond with the pipeline-experiment token regardless of whether it accepts pipeline requests or not. The server MUST specify the value of this token as 1 if the client is to proceed with sending the pipeline requests. If the client specified the pipeline-experiment token in a request and the response does not include this token, or if the response specifies this token with a value of 0, then the client MUST assume that the server does not accept pipeline requests.

Because pipelining is a feature specific to HTTP 1.1, a client that uses HTTP 1.0 MUST NOT send the pipeline-experiment token.

The pipeline-experiment token is defined only for servers with a version greater than or equal to 5.0 (as specified by the server in the Server header), and only for clients with a version greater than or equal to 9.0 (as specified by the client in the User-Agent header).<25>

The syntax of the pipeline-experiment token is defined as follows:

```
pipeline-experiment = "pipeline-experiment=" ("0" / "1")
```

Example:

```
Pragma: pipeline-experiment=1
```

2.2.1.4.16 pipeline-request

This token is used by the client to identify the current request as a pipeline request (section 2.2.2.5). When attempting to determine if the pipelined mode of operation can be used, the client will send two pipeline requests and the pipeline request token MUST be present in both of those requests. The two requests MUST be sent "back-to-back", that is, be pipelined, as specified in [RFC2616] section 8.1.2.2.

This token MUST have a value of 1.

This token is only defined for use in requests sent to a server using the HTTP 1.1 protocol.<26>

The syntax of the pipeline-request token is defined as follows:

```
pipeline-request    = "pipeline-request=1"
```

Example:

```
Pragma: pipeline-request=1
```

2.2.1.4.17 pipeline-result

This token is used in a response to a pipeline request (section 2.2.2.5), and it specifies if the pipelined mode of the protocol can be used. The token can have a value of 0 or 1.

This token is only defined for use in responses sent to a client.<27>

The syntax of the pipeline-result token is defined as follows:

```
pipeline-result = "pipeline-result=" ("0" / "1")
```

Example:

```
Pragma: pipeline-result=1
```

2.2.1.4.18 playlist-gen-id

This token specifies the identifier of the playlist entry to which the current request applies. After the client has obtained the identifier for the current playlist entry (using the $M packet (section 2.2.3.6)), the client MUST include it in all subsequent GET and POST requests for the current session. In other words, all requests that include a Pragma header with the client-id (section 2.2.1.4.5) token MUST also include the playlist-gen-id token if the server has specified one.

This token is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<28>

The value MUST be an integer in the range from 1 through 4,294,967,295.

The syntax of the playlist-gen-id token is defined as follows:

```
playlist-gen-id = "playlist-gen-id=" 1*10DIGIT
```

Example:

```
Pragma: playlist-gen-id=2587
```

2.2.1.4.19 playlist-seek-id

This token requests the server to seek the playlist entry with the ID specified as the value of this token.

If a client includes this token in a request, the value MUST be either the identifier of the current playlist entry or the identifier of the previous playlist entry.<29>

The value MUST be an integer in the range from 1 through 4,294,967,295.

The syntax of the playlist-seek-id token is defined as follows:

```
playlist-seek-id = "playlist-seek-id=" 1*10DIGIT
```

Example:

```
Pragma: playlist-seek-id=3470
```

For more information about how the server specifies the ID of a playlist entry, see $M packet (section 2.2.3.6).

2.2.1.4.20 pl-offset

This token indicates to the server whether to move forward or backward to an entry in a playlist relative to the entry ID specified in the playlist-seek-id (section 2.2.1.4.19) token. If the value of the

pl-offset token is 1, it specifies that the server is requested to move forward to the next entry. A value of -1 specifies that the server is requested to move backward to the previous entry.

This token is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<30>

The syntax of the pl-offset token is defined as follows:

```
pl-offset = "pl-offset=" ("0" / "1" / "-1")
```

Example:

```
Pragma: pl-offset=1
```

2.2.1.4.21 proxy-client-agent

This token is sent by intermediate devices (such as proxy servers) and specifies the information that the original client who initiated the HTTP request specified on the User-Agent (section 2.2.1.8) header.

Because each intermediate device replaces the information on the User-Agent header with their own information, the proxy-client-agent token enables the information on the original User-Agent header to be preserved and forwarded across possibly multiple intermediate devices to the server.

This header is only defined for requests sent to a server by an intermediate device, such as a proxy server, which is acting as a client.

The syntax of the proxy-client-agent token is defined as follows:

```
proxy-client-agent = "proxy-client-agent="
                      %x22
                      user-agent-data
                      %x22
```

Example:

```
Pragma: proxy-client-agent="NSPlayer/9.0.0.0"
```

2.2.1.4.22 rate

This token specifies the requested playback mode (trick mode or normal playback). For example, a value of -5 specifies rewind, 5 specifies fast-forward, and 1 specifies normal playback.

The value MUST be a number in the range from -5.000 through 5.000. However, a value of 0 MUST NOT be specified.

This token is only defined for use in requests sent to a server.

The syntax of the rate token is defined as follows:

```
rate-dig = "1" / "2" / "3" / "4" / "5"
rate     = "rate=" (rate-dig / ("-" rate-dig))
           ["." 1*3DIGIT]
```

Example:

```
Pragma: rate=1.000
```

2.2.1.4.23 request-context

This token specifies a value whose significance is implementation-specific. A client MAY choose to send this token in GET requests. Servers SHOULD ignore this token.<31>

The value MUST be an integer in the range from 1 through 4,294,967,295.

This token is only defined for use in requests sent to a server.

The syntax of the request-context token is defined as follows:

```
request-context = "request-context=" 1*10DIGIT
```

Example showing first request sent by client:

```
Pragma: request-context=1
```

2.2.1.4.24 speed

When used in a request, this token specifies a speed-up factor, in which the client is requesting that the server apply its normal transmission rate.

When used in a response, this token specifies the speed-up factor that the server is intending to apply to its normal transmission rate.

The normal transmission rate is defined as the sum of the average bit rates of the ASF streams that will be transmitted. If an AcceleBW (section 2.2.1.4.1) token has been specified in the request, this MUST NOT affect the determination of the normal bit rate.

This token is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<32>

The value MUST be a real number in the range from -5.000 to 5.000 inclusive. However, a value of 0 MUST NOT be specified.

The syntax of the Speed token is defined as follows:

```
rate-dig = "1" / "2" / "3" / "4" / "5"
speed    = "Speed=" (rate-dig / ("-" rate-dig))
           ["." 1*3DIGIT]
```

In the example below, the value indicates 5 times faster than real time:

```
Pragma: Speed=5.000
```

2.2.1.4.25 stream-offset

This token specifies a byte offset from which playback should begin. The byte offset is measured from the beginning of the ASF file and is expressed as two decimal numbers separated by a colon. The value for stream-offset MUST match the start of a packet within the ASF file.

Each of the two numbers MUST be in the range from 0 through 4,294,967,295.

The first number gives the most significant 32 bits of the byte offset and the second number gives the least significant 32 bits of the byte offset. If both numbers are set to 4,294,967,295, the server MUST use either the stream-time or the packet-num token to determine the start offset and MUST NOT use the value of the stream-offset token.

This token is only defined for use in requests sent to a server.

The syntax of the stream-offset token is defined as follows:

```
stream-offset = "stream-offset=" 1*10DIGIT ":" 1*10DIGIT
```

Example to start playing at 1,024 bytes into the file:

```
Pragma: stream-offset=0:1024
```

Example to ignore stream-offset:

```
Pragma: stream-offset=4294967295:4294967295
```

2.2.1.4.26 stream-switch-count

This token specifies the number of stream-switch entries in the stream-switch-entry (section 2.2.1.4.27) token.

This token is only defined for use in requests sent to a server.

The value MUST be an integer in the range from 1 through 4,294,967,295.

The syntax of the stream-switch-count token is defined as follows:

```
stream-switch-count = "stream-switch-count=" 1*10DIGIT
```

Example showing two switches:

```
Pragma: switch-stream-count=2
```

2.2.1.4.27 stream-switch-entry

This token specifies one or more stream-switch entries. The number of stream-switch entries MUST be equal to the value specified by the stream-switch-count (section 2.2.1.4.26) token.

If a client sends this token, it MUST be sent on a separate Pragma header, that is, no other tokens are allowed to be included on the same Pragma header as the stream-switch-entry token.

This token is only defined for use in requests sent to a server.

Each stream-switch-entry consists of three 16-bit hexadecimal numbers separated by colons:

- The first number is the source ASF stream number to switch from.
- The second number is the destination ASF stream number to switch to.
- The third number is the stream thinning level to apply to the destination ASF stream number.
  - Thinning level 0: The ASF packets in the stream should not be filtered (no thinning is applied).
  - Thinning level 1: The ASF packets in the stream should be filtered such that they only contain key frame media objects.
  - Thinning level 2: The ASF packets in the stream should be filtered such that all media objects belonging to the stream are removed.

  Either stream can be set to a special value of "ffff" to indicate that no stream is specified. If the destination stream number is set to "ffff", then the thinning level MUST be ignored. Stream numbers are located in the ASF file header delivered by the server prior to streaming the file. For details about ASF headers, ASF packets, and media objects, see the ASF specification as described in [ASF]. For more information about stream thinning in the Windows Media Services 9.0 Series SDK, see [WMSSDK].

The syntax of the stream-switch-entry token is defined as follows:

```
stream-switch-entry = "stream-switch-entry="
                      1*4HEXDIG ":" 1*4HEXDIG ":"
                      ("0" / "1" / "2")
```

Example for "turning on" streams one and two:

```
Pragma: switch-stream-entry=ffff:1:0 ffff:2:0
```

Example for "turning off" stream two:

```
Pragma: switch-stream-entry=2:ffff:0
```

Example for thinning stream one:

```
Pragma: switch-stream-entry=1:1:1
```

2.2.1.4.28 stream-time

This token specifies the absolute time position, in milliseconds, from which playback should begin. A value of 0 specifies that playback is requested to begin from time 0 or the beginning of the content if the content is live (as specified by the features (section 2.2.1.4.8) token).

The value MUST be an integer in the range from 0 through 4,294,967,295.

A server MUST NOT use the stream-time token if the packet-num (section 2.2.1.4.13) token is not equal to 4,294,967,295.

This token is only defined for use in requests sent to a server.

The syntax of the stream-time token is defined as follows:

```
stream-time = "stream-time=" 1*10DIGIT
```

Example:

```
Pragma: stream-time=1000
```

2.2.1.4.29 timeout

This token specifies the longest time interval, in milliseconds, that the server allows between KeepAlive (section 2.2.2.3) requests without timing out an idle session.

If this token is not specified, clients MUST assume that the server uses a time out of one minute. The behavior of the server if the client has been idle for the time out time interval is implementation-specific.<33>

This token is only defined for use in responses sent to a client.

The value MUST be an integer in the range from 0 through 4,294,967,295.

The syntax of the timeout token is defined as follows:

```
timeout = "timeout=" 1*10DIGIT
```

Example:

```
Pragma: timeout=60000
```

2.2.1.4.30 version11-enabled

This token specifies if the pipelined mode of the protocol should be used.

The token MUST have a value of 0 or 1.<34>

This token is only defined for use in requests sent to a server.

The syntax of the version11-enabled token is defined as follows:

```
version11-enabled = "version11-enabled=" ("0" / "1")
```

Example:

```
Pragma: version11-enabled=1
```

2.2.1.4.31 version-info

This token specifies the earliest version number of Windows Media Player that is supported by Windows Media Services.

This token is only defined for use in responses sent to a client.<35>

The syntax of the version-info token is defined as follows:

```
major        = 1*2DIGIT
minor        = 1*2DIGIT "." 1*4DIGIT "." 1*4DIGIT
version-info = "version-info= " major "." minor
```

Example:

```
Pragma: version-info=4.0.1.3850
```

2.2.1.4.32 version-url

This token specifies a URL from which Windows Media Player can download an update.

This token is only defined for use in responses sent to a client.<36>

The syntax of the version-url token is defined as follows:

```
version-url = "version-url=" %x22 httpurl %x22
httpurl     ; as defined in [RFC1738]
```

Example:

```
Pragma: version-url=
```

```
"http://codecs.microsoft.com/isapi/mpupgrade.dll"
```

2.2.1.4.33 xClientGUID

This token specifies an identifier that uniquely identifies the client software installation that originated the request. The identifier MUST be identical for all requests belonging to the same streaming session. The identifier MUST be a **globally unique identifier (GUID)**. The GUID is expressed in registry format and is not enclosed in quotation marks, as shown by the ABNF syntax below.

This token is only defined for use in requests sent to a server.

The syntax of the xClientGUID token is defined as follows:

```
guid-value  = "{" 8HEXDIG "-" 4HEXDIG "-" 4HEXDIG "-"
              4HEXDIG "-" 12HEXDIG "}"
xClientGUID = "xClientGUID=" guid-value
```

Example:

```
xClientGUID={11223344-1122-1122-1122-AABBCCDDEEFF}
```

2.2.1.4.34 xKeepAliveInPause

This token is used to request the server to not time out the current session (the session is identified using the client-id (section 2.2.1.4.5) token).

The token MUST have a value of 1.

This token is only defined for use in requests sent to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<37>

A server MAY ignore the xKeepAliveInPause token.

The syntax of the xKeepAliveInPause token is defined as follows:

```
xKeepAliveInPause = "xKeepAliveInPause=1"
```

Example:

```
Pragma: xKeepAliveInPause=1
```

2.2.1.4.35 xPlayNextEntry

This token specifies that the client is ready to receive the current entry in a server-side playlist, and that the server should start streaming the $D packets (section 2.2.3.3) for that entry if it has not already done so.

The token MUST have a value of 1.

This token is defined only for use in requests sent using the pipelined mode of the protocol to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header.) <38>

The syntax of the xPlayNextEntry token is defined as follows:

```
xPlayNextEntry = "xPlayNextEntry=1"
```

Example:

```
Pragma: xPlayNextEntry=1
```

2.2.1.4.36 xPlayStrm

This token is used to request the server to start streaming the content.

The token MUST have a value of either 0 or 1.

This token is only defined for use in requests sent to a server.

The syntax of the xPlayStrm token is defined as follows:

```
xPlayStrm = "xPlayStrm=" ("0" / "1")
```

Example (Play request):

```
Pragma: xPlayStrm=1
```

2.2.1.4.37 xResetStrm

This token indicates to the client that the session that was specified by the client in the client-id token in the request has ended and that a new session has been created. For example, this can happen when a request is received for an invalid session or for a session that has timed out (see timeout (section 2.2.1.4.29) token).

The token MUST have a value of 1.

This token is only defined for use in responses sent to a client.

The syntax of the xResetStrm token is defined as follows:

```
xResetStrm = "xResetStrm=1"
```

Example:

```
Pragma: xResetStrm=1
```

2.2.1.4.38 xStopStrm

This token is used to request that the server stop streaming without closing the connection.

The token MUST have a value of 1.

This token is only defined for use in requests sent by using the pipelined mode of the protocol to a server with a version greater than or equal to 5.0 (as specified by the server in the Server header).<39>

The syntax of the xStopStrm token is defined as follows:

```
xStopStrm = "xStopStrm=1"
```

Example:

```
Pragma: xStopStrm=1
```

2.2.1.5 Server

The Server header specifies the major and minor version numbers of the Microsoft software product that is responding to the HTTP request.

This header is only defined for use in responses sent to a client.

The syntax of the Server header is defined as follows:

```
server-token = ( "Cougar" / "Rex" )
major        = 1*2DIGIT
minor        = 1*2DIGIT ["." 1*4DIGIT "." 1*4DIGIT]
product      = ; as defined in section 3.7 of [RFC1945]
Server       = "Server: server-token "/"
               major "." minor *( SP product ) CRLF
```

Servers MUST assign the values of the server-token, major, and minor ABNF syntax elements according to the table below:

Product Name	Server-Token	Major	Minor
Windows Media Services 4.0	Cougar	4	0
Windows Media Services 4.1	Cougar	4	1
Windows Media Services 9.0	Cougar	9	0
Windows Media Services 9.1	Cougar	9	1
Windows Media Encoder 4.0	Rex	4	0
Windows Media Format 7.0 SDK	Rex	7	0

Product Name	Server-Token	Major	Minor
Windows Media Format 7.1 SDK	Rex	7	1
Windows Media Player for Windows XP	Rex	8	0
Windows Media Format 9 Series SDK	Rex	9	0
Windows Media Format 9.5 SDK	Rex	10	0
Windows Vista	Rex	11	0

2.2.1.6 Set-Cookie

The syntax of the Set-Cookie header MUST be as specified in [RFC2109].

This header is defined for use in responses sent to a client. Clients SHOULD share a single repository for RTSP cookies and HTTP cookies, and treat http:// and rtsp:// URLs as a single protocol.

This means that if a cookie is set for the URL rtsp://example.com/, and then a cookie with the same name is set for the URL http://example.com/, then the second cookie overrides the first cookie, because the two URLs are considered equivalent.

2.2.1.7 Supported

This header is used for specifying features of the protocol that are supported and that are allowed to be used in the current session. Different features may apply to different entries in a server-side playlist.

Some features are not to be used unless indicated by the Supported header. Other features are assumed to be supported by default, or as implied by the User-Agent or Server header, and the absence of a particular feature on the Supported header is then used to indicate that the feature is not to be used.

The Supported header is defined only for servers with a version greater than or equal to 9.0 (as specified by the server in the Server header) and only for clients with a version greater than or equal to 9.0 (as specified by the client in the User-Agent header).<40>

If a feature is listed on the Supported header, then the feature is supported and the feature SHOULD be used, if appropriate. If a feature has been defined for use on the Supported header but is not listed on the Supported header, then that feature MUST NOT be used.

A missing Supported header (from either a request or a response) MUST NOT be interpreted as changing the list of features that is currently supported.

The syntax of the Supported header is defined as follows:

```
WMCfeat   = "com.microsoft.wm.fastcache"
            / "com.microsoft.wm.predstrm"
            / "com.microsoft.wm.srvppair"
            / "com.microsoft.wm.sswitch"
            / "com.microsoft.wm.startupprofile"
Supported = "Supported: " WMCfeat *4["," [SP] WMCfeat]
            CRLF
```

Example:

```
Supported: com.microsoft.wm.srvppair,
  com.microsoft.wm.sswitch, com.microsoft.wm.predstrm,
  com.microsoft.wm.startupprofile
```

The tokens that can be used on the Supported header are specified in the subsequent sections (com.microsoft.wm.fastcache through com.microsoft.wm.startupprofile).

2.2.1.7.1 com.microsoft.wm.fastcache

This token specifies that the server permits the use of the Speed (section 2.2.1.4.24) token on the Pragma header.

This token is only defined for use in responses sent to a client.

If a server never sends the Supported (section 2.2.1.7) header, clients MUST assume that usage of the Speed token on the Pragma header is not allowed.

2.2.1.7.2 com.microsoft.wm.predstrm

This token specifies support for Predictive Stream Selection. This is a technique in which the server selects a set of streams from the next entry in a server-side playlist on the client's behalf and starts streaming those streams. When Predictive Stream Selection is not used, the server will not start streaming the next entry in the server-side playlist until the client has sent a SelectStream request (section 2.2.2.8) or PlayNextEntry request (section 2.2.2.7), as appropriate (it depends on whether pipelined mode or non-pipelined mode is used).

If a client never sends the Supported (section 2.2.1.7) header, servers SHOULD assume that the client supports the use of Predictive Stream Selection.

If a server never sends the Supported header, clients MUST assume that the server will always use Predictive Stream Selection.

2.2.1.7.3 com.microsoft.wm.srvppair

This token specifies that the server permits the use of the packet-pair-experiment (section 2.2.1.4.14) token on the Pragma header.

If a server never sends the Supported (section 2.2.1.7) header, clients MUST assume that usage of the packet-pair-experiment token on the Pragma header is not allowed.

2.2.1.7.4 com.microsoft.wm.sswitch

This token specifies support for the SelectStream request (section 2.2.2.8).

If a client never sends the Supported (section 2.2.1.7) header, servers SHOULD assume that the client supports sending the SelectStream request.

If a server never sends the Supported header, clients MUST assume that the server does support the SelectStream request.

2.2.1.7.5 com.microsoft.wm.startupprofile

This token specifies support for the X-StartupProfile (section 2.2.1.12) header.

If a client never sends the Supported (section 2.2.1.7) header, servers SHOULD assume that the client does not support parsing the X-StartupProfile header.

If a server never sends the Supported header, clients SHOULD assume that the server will not send the X-StartupProfile header.<41>

2.2.1.8 User-Agent

The User-Agent header specifies the major and minor version number of the Microsoft software product that is sending the HTTP request.

This header is only defined for use in requests sent to a server.

The syntax of the User-Agent header is defined as follows:

```
client-token = ( "NSPlayer" / "NSServer" / "WMCacheProxy" )
major = 1*2DIGIT
minor = 1*2DIGIT ["." 1*4DIGIT "." 1*4DIGIT]
product = ; as defined in section 3.7 of [RFC1945]
user-agent-data = client-token "/" major "." minor
                  [ ";" SP "via" SP "WMCacheProxy" ]
                  *( SP product )
User-Agent= "User-Agent: " user-agent-data CRLF
```

Clients MUST assign the values of the client-token, and major and minor ABNF syntax elements according to the table below:

Product name	Client-Token	Major	Minor
Windows Media Services 4.0	NSServer	4	0
Windows Media Services 4.1	NSServer	4	1
Windows Media Services 9.0	NSServer	9	0
Windows Media Services 9.0	NSPlayer	9	0
Windows Media Services 9.0	WMCacheProxy	9	0
Windows Media Services 9.1	NSServer	9	1
Windows Media Services 9.1	NSPlayer	9	1
Windows Media Services 9.1	WMCacheProxy	9	1
Windows Media Player 6.4	NSPlayer	6	4

Product name	Client-Token	Major	Minor
Windows Media Format 7.0 SDK	NSPlayer	7	0
Windows Media Format 7.1 SDK	NSPlayer	7	1
Windows Media Player for Windows XP	NSPlayer	8	0
Windows Media Format 9 Series SDK	NSPlayer	9	0
Windows Media Format 9.5 SDK	NSPlayer	10	0
Windows Vista	NSPlayer	11	0

2.2.1.9 X-Accept-Authentication

The X-Accept-Authentication header specifies the HTTP authentication schemes that the client supports. The authentication schemes that are supported by a server, if any, are implementation-specific and MAY be enabled arbitrarily by a server administrator.

The client MUST list the authentication scheme names in order of preference with the most preferred authentication scheme listed first.

If a server or a proxy requires the client to be authenticated, it SHOULD use one of the schemes listed by the client on the X-Accept-Authentication header as the HTTP authentication scheme. However, if the server is acting as a proxy server, it SHOULD use the X-Accept-Proxy-Authentication (section 2.2.1.10) header first and only use the X-Accept-Authentication header if no X-Accept-Proxy-Authentication header is specified in the request.

If multiple X-Accept-Authentication headers are present, or if the header contains multiple comma-separated scheme names, the server MUST consider the scheme names to be listed in order of preference. The server MUST consider the most preferred authentication scheme to be the scheme that is listed first.<42>

This header is only defined for use in requests sent to a server.

HTTP authentication information is specified in [RFC1945] section 11.

The syntax of the X-Accept-Authentication header is defined as follows:

```
X-Accept-Authentication = "X-Accept-Authentication: "
  auth-scheme *("," auth-scheme) CRLF
auth-scheme    ; as defined in section 11 of [RFC1945]
```

Example:

```
X-Accept-Authentication: Negotiate, MS-NLMP, Digest, Basic
```

2.2.1.10 X-Accept-Proxy-Authentication

The X-Accept-Proxy-Authentication header specifies the HTTP authentication schemes that the client supports when challenged by a proxy server. The authentication schemes that are supported by

proxy servers, if any, are implementation-specific and MAY be enabled arbitrarily by a server administrator.

The client MUST list the authentication scheme names in order of preference with the most preferred authentication scheme listed first.

If a proxy server requires the client to be authenticated, it SHOULD use one of the schemes listed by the client on the X-Accept-Proxy-Authentication header as the HTTP authentication scheme. If multiple X-Accept-Proxy-Authentication headers are present, or if the header contains multiple comma-separated scheme names, the server MUST consider the scheme names to be listed in order of preference. The server MUST consider the most preferred authentication scheme to be the scheme that is listed first.<43>

An example of how HTTP authentication works is specified in [RFC1945] section 11.

This header is only defined for use in requests sent to a server.

The syntax of the X-Accept-Proxy-Authentication header is defined as follows:

```
X-Accept-Proxy-Authentication = "X-Accept-Proxy-Authentication: "
        auth-scheme *("," auth-scheme) CRLF
auth-scheme    ; as defined in section 11 of [RFC1945]
```

Example:

```
X-Accept-Proxy-Authentication: Negotiate, MS-NLMP, Digest, Basic
```

2.2.1.11 X-Proxy-Client-Verb

This header is sent by intermediate devices (such as proxy servers) and specifies the name of the HTTP or RTSP request method used by the client that triggered the HTTP request sent by the intermediate device.

The header is useful when Digest authentication as specified in [RFC2617] is used and one or more of the intermediate devices is performing a protocol translation between HTTP and RTSP, or vice versa. When Digest authentication is used, the name of the HTTP or RTSP request method might be a part of the authentication challenge. Intermediate devices may change the request method. In particular, this is true if the intermediate device translates requests between different streaming protocols. The X-Proxy-Client-Verb header enables the name of the request method to be preserved and be forwarded across possibly multiple intermediate devices to the server.

This header is only defined for requests sent to a server by an intermediate device, such as a proxy server, which is acting as a client.

The syntax of the X-Proxy-Client-Verb header is defined as follows:

```
X-Proxy-Client-Verb = "X-Proxy-Client-Verb: "
            ( Method    ; section 6.1 of [RFC2326]
            | Method )  ; section 5.1.1 of [RFC2616]
```

2.2.1.12 X-StartupProfile

This header specifies a list of streaming bit rates, and for each bit rate it specifies the maximum amount of data that the audio and video decoders will need and the time stamp of the ASF payload at which this maximum occurs. The client should buffer at least the amount of data indicated by this header to prevent buffer underflow.

The information on this header is only valid during an initial time period during which the server is transmitting data faster than real time. The header specifies the first and last time stamp of the ASF payloads for which the information is valid.

The *Rate*, *MaxBytes*, *Time*, and *ByteRate* parameters on the X-StartupProfile header are arrays. This means that the first value in the *Rate* array corresponds to the first value in the *MaxBytes* array, and the second value in the *Rate* array corresponds to the second value in the *MaxBytes* array, and so on.

This header is only defined for use in responses sent to a client.<44>

The syntax of the X-StartupProfile header is defined as follows:

```
X-StartupProfile= "X-StartupProfile: " XSP-Rate ";"
  XSP-MaxBytes ";" XSP-Time ";" XSP-STime ";"
  XSP-LTime ";" XSP-MaxDTime ";" XSP-MaxDSTime ";"
  XSP-ByteRate ";" CRLF
```

Example:

```
X-StartupProfile: Rate=10,12,15,20,30;
MaxBytes=12874,9407,9086,8551,7482;Time=12512,34,34,34,34;
StartTime=1694318962;LastTime=19521;MaxDiffTime=0;
MaxDiffSndTime=0;ByteRate=30794,31469,31469,31469,31469;
```

2.2.1.12.1 Rate

This parameter contains a comma-separated list of possible acceleration rates, each of which MUST be expressed as a speed-up factor relative to the average bit rate of the content, multiplied by 10. For example, a value of 15 indicates that the rate equals 1.5 times the average bit rate of the content.

The syntax of the *Rate* parameter is defined as follows:

```
rate-value= 1*10DIGIT
XSP-Rate= "Rate=" #rate-value
```

The *rate-value* parameter MUST be an integer in the range from 1 through 4,294,967,295.

2.2.1.12.2 MaxBytes

This parameter contains a comma-separated list of values, each of which MUST specify the maximum buffer underflow, in bytes, that occurs at the acceleration rate given by the corresponding rate value in the *Rate* parameter.

The maximum buffer underflow specified by each value in the *MaxBytes* parameter occurs at the time specified by the corresponding time value in the *Time* parameter.

The syntax of the *MaxBytes* parameter is defined as follows:

```
maxbyte-value= 1*10DIGIT
XSP-MaxBytes= "MaxBytes=" #maxbyte-value
```

The *maxbyte-value* parameter MUST be an integer in the range from 1 through 4,294,967,295.

2.2.1.12.3 Time

This parameter contains comma-separated list of presentation times, in milliseconds, that specify when maximum buffer underflows will occur. Each presentation time value MUST specify when the maximum buffer underflow will occur when data is transmitted by using the acceleration rate given by the corresponding rate value in the *Rate* parameter.

Each of the time values MUST be expressed as an offset from the presentation time given by the *StartTime* parameter.

The syntax of the *Time* parameter is defined as follows:

```
time-value= 1*10DIGIT
XSP-Time= "Time=" #time-value
```

The *time-value* parameter MUST be an integer in the range from 0 through 4,294,967,295.

2.2.1.12.4 StartTime

This parameter MUST specify the lowest ASF presentation time value of all the ASF payloads that will be transmitted during the acceleration interval. The duration of the acceleration interval is given by the *LastTime* parameter.

Details about ASF presentation time are as specified in [ASF] section 5.2.3.

The syntax of the *StartTime* parameter is defined as follows:

```
XSP-STime= "StartTime=" 1*10DIGIT
```

The value of the *StartTime* parameter MUST be an integer in the range from 0 through 4,294,967,295.

2.2.1.12.5 LastTime

This parameter MUST specify the amount of data, represented as a length of time in milliseconds, that the server will transmit at the accelerated rate.

The syntax of the *LastTime* parameter is defined as follows:

```
XSP-LTime= "LastTime=" 1*10DIGIT
```

The value of the *LastTime* parameter MUST be an integer in the range from 0 through 4,294,967,295.

2.2.1.12.6 MaxDiffTime

This parameter specifies the presentation time of the ASF payload that has the maximum latency specified by the *MaxDiffSndTime* parameter. The parameter MUST be specified as an offset from the presentation time given by the *StartTime* parameter, and MUST be expressed in millisecond units.

Details of the ASF presentation time are as specified in [ASF], section 5.2.3.

The syntax of the *MaxDiffTime* parameter is defined as follows:

```
XSP-MaxDTime    = "MaxDiffTime=" 1*10DIGIT
```

The value of the *MaxDiffTime* parameter MUST be an integer in the range from 0 to 4,294,967,295 inclusive.

2.2.1.12.7 MaxDiffSndTime

This parameter specifies the maximum latency, in milliseconds, between the video and the audio streams during the acceleration interval. The duration of the acceleration interval is specified by the *LastTime* parameter.

If an ASF payload that contains audio data has a presentation time value that is lower than the presentation time value of the most recent ASF payload that contained video data, the audio stream is considered to lag behind the video stream. The *MaxDiffSndTime* parameter MUST specify the largest such lag, expressed as a non-negative number.

If the video stream consistently lags behind the audio stream during the acceleration interval, the *MaxDiffSndTime* parameter MUST specify the largest such lag, as a negative number.

If the content does not have both an audio and a video stream, the value of this parameter MUST be 0.

More information about ASF presentation time is specified in [ASF] section 5.2.3.

The syntax of the *MaxDiffSndTime* parameter is defined as follows:

```
XSP-MaxDSTime   = "MaxDiffSndTime=" [ "-" ] 1*10DIGIT
```

The value of the *MaxDiffSndTime* parameter MUST be an integer in the range from 0 through 2,147,483,647.

2.2.1.12.8 ByteRate

A comma-separated list of bit rate values, expressed in units of bytes per second. Each rate value MUST specify the average encoded bit rate of the content, computed over the acceleration time interval. The acceleration time interval is specified by the *LastTime* parameter.

Because the interval from which the average is computed typically is shorter than the entire content, the rate values may be different from the average bit rate of the content.

The syntax of the *ByteRate* parameter is defined as follows:

```
byterate-val    = 1*10DIGIT
XSP-ByteRate    = "ByteRate=" #byterate-val
```

The *byterate-val* parameter MUST be an integer in the range from 1 through 4,294,967,295.

2.2.2 Request Types

The Windows Media HTTP Streaming Protocol defines requests that a client can send to a server.

The requests from the client and the corresponding responses from the server are exchanged using HTTP request methods. Each request defined by the Windows Media HTTP Streaming Protocol is mapped to one of the following HTTP request methods: GET, POST, and OPTIONS.

This section defines the syntax of those requests using ABNF syntax, as specified in [RFC4234]. Any ABNF syntax rules that are not defined in [RFC4234] use the ABNF extensions that are defined as specified in [RFC1945] or [RFC2616].

In addition to complying with the ABNF syntax, all requests MUST also include a request line, all of the required headers, and one or more Pragma headers with the required Pragma header tokens. In the ABNF syntax for each request type, these components are indicated by the inclusion of "-Line", "-Header-REQ" and "-Token-REQ" in the associated ABNF rule name, respectively.

Pragma header tokens that are indicated as optional for HTTP 1.1 SHOULD NOT be sent by the client if HTTP-Version is "HTTP/1.0". Tokens that are optional for HTTP 1.1 include "-Token-OPT11" in the associated ABNF rule name.

Below are some common constructions used throughout this section:

```
HTTP-Header-Types       = *(( general-header
  / request-header
  / entity-header ) CRLF )
```

2.2.2.1 Describe Request

The purpose of the Describe request is to ask the server to send information about the content that is identified by the URL included in the request. A URL MUST be formed according to rules specified in [RFC1738].

The Describe request MUST be sent using the HTTP GET method. It SHOULD NOT include a Pragma header with the xPlayStrm (section 2.2.1.4.36) token, but it MAY include this token if the value is set to 0. The Describe request MUST NOT include the xPlayNextEntry (section 2.2.1.4.35) token on

a Pragma header, and the request MUST NOT include the pipeline-request (section 2.2.1.4.16) token on a Pragma header.

A server that receives a GET request MUST treat it as a Describe request if the Pragma header is missing or if all of the following conditions are satisfied: The value of the xPlayStrm token on the Pragma header is 0, and the xPlayNextEntry token is missing from the Pragma header, and the pipeline-request token is missing from the Pragma header, and the stream-switch-entry (section 2.2.1.4.27) token is missing from the Pragma header.

If the request succeeds, the server MUST respond to the request with the $H (Header) packet (section 2.2.3.5) for the content. If the URL identifies a server-side playlist, the $H packet MUST pertain to the first entry, or current entry, in the playlist. The response SHOULD also contain $P packets (section 2.2.3.7) and an $M packet (section 2.2.3.6), as appropriate.

The Describe request can be sent by using either the HTTP 1.0 protocol, as specified in [RFC1945], or the HTTP 1.1 protocol, as specified in [RFC2616]. The response MUST NOT use Chunked Transfer Coding even if HTTP 1.1 is used.

The syntax of the Describe request is defined as follows.

```
WMS-Describe-Request    = WMS-Describe-Req-Line
  WMS-DescReq-Headers CRLF

WMS-Describe-Req-Line   = "GET" SP Request-URI SP HTTP-Version CRLF

WMS-DescReq-Headers     = *( DescReq-Header-REQ
                           / DescReq-Header-OPT
                           / DescReq-Pragma
                           / HTTP-Header-Types )

DescReq-Header-REQ      = User-Agent                ; section 2.2.1.8

DescReq-Header-OPT      = Cookie                    ; section 2.2.1.3
                           / Supported              ; section 2.2.1.7
                           / X-Accept-Authentication  ; section 2.2.1.9
                           / X-Proxy-Client-Verb      ; section 2.2.1.11

DescReq-Pragma          = "Pragma: " #DescReq-Pragma-Types CRLF

DescReq-Pragma-Types    = DescReq-Token-REQ
                           / DescReq-Token-OPT
                           / DescReq-Token-OPT11

DescReq-Token-REQ       = no-cache                  ; section 2.2.1.4.12

DescReq-Token-OPT       = / client-id               ; section 2.2.1.4.5
                           / packet-pair-experiment   ; section 2.2.1.4.14
                           / proxy-client-agent       ; section 2.2.1.4.21
                           / request-context          ; section 2.2.1.4.23
                           / xClientGuid              ; section 2.2.1.4.33

DescReq-Token-OPT11     = pipeline-experiment       ; section 2.2.1.4.15
                           / version11-enabled        ; section 2.2.1.4.30
```

The syntax of the Describe response is defined as follows.

```
WMS-Describe-Response   = Status-Line
                           WMS-DescResp-Headers
                           CRLF
```

```
                          WMS-DescResp-Body

WMS-DescResp-Headers    = *( DescResp-Header-REQ
                          / DescResp-Header-OPT
                          / DescResp-Pragma
                          / HTTP-Header-Types )

DescResp-Header-REQ     = Cache-Control              ; section 2.2.1.1
                          / Server                   ; section 2.2.1.5

DescResp-Header-OPT     = Set-Cookie                 ; section 2.2.1.6
                          / Supported                ; section 2.2.1.7
                          / X-StartupProfile         ; section 2.2.1.12

DescResp-Pragma         = "Pragma: " #DescResp-Pragma-Types CRLF

DescResp-Pragma-Types   = DescResp-Token-REQ
                          / DescResp-Token-OPT
                          / DescResp-Token-OPT11

DescResp-Token-REQ      = no-cache                   ; section 2.2.1.4.12
                          / client-id                ; section 2.2.1.4.5
                          / features                 ; section 2.2.1.4.8

DescResp-Token-OPT      = timeout                    ; section 2.2.1.4.29
                          / version-info             ; section 2.2.1.4.31
                          / version-url              ; section 2.2.1.4.32
                          / packet-pair-experiment   ; section 2.2.1.4.14

DescResp-Token-OPT11    = pipeline-experiment        ; section 2.2.1.4.15

WMS-DescResp-Body       = [ 3<$P Packet-Pair packet> ] ; section 2.2.3.7
                          [ <$M Metadata packet> ]   ; section 2.2.3.6
                          <$H Header packet>         ; section 2.2.3.5
```

The following example shows a Describe request:

```
GET /welcome.asf HTTP/1.0
Accept: */*
User-Agent: NSPlayer/10.0.0.3802
Host: SampleServer
X-Accept-Authentication: Negotiate, NTLM, Digest, Basic
Pragma: no-cache
Pragma: packet-pair-experiment=1
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.predstrm, com.microsoft.wm.startupprofile
Pragma: xClientGUID={11223344-1122-1122-1122-AABBCCDDEEFF}
Accept-Language: en-US, *;q=0.1
122-1122-1122-AABBCCDDEEFF}
Accept-Language: en-US, *;q=0.1
```

2.2.2.2 GetContentInfo Request

The purpose of the GetContentInfo request is to retrieve cache-control information from the server without incurring the overhead of a Describe request.

The GetContentInfo request MUST be sent using the HTTP POST method. It MUST include a Content-Type (section 2.2.1.2) header with the application/x-wms-getcontentinfo content-type (as specified in 2.2.1.2.4).

The GetContentInfo request MUST NOT include any of the following tokens on a Pragma header: log-line (section 2.2.1.4.10), pipeline-request (section 2.2.1.4.16), stream-switch-entry (section 2.2.1.4.27), xKeepAliveInPause (section 2.2.1.4.34), xStopStrm (section 2.2.1.4.38).

The message body of the GetContentInfo request MUST NOT be empty, but the value MUST be ignored by the server. The message body SHOULD consist of only one byte, which SHOULD be chosen arbitrarily.

A server that receives a POST request MUST treat it as a GetContentInfo request if the Content-Type header specifies the application/x-wms-getcontentinfo content-type.

The response to a GetContentInfo request MUST specify the Cache-Control (section 2.2.1.1) header and MUST contain a zero-length message body.

The GetContentInfo request can be sent by using either the HTTP 1.0 protocol, as specified in [RFC1945], or the HTTP 1.1 protocol, as specified in [RFC2616].<45>

The syntax of the GetContentInfo request is defined as follows.

```
WMS-GCInfo-Request   = WMS-GCI-Req-Line
                       WMS-GCIReq-Headers
                       CRLF
                       WMS-GCIReq-Body

WMS-GCI-Req-Line    = "POST" SP Request-URI SP HTTP-Version CRLF

WMS-GCIReq-Headers      = *( GCIReq-Header-REQ
                         / GCIReq-Header-OPT
                         / GCIReq-Pragma
                         / HTTP-Header-Types )

GCIReq-Header-REQ       = Content-Type       ; section 2.2.1.2
                         / User-Agent        ; section 2.2.1.8

GCIReq-Header-OPT       = Cookie             ; section 2.2.1.3
                         / Supported         ; section 2.2.1.7
                         / X-Accept-Authentication ; section 2.2.1.9
                         / X-Proxy-Client-Verb ; section 2.2.1.11

GCIReq-Pragma           = "Pragma: " #GCIReq-Pragma-Types CRLF

GCIReq-Pragma-Types     = GCIReq-Token-OPT

GCIReq-Token-OPT        = client-id          ; section 2.2.1.4.5
                         / no-cache          ; section 2.2.1.4.12
                         / request-context   ; section 2.2.1.4.23
                         / xClientGuid       ; section 2.2.1.4.33

WMS-GCIReq-Body         = CHAR
```

The syntax of the GetContentInfo response is defined as follows.

```
WMS-GCInfo-Response = Status-Line
  WMS-GCIResp-Headers CRLF
```

```
WMS-GCIResp-Headers     = *( GCIResp-Header-REQ
                           / GCIResp-Header-OPT
                           / GCIResp-Pragma
                           / HTTP-Header-Types )

GCIResp-Header-REQ      = Cache-Control  ; section 2.2.1.1
                           / Server        ; section 2.2.1.5

GCIResp-Header-OPT      = Set-Cookie      ; section 2.2.1.6
                           / Supported     ; section 2.2.1.7

GCIResp-Pragma          = "Pragma: " #GCIResp-Pragma-Types CRLF

GCIResp-Pragma-Types    = GCIResp-Token-REQ
                           / GCIResp-Token-OPT

GCIResp-Token-REQ       = no-cache        ; section 2.2.1.4.12

GCIResp-Token-OPT       = client-id       ; section 2.2.1.4.5
                           / timeout       ; section 2.2.1.4.29
```

The following example shows a GetContentInfo request:

```
POST /welcome2.asf?WMCache=0 HTTP/1.1
User-Agent: WMCacheProxy/9.0.0.3177
Via: HTTP/1.1 WMCacheProxy (WMCacheProxy/9.0.0.3177)
Max-Forwards: 9
Accept-Charset: UTF-8, *;q=0.1
Pragma: xClientGUID={00000000-0000-0000-0000-000000000000}
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Content-Type: application/x-wms-getcontentinfo
Host: myhost:87
Content-Length: 1
Connection: Keep-Alive
```

2.2.2.3 KeepAlive Request

The purpose of the KeepAlive request is to prevent the server from timing out any state that it maintains for the streaming session. This request is useful if the server has stopped streaming (for example, as the result of the client sending a Stop request (section 2.2.2.10) and the client eventually intends to send a Play request (section 2.2.2.6) to ask the server to resume streaming.

If the client allows the server's session state to expire, it can become difficult to resume streaming from the same position that the server was stopped at, especially if the server was streaming from a server-side playlist with multiple entries (for example, consider the case of a server randomly generating a new server-side playlist each time a new session state is created).<46>

The server uses the timeout (section 2.2.1.4.29) token on the Pragma header to specify the interval at which it must receive a KeepAlive request to not time out an idle session. A session is considered idle when multimedia data is not being transmitted by the server and no requests are sent by the client.

The composition of the KeepAlive request is different depending on whether the pipelined mode or the non-pipelined mode of the protocol is used.

2.2.2.3.1 Non-Pipelined Mode

When the non-pipelined mode of the Windows Media HTTP Streaming Protocol is in use, the KeepAlive request (section 2.2.2.3) MUST use the HTTP POST method. The POST request MUST be sent on a TCP connection that is different from the one used by the server to transmit multimedia data to the client.

The KeepAlive request MUST include a Pragma header with the xKeepAliveInPause (section 2.2.1.4.34) token. The message body of the POST request MUST be zero length, and the Content-Type header MUST NOT be specified.

The KeepAlive request MUST NOT include any of the following tokens on a Pragma header: log-line (section 2.2.1.4.10), pipeline-request (section 2.2.1.4.16), stream-switch-entry (section 2.2.1.4.27), xStopStrm (section 2.2.1.4.38).

A server that receives a POST request MUST treat it as a KeepAlive request if the xKeepAliveInPause token is present on a Pragma header.

The response to a KeepAlive request MUST contain a zero-length message body.

The KeepAlive request can be sent by using either the HTTP 1.0 protocol, as specified in [RFC1945], or the HTTP 1.1 protocol, as specified in [RFC2616].

The syntax of the KeepAlive request using the non-pipelined mode is defined as follows.

```
WMS-KeepAlive10-Request = WMS-KeepAlive10-Req-Line
   WMS-KeepReq10-Headers CRLF

WMS-KeepAlive10-Req-Line = "POST" SP Request-URI SP HTTP-Version CRLF

WMS-KeepReq10-Headers = *( KeepReq10-Header-REQ
                         / KeepReq10-Header-OPT
                         / KeepReq10-Pragma
                         / HTTP-Header-Types )

KeepReq10-Header-REQ = "Content-Length: 0"
                         / User-Agent    ; section 2.2.1.8

KeepReq10-Header-OPT     = Supported       ; section 2.2.1.7
                         / X-Accept-Authentication; section 2.2.1.9

KeepReq10-Pragma         = "Pragma: " #KeepReq10-Pragma-Types CRLF

KeepReq10-Pragma-Types   = KeepReq10-Token-REQ
                         / KeepReq10-Token-OPT

KeepReq10-Token-REQ      = client-id       ; section 2.2.1.4.5
                         / no-cache        ; section 2.2.1.4.12
                         / xKeepAliveInPause ; section 2.2.1.4.34

KeepReq10-Token-OPT      = request-context ; section 2.2.1.4.23
                         / xClientGuid     ; section 2.2.1.4.33
```

The syntax of the KeepAlive response is defined as follows.

```
WMS-KeepAlive-Response  = Status-Line
                          WMS-KeepResp-Headers CRLF

WMS-KeepResp-Headers    = *( KeepResp-Header-REQ
                          / KeepResp-Header-OPT
                          / KeepResp-Pragma
                          / HTTP-Header-Types )

KeepResp-Header-REQ     = Cache-Control ; section 2.2.1.1
                          / "Content-Length: 0"
                          / Server       ; section 2.2.1.5


KeepResp-Header-OPT     = Supported      ; section 2.2.1.7

KeepResp-Pragma         = "Pragma: " #KeepResp-Pragma-Types CRLF

KeepResp-Pragma-Types   = KeepResp-Token-REQ
                          / KeepResp-Token-OPT

KeepResp-Token-REQ      = no-cache       ; section 2.2.1.4.12

KeepResp-Token-OPT      = client-id      ; section 2.2.1.4.5
                          / timeout      ; section 2.2.1.4.29
```

The following example shows a KeepAlive request in the non-pipelined mode:

```
POST http://content.microsoft.net/movies/foo.asf HTTP/1.0
Accept: */*
User-Agent: NSPlayer/10.0.0.3802
Host: SampleServer
Pragma: xClientGUID={11223344-1122-1122-1122-AABBCCDDEEFF}
X-Accept-Authentication: Negotiate, MS-NLMP, Digest, Basic
Pragma: xKeepAliveInPause=1
Pragma: client-id= 4259367329
Content-Length: 0
Connection: Keep-Alive
```

2.2.2.3.2 Pipelined Mode

When the pipelined mode of the protocol is in use, and the TCP connection that is used by the server to transmit multimedia data to the client has been closed, the KeepAlive request (section 2.2.2.3) MUST be sent in the same way as if the non-pipelined mode is used, as specified in section 2.2.2.3.1.

Otherwise, the KeepAlive request MUST use the HTTP OPTIONS method, and the request MUST be sent on the same TCP connection that is used by the server to transmit multimedia data to the client.

The KeepAlive request MUST include a Pragma header with the xKeepAliveInPause (section 2.2.1.4.34) token.

A server that receives an OPTIONS request MUST treat it as a KeepAlive request if the xKeepAliveInPause token is present on a Pragma header.

The syntax of the KeepAlive request using OPTIONS is defined as follows.

```
WMS-KeepAlive11-Request    = WMS-KeepAlive11-Req-Line
                                    WMS-KeepReq11-Headers CRLF

WMS-KeepAlive11-Req-Line   = "OPTIONS" SP "*" SP "HTTP/1.1" CRLF

WMS-KeepReq11-Headers    = User-Agent
                       / "Pragma: " client-id CRLF ; section 2.2.1.4.5
                       / HTTP-Header-Types
```

The syntax of the KeepAlive response using OPTIONS is defined as follows.

```
WMS-KeepAlive11-Response = WMS-KeepAlive-Response ; section 2.2.2.3.1
```

The following example shows a KeepAlive request using OPTIONS:

```
OPTIONS * HTTP/1.1
User-Agent: NSPlayer/9.0.0.1234
Host: SampleServer
Pragma: client-id=3333510490
```

The following example shows the server response:

```
HTTP 1.1 200 OK
Server: Cougar/9.00.00.1234
Date: Mon, 10 Mar 2003 08:09:46 GMT
Public: GET, POST, OPTIONS
Allow: OPTIONS
Pragma: client-id=3333510490, timeout=60000
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
           com.microsoft.wm.predstrm, com.microsoft.wm.fastcache
Content-Length: 0
```

2.2.2.4 Log Request

The purpose of the Log request is to submit statistics about the streamed content to the server. The request specifies parameters such as streaming quality and packet transmission statistics.

The Log request MUST be sent using the HTTP POST method. It MUST include either a Pragma header with the log-line (section 2.2.1.4.10) token, or a Content-Type header with the application/x-wms-LogStats content-type (as specified in 2.2.1.2.5).

The Log request MUST NOT include any of the following tokens on a Pragma header: pipeline-request (section 2.2.1.4.16), stream-switch-entry (section 2.2.1.4.27), xKeepAliveInPause (section 2.2.1.4.34), xStopStrm (section 2.2.1.4.38).

If the log-line token is specified on a Pragma header in the request, the message body of the POST request MUST be empty. If it is empty, the Content-Type header MUST either be omitted or specify the content-type text/plain (section 2.2.1.2.7).

If the Content-Type header with the application/x-wms-LogStats content-type is specified in the request, then the message body of the POST request MUST contain a logging message in XML format, as specified in [MS-WMLOG].

If the pipelined mode of the protocol is used, the Log request SHOULD be sent on the connection that the server is using for transmitting the multimedia data to the client.

A server that receives a POST request MUST treat it as a Log request if the log-line token is present on a Pragma header or if the Content-Type header specifies the application/x-wms-LogStats content-type.

The response to a Log request MUST contain a zero-length message body.

The Log request can be sent by using either the HTTP 1.0 protocol, as specified in [RFC1945], or the HTTP 1.1 protocol, as specified in [RFC2616].

The syntax of the Log request is defined as follows.

```
WMS-Log-Request         = WMS-Log-Req-Line
                          WMS-LogReq-Headers
                          CRLF
                          WMS-LogReq-Body

WMS-Log-Req-Line        = "POST" SP Request-URI SP HTTP-Version CRLF

WMS-LogReq-Headers      = *( LogReq-Header-REQ
                          / LogReq-Header-OPT
                          / LogReq-Pragma
                          / HTTP-Header-Types )

LogReq-Header-REQ       = User-Agent          ; section 2.2.1.8

LogReq-Header-OPT       = Cookie              ; section 2.2.1.3
Supported           ; section 2.2.1.7
                          / X-Accept-Authentication ; section 2.2.1.9
                          / X-Proxy-Client-Verb ; section 2.2.1.11

LogReq-Pragma           = "Pragma: " #LogReq-Pragma-Types CRLF

LogReq-Pragma-Types     = LogReq-Token-REQ
                          / LogReq-Token-OPT

LogReq-Token-REQ        = client-id           ; section 2.2.1.4.5
                          / no-cache          ; section 2.2.1.4.12

LogReq-Token-OPT        = log-line            ; section 2.2.1.4.10
                          / proxy-client-agent ; section 2.2.1.4.21
                          / request-context   ; section 2.2.1.4.23
                          / xClientGuid       ; section 2.2.1.4.33

WMS-LogReq-Body         = [ <XML-format-logging-info> ] ; [MS-WMLOG]
```

The syntax of the Log response is defined as follows.

```
WMS-Log-Response    = Status-Line
  WMS-LogResp-Headers CRLF

WMS-LogResp-Headers     = *( LogResp-Header-REQ
                           / LogResp-Header-OPT
                           / LogResp-Pragma
                           / HTTP-Header-Types )

LogResp-Header-REQ      = Cache-Control          ; section 2.2.1.1
                           / Server               ; section 2.2.1.5

LogResp-Header-OPT      = Supported              ; section 2.2.1.7

LogResp-Pragma          = "Pragma: " #LogResp-Pragma-Types CRLF

LogResp-Pragma-Types    = LogResp-Token-REQ
                           / LogResp-Token-OPT

LogResp-Token-REQ       = no-cache             ; section 2.2.1.4.12

LogResp-Token-OPT       = client-id            ; section 2.2.1.4.5
                           / timeout             ; section 2.2.1.4.28
```

The following example shows a Log request:

```
POST /welcome.asf HTTP/1.0
Accept: */*
User-Agent: NSPlayer/10.0.0.3802
Host: SampleServer
Pragma: xClientGUID={11223344-1122-1122-1122-AABBCCDDEEFF}
X-Accept-Authentication: Negotiate, MS-NLMP, Digest, Basic
Pragma: client-id=8675309
Content-Length: 3273
Content-Type: application/x-wms-LogStats;charset=UTF-8
```

2.2.2.5 Pipeline Request

The purpose of the Pipeline request is to determine if the pipelined mode of operation of the protocol is possible.

A client MUST NOT send a Pipeline request to a server unless the response to the Describe request (section 2.2.2.1) specified the pipeline-experiment (section 2.2.1.4.15) token on a Pragma header, and its value was 1.

If a client sends a Pipeline request, a second Pipeline request MUST be sent immediately after the first Pipeline request without waiting for the response to the first Pipeline request to be completely received by the client. In other words, the two requests are "pipelined", as specified in [RFC2616] section 8.1.2.2.

If a client sends a Pipeline request, the first Pipeline request MUST be sent after the first Describe request and before the first Play (section 2.2.2.6) request.

The Pipeline request MUST be sent using the HTTP GET method. It MUST include a Pragma header with the pipeline-request (section 2.2.1.4.16) token on a Pragma header. The Pipeline request (section 2.2.2.5) MUST NOT include the xPlayNextEntry (section 2.2.1.4.35) token on a Pragma header, and the request MUST NOT include the xPlayStrm (section 2.2.1.4.36) token on a Pragma header.

A server that receives a GET request MUST treat it as a Pipeline request if the pipeline-request token is present on the Pragma header.

The server MUST respond to the first Pipeline request in accordance to the ABNF syntax for WMS-Pipeline-Response-1, below, and it MUST respond to the second Pipeline request in accordance to the ABNF syntax for WMS-Pipeline-Response-2.

The Pipeline request and response MUST be sent using the HTTP 1.1 protocol, as specified in [RFC2616].<47>

The response to the first Pipeline request MUST use Chunked Transfer Coding, as specified in [RFC2616] section 3.6.1.

The response to the second Pipeline request MUST have a zero-size message body and MUST include a Pragma header with the pipeline-result (section 2.2.1.4.17) token.

The syntax of the Pipeline request is defined as follows:

```
WMS-Pipeline-Request     = WMS-Pipeline-Req-Line
                                WMS-PipeReq-Headers CRLF

WMS-Pipeline-Req-Line    = "GET" SP Request-URI SP "HTTP/1.1" CRLF

WMS-PipeReq-Headers      = *( User-Agent      ; section 2.2.1.8
                           / "Connection: keep-alive" CRLF
                           / PipeReq-Pragma
                           / HTTP-Header-Types )

PipeReq-Pragma           = "Pragma: " #PipeReq-Token-REQ CRLF

PipeReq-Token-REQ        = client-id          ; section 2.2.1.4.5
                           / no-cache         ; section 2.2.1.4.12
                           / pipeline-request ; section 2.2.1.4.16
```

The syntax of the first Pipeline response is defined as follows.

```
WMS-Pipeline-Response-1 = Status-Line
                           WMS-PipeResp1-Headers
                           CRLF
                           WMS-PipeResp1-Body

WMS-PipeResp1-Headers    = *( PipeResp1-Header-REQ
                           / PipeResp1-Header-OPT
                           / PipeResp1-Pragma
                           / HTTP-Header-Types )

PipeResp1-Header-REQ     = Cache-Control    ; section 2.2.1.1
                           / "Connection: keep-alive" CRLF
                           / Server      ; section 2.2.1.5
```

```
                          / "Transfer-Encoding: chunked" CRLF

PipeResp1-Header-OPT    = Supported         ; section 2.2.1.7

PipeResp1-Pragma        = "Pragma: " #PipeResp1-Pragma-Types CRLF

PipeResp1-Pragma-Types  = PipeResp1-Token-REQ
                          / PipeResp1-Token-OPT

PipeResp1-Token-REQ     = client-id         ; section 2.2.1.4.5
                          / no-cache        ; section 2.2.1.4.12


PipeResp1-Token-OPT     = timeout           ; section 2.2.1.4.29

WMS-PipeResp1-Body     = "4" CRLF
                         "$T" %x00 %x00
                         CRLF
```

The syntax of the second Pipeline response is defined as follows.

```
WMS-Pipeline-Response-2 = "0" CRLF
                           CRLF
                           Status-Line
                           WMS-PipeResp2-Headers
                           CRLF

WMS-PipeResp2-Headers    = *( PipeResp2-Header-REQ
                          / PipeResp2-Header-OPT
                          / PipeResp2-Pragma
                          / HTTP-Header-Types )

PipeResp2-Header-REQ     = Cache-Control     ; section 2.2.1.1
                          / "Content-Length: 0" CRLF
                          / Server      ; section 2.2.1.5

PipeResp2-Header-OPT     = Supported         ; section 2.2.1.7

PipeResp2-Pragma         = "Pragma: " #PipeResp2-Pragma-Types CRLF

PipeResp2-Pragma-Types   = PipeResp2-Token-REQ
                          / PipeResp2-Token-OPT

PipeResp2-Token-REQ      = client-id         ; section 2.2.1.4.5
                          / no-cache        ; section 2.2.1.4.12
                          / pipeline-result ; section 2.2.1.4.17

PipeResp2-Token-OPT      = timeout           ; section 2.2.1.4.29
```

2.2.2.6 Play Request

The purpose of the Play request is to request the server to start streaming the content that is identified by the URL, as specified in [RFC1738], and which is included in the request.

If the client has previously sent a Describe request (section 2.2.2.1), the client will have obtained at least a $H packet (section 2.2.3.5) and perhaps also a $M packet (section 2.2.3.6) and $P packets (section 2.2.3.7). Based on the information provided by those packets, the client can include the stream-switch-entry (section 2.2.1.4.27) token on the Pragma header, specifying the streams of the content that the client wants to receive.

The Play request MUST be sent using the HTTP GET method. It MUST include a Pragma header with the xPlayStrm (section 2.2.1.4.36) token, and the value of this token MUST be 1. The Play request MUST NOT include the xPlayNextEntry (section 2.2.1.4.35) token on a Pragma header, and the request MUST NOT include the pipeline-request (section 2.2.1.4.16) token on a Pragma header.

A server that receives a GET request MUST treat it as a Play request if the xPlayStrm token is present on Pragma header, and the value of the token is 1.

If the request succeeds, the server MUST respond to the request in accordance to the ABNF syntax in the following code. If the URL identifies a server-side playlist, the first $H packet MUST pertain to the first entry, or current entry, in the playlist.

The Play request can be sent by using either the HTTP 1.0 protocol, as specified in [RFC1945], or the HTTP 1.1 protocol, as specified in [RFC2616].

If the pipelined mode of the protocol is used, the Play response MUST use HTTP 1.1, as specified in [RFC2616], and Chunked Transfer Coding, as specified in [RFC2616] section 3.6.1.

The syntax of the Play request is defined as follows:

```
WMS-Play-Request        = WMS-Play-Req-Line
                          WMS-PlayReq-Headers CRLF

WMS-Play-Req-Line       = "GET" SP Request-URI SP HTTP-Version CRLF

WMS-PlayReq-Headers     = *( PlayReq-Header-REQ
                          / PlayReq-Header-OPT
                          / PlayReq-Pragma
                          / HTTP-Header-Types )

PlayReq-Header-REQ      = User-Agent              ; section 2.2.1.8

PlayReq-Header-OPT      = Cookie                  ; section 2.2.1.3
                          / Supported             ; section 2.2.1.7
                          / X-Accept-Authentication ; section 2.2.1.9
                          / X-Proxy-Client-Verb   ; section 2.2.1.11

PlayReq-Pragma          = "Pragma: " #PlayReq-Pragma-Types CRLF

PlayReq-Pragma-Types    = PlayReq-Token-REQ
                          / PlayReq-Token-OPT
                          / PlayReq-Token-OPT11

PlayReq-Token-REQ       = no-cache                ; section 2.2.1.4.12
                          / packet-num            ; section 2.2.1.4.13
                          / rate                  ; section 2.2.1.4.22
                          / stream-offset         ; section 2.2.1.4.25
                          / stream-time           ; section 2.2.1.4.28
                          / xPlayStrm             ; section 2.2.1.4.36

PlayReq-Token-OPT       = AccelBW                 ; section 2.2.1.4.1
```

```
                          / AccelDuration        ; section 2.2.1.4.2
                          / BurstBW              ; section 2.2.1.4.3
                          / BurstDuration        ; section 2.2.1.4.4
                          / client-id            ; section 2.2.1.4.5
                          / client-lag           ; section 2.2.1.4.6
                          / LinkBW               ; section 2.2.1.4.9
                          / max-duration         ; section 2.2.1.4.11
                          / packet-pair-experiment ; section 2.2.1.4.14
                          / playlist-gen-id      ; section 2.2.1.4.18
                          / playlist-seek-id     ; section 2.2.1.4.19
                          / pl-offset            ; section 2.2.1.4.20
                          / request-context      ; section 2.2.1.4.23
                          / speed                ; section 2.2.1.4.24
                          / stream-switch-count  ; section 2.2.1.4.26
                          / stream-switch-entry  ; section 2.2.1.4.27
                          / xClientGuid          ; section 2.2.1.4.33

PlayReq-Token-OPT11     = version11-enabled      ; section 2.2.1.4.30
```

The syntax of the Play response is defined as follows:

```
WMS-Play-Response   = Status-Line
                        WMS-PlayResp-Headers
                        CRLF
                        WMS-PlayResp-Body

WMS-PlayResp-Headers    = *( PlayResp-Header-REQ
                        / PlayResp-Header-OPT
                        / PlayResp-Pragma
                        / HTTP-Header-Types )

PlayResp-Header-REQ     = Cache-Control          ; section 2.2.1.1
                        / Server                 ; section 2.2.1.5

PlayResp-Header-OPT     = Set-Cookie             ; section 2.2.1.6
                        / Supported              ; section 2.2.1.7
                        / X-StartupProfile       ; section 2.2.1.12

PlayResp-Pragma         = "Pragma: " #PlayResp-Pragma-Types CRLF

PlayResp-Pragma-Types   = PlayResp-Token-REQ
                         / PlayResp-Token-OPT

PlayResp-Token-REQ      = client-id              ; section 2.2.1.4.5
                         / features              ; section 2.2.1.4.8
                         / no-cache              ; section 2.2.1.4.12

PlayResp-Token-OPT      = AccelBW                ; section 2.2.1.4.1
                         / AccelDuration         ; section 2.2.1.4.2
                         / BurstBW               ; section 2.2.1.4.3
                         / BurstDuration         ; section 2.2.1.4.4
                         / expect-new-header     ; section 2.2.1.4.7
                         / packet-pair-experiment; section 2.2.1.4.14
                         / speed                 ; section 2.2.1.4.24
                         / timeout               ; section 2.2.1.4.29
```

```
                        / xResetStrm            ; section 2.2.1.4.37

Header-Packets= [<$M Metadata packet type>]    ; section 2.2.3.6
                  <$H Header packet type>        ; section 2.2.3.5

Stream-Change= <$C Change packet type>        ; section 2.2.3.2
                Header-Packets

Playlist-Entry= *( <$D Data packet type> )     ; section 2.2.3.3
                <$E EOS packet type>            ; section 2.2.3.4

WMS-PlayResp-Body= Header-Packets
                    Playlist-Entry
                    *( Stream-Change Playlist-Entry )
```

The following example shows a Play request:

```
GET /welcome.asf HTTP/1.0
Accept: */*
User-Agent: NSPlayer/10.0.0.3802
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest, Basic
Pragma: no-cache,rate=1.000,stream-time=0,
stream-offset=4294967295:4294967295,
packet-num=4294967295,max-duration=0
Pragma: xPlayStrm=1
Pragma: client-id=8675309
Pragma: LinkBW=2147483647, AccelBW=2147483647, AccelDuration=18000
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.predstrm, com.microsoft.wm.startupprofile
Pragma: xClientGUID={11223344-1122-1122-1122-AABBCCDDEEFF}
Pragma: stream-switch-count=2
Pragma: stream-switch-entry=ffff:1:0 ffff:2:0
Accept-Language: en-us, *;q=0.1
```

2.2.2.7 PlayNextEntry Request

The purpose of the PlayNextEntry request is to request that the server start streaming the current entry in the server-side playlist. If the server has already started to stream the next current entry, the PlayNextEntry request serves as a confirmation to the server that the client is now receiving the current entry. Also, the PlayNextEntry request allows the client to change the streams that are selected by using the stream-switch-entry (section 2.2.1.4.27) token on the Pragma header.

The PlayNextEntry request MUST NOT be sent when the non-pipelined mode of the protocol is used.<48>

The PlayNextEntry request MUST be sent using the HTTP GET method, and it MUST be sent on the same connection that the server uses for transmitting multimedia data to the client. The request MUST include a Pragma header with the xPlayNextEntry (section 2.2.1.4.35) token.

The PlayNextEntry request SHOULD NOT include the xPlayStrm (section 2.2.1.4.36) token on a Pragma header. If the xPlayStrm token is included, its value MUST be 0.

The PlayNextEntry request MUST NOT include the pipeline-request (section 2.2.1.4.16) token on a Pragma header.

A server that receives a GET request MUST treat it as a PlayNextEntry request if the xPlayNextEntry token is present on a Pragma header.

If the request succeeds, the server MUST respond to the request in accordance to the ABNF syntax in the following code. If the URL identifies a server-side playlist, the first $H packet (section 2.2.3.5) MUST pertain to the current entry in the playlist.

The PlayNextEntry response MUST use the HTTP 1.1 protocol, as specified in [RFC2616], and Chunked Transfer Coding, as specified in [RFC2616] section 3.6.1.

The syntax of the PlayNextEntry request is defined as follows.

```
WMS-PlayNext-Request     = WMS-Play-Req-Line
                           WMS-PlayReq-Headers CRLF

WMS-Play-Req-Line        = "GET" SP Request-URI SP "HTTP/1.1" CRLF

WMS-PlayReq-Headers      = *( PlayReq-Header-REQ
                           / PlayReq-Header-OPT
                           / PlayReq-Pragma
                           / HTTP-Header-Types )

PlayReq-Header-REQ       = User-Agent                ; section 2.2.1.8

PlayReq-Header-OPT       = Cookie                  ; section 2.2.1.3
                           / Supported               ; section 2.2.1.7
                           / X-Accept-Authentication; section 2.2.1.9
                           / X-Proxy-Client-Verb     ; section 2.2.1.11

PlayReq-Pragma           = "Pragma: " #PlayReq-Pragma-Types CRLF

PlayReq-Pragma-Types     = PlayReq-Token-REQ
                           / PlayReq-Token-OPT
                           / PlayReq-Token-OPT11

PlayReq-Token-REQ        = client-id               ; section 2.2.1.4.5
                           / xPlayNextEntry        ; section 2.2.1.4.35

PlayReq-Token-OPT        = AccelBW                 ; section 2.2.1.4.1
                           / AccelDuration         ; section 2.2.1.4.2
                           / BurstBW               ; section 2.2.1.4.3
                           / BurstDuration         ; section 2.2.1.4.4
                           / client-lag            ; section 2.2.1.4.6
                           / LinkBW                ; section 2.2.1.4.9
                           / max-duration          ; section 2.2.1.4.11
                           / no-cache              ; section 2.2.1.4.12
                           / packet-num            ; section 2.2.1.4.13
                           / packet-pair-experiment; section 2.2.1.4.14
                           / playlist-gen-id       ; section 2.2.1.4.18
                           / playlist-seek-id      ; section 2.2.1.4.19
                           / pl-offset             ; section 2.2.1.4.20
                           / proxy-client-agent    ; section 2.2.1.4.21
                           / rate                  ; section 2.2.1.4.22
                           / request-context       ; section 2.2.1.4.23
                           / speed                 ; section 2.2.1.4.24
```

```
                          / stream-offset        ; section 2.2.1.4.25
                          / stream-switch-count  ; section 2.2.1.4.26
                          / stream-switch-entry  ; section 2.2.1.4.27
                          / stream-time          ; section 2.2.1.4.28
                          / xClientGuid          ; section 2.2.1.4.33

PlayReq-Token-OPT11     = version11-enabled      ; section 2.2.1.4.30
```

The syntax of the PlayNextEntry response is defined as follows.

```
WMS-PlayNext-Response  = Status-Line
                         WMS-PlayResp-Headers
                         CRLF
                         WMS-PlayResp-Body

WMS-PlayResp-Headers    = *( PlayResp-Header-REQ
                         / PlayResp-Header-OPT
                         / PlayResp-Pragma
                         / HTTP-Header-Types )

PlayResp-Header-REQ     = Cache-Control           ; section 2.2.1.1
                         / Server                 ; section 2.2.1.5

PlayResp-Header-OPT     = Set-Cookie              ; section 2.2.1.6
                         / Supported              ; section 2.2.1.7
                         / X-StartupProfile       ; section 2.2.1.12

PlayResp-Pragma         = "Pragma: " #PlayResp-Pragma-Types CRLF

PlayResp-Pragma-Types   = PlayResp-Token-REQ
                         / PlayResp-Token-OPT

PlayResp-Token-REQ      = client-id               ; section 2.2.1.4.5
                         / features               ; section 2.2.1.4.8
                         / no-cache               ; section 2.2.1.4.12

PlayResp-Token-OPT      = AccelBW                 ; section 2.2.1.4.1
                         / AccelDuration          ; section 2.2.1.4.2
                         / BurstBW                ; section 2.2.1.4.3
                         / BurstDuration          ; section 2.2.1.4.4
                         / expect-new-header      ; section 2.2.1.4.7
                         / packet-pair-experiment; section 2.2.1.4.14
                         / speed                  ; section 2.2.1.4.24
                         / timeout                ; section 2.2.1.4.29
                         / xResetStrm             ; section 2.2.1.4.37

Header-Packets = [ <$M Metadata packet type> ]   ; section 2.2.3.6
                 <$H Header packet type>         ; section 2.2.3.5

Stream-Change  = <$C Change packet type>         ; section 2.2.3.2
                 Header-Packets

Playlist-Entry = *( <$D Data packet type> )      ; section 2.2.3.3
                 <$E EOS packet type>            ; section 2.2.3.4
```

```
WMS-PlayResp-Body = ( Header-Packets Playlist-Entry
                    *( Stream-Change Playlist-Entry ) )
                    / ( Playlist-Entry
                    *( Stream-Change Playlist-Entry ) )
                    / *( Stream-Change Playlist-Entry )
```

The following example shows a PlayNextEntry request:

```
GET /simple.wsx?WMCache=0 HTTP/1.1
Accept: */*
User-Agent: NSPlayer/9.0.0.2833
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: client-id=2380927133
Pragma: xPlayNextEntry=1
Pragma: no-cache,rate=1.000
Pragma: client-lag=0
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
           com.microsoft.wm.predstrm
Accept-Language: en-us, *;q=0.1
Pragma: stream-switch-count=4
Pragma: stream-switch-entry=ffff:1:0 ffff:2:2 ffff:4:2 ffff:5:0
```

2.2.2.8 SelectStream Request

The purpose of the SelectStream request is to request that the server turn on or off specified streams in the content (or current playlist entry if server-side playlists are used).

If the pipelined mode of the protocol is used, and the SelectStream request is sent on the connection that the server is using for transmitting the multimedia data to the client, then the SelectStream request MUST be sent using the HTTP GET method. Otherwise, the SelectStream request MUST be sent using the HTTP POST method.

The SelectStream request MUST include a Pragma header with the stream-switch-entry (section 2.2.1.4.27) token. If the POST method is used, the message body MUST be zero length, and the Content-Type header MUST NOT be specified.

The SelectStream request MUST NOT include any of the following tokens on a Pragma header: log-line (section 2.2.1.4.10), pipeline-request (section 2.2.1.4.16), xKeepAliveInPause (section 2.2.1.4.34), xPlayNextEntry (section 2.2.1.4.35), xPlayStrm (section 2.2.1.4.36), xStopStrm (section 2.2.1.4.38).

A server that receives a POST or a GET request MUST treat it as a SelectStream request if the stream-switch-entry token is present on a Pragma header.

If the non-pipelined mode of the protocol is used, the response to a SelectStream request MUST contain a zero-length message body.

If the pipelined mode of the protocol is used, and the SelectStream request was sent on the connection that the server is using for transmitting the multimedia data to the client, the response to the SelectStream request SHOULD use Chunked Transfer Coding, as specified in [RFC2616] section 3.6.1. Otherwise, the response MUST contain a zero-length message body.

The SelectStream request can be sent using either the HTTP 1.0 protocol, as specified in [RFC1945], or the HTTP 1.1 protocol, as specified in [RFC2616].

The syntax of the SelectStream request is defined as follows.

```
WMS-SelectStrm-Request= WMS-SelectStrm-Req-Line
                        WMS-StrmSelReq-Headers CRLF

WMS-SelectStrm-Req-Line= ( "GET" / "POST" )
                        SP Request-URI SP HTTP-Version CRLF

WMS-SelStrmReq-Headers= *( SelStrmReq-Header-REQ
                        / SelStrmReq-Header-OPT
                        / SelStrmReq-Pragma
                        / HTTP-Header-Types )

SelStrmReq-Header-REQ   = User-Agent               ; section 2.2.1.8

SelStrmReq-Header-OPT   = Cookie                   ; section 2.2.1.3
                        / Supported                ; section 2.2.1.7
                        / X-Accept-Authentication; section 2.2.1.9
                        / X-Proxy-Client-Verb      ; section 2.2.1.11

SelStrmReq-Pragma       = "Pragma: " #SelStrmReq-Pragma-Types CRLF

SelStrmReq-Pragma-Types= SelStrmReq-Token-REQ
                        / SelStrmReq-Token-OPT
                        / SelStrmReq-Token-OPT11

SelStrmReq-Token-REQ    = client-id              ; section 2.2.1.4.5
                        / no-cache               ; section 2.2.1.4.12
                        / stream-switch-count    ; section 2.2.1.4.26
                        / stream-switch-entry    ; section 2.2.1.4.27

SelStrmReq-Token-OPT    = client-lag             ; section 2.2.1.4.6
                        / proxy-client-agent     ; section 2.2.1.4.21
                        / request-context        ; section 2.2.1.4.23
                        / xClientGuid            ; section 2.2.1.4.33

SelStrmReq-Token-OPT11= version11-enabled      ; section 2.2.1.4.30
```

The syntax of the SelectStream response is defined as follows.

```
WMS-SelectStrm-Response= Status-Line
                        WMS-SelStrmResp-Headers
                        CRLF
                        WMS-SelStrmResp-Body

WMS-SelStrmResp-Headers = *( SelStrmResp-Header-REQ
                        / SelStrmResp-Header-OPT
                        / SelStrmResp-Pragma
                        / HTTP-Header-Types )

SelStrmResp-Header-REQ  = Cache-Control            ; section 2.2.1.1
                        / Server                   ; section 2.2.1.5
```

```
SelStrmResp-Header-OPT  = Set-Cookie                ; section 2.2.1.6
                          / Supported               ; section 2.2.1.7

SelStrmResp-Pragma      = "Pragma: " #SelStrmResp-Pragma-Types CRLF

SelStrmResp-Pragma-Types = SelStrmResp-Token-REQ
                           / SelStrmResp-Token-OPT

SelStrmResp-Token-REQ   = no-cache                  ; section 2.2.1.4.12

SelStrmResp-Token-OPT   = AccelBW                   ; section 2.2.1.4.1
                          / AccelDuration           ; section 2.2.1.4.2
                          / BurstBW                 ; section 2.2.1.4.3
                          / BurstDuration           ; section 2.2.1.4.4
                          / client-id               ; section 2.2.1.4.5
                          / speed                   ; section 2.2.1.4.24
                          / timeout                 ; section 2.2.1.4.29

WMS-SelStrmResp-Body    = *( <$D Data packet type> ); section 2.2.3.3
```

The following example shows a SelectStream request:

```
Accept: */*
User-Agent: NSPlayer/10.0.0.3802
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest, Basic
Pragma: no-cache
Pragma: client-id=8675309
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.predstrm, com.microsoft.wm.startupprofile
Pragma: xClientGUID={11223344-1122-1122-1122-AABBCCDDEEFF}
Pragma: stream-switch-count=2
Pragma: stream-switch-entry=2:ffff:0
Accept-Language: en-us, *;q=0.1
```

2.2.2.9 SendEvent Request

The purpose of the SendEvent request is to submit a remote event to the server. The most common remote event is "remote-log", which specifies rendering statistics independently of streaming statistics. It is possible for clients to send "remote-log" events to a server after having played content entirely from a cache without having had a streaming connection to the server. For more information about remote events, see section 2.2.5.

The SendEvent request MUST be sent using the HTTP POST method. It MUST include a Content-Type header with the application/x-wms-sendevent (section 2.2.1.2.6) content-type.

The SendEvent request MUST NOT include any of the following tokens on a Pragma header: log-line (section 2.2.1.4.10), pipeline-request (section 2.2.1.4.16), stream-switch-entry (section 2.2.1.4.27), xKeepAliveInPause (section 2.2.1.4.34), xStopStrm (section 2.2.1.4.38).

The remote event included in the message body of the SendEvent request MUST be of one of the types previously specified by the server using the x-wms-event-subscription (section 2.2.1.1.10) directive on the Cache-Control header.

If the pipelined mode of the protocol is used, the SendEvent request SHOULD be sent on the connection that the server is using for transmitting the multimedia data to the client.

A server that receives a POST request MUST treat it as a SendEvent request if the Content-Type header specifies the application/x-wms-sendevent (section 2.2.1.2.6) content-type.

The response to a SendEvent request MUST contain a zero-length message body.

The SendEvent request can be sent by using either the HTTP 1.0 protocol, as specified in [RFC1945], or the HTTP 1.1 protocol, as specified in [RFC2616].<49>

The syntax of the SendEvent request is defined as follows:

```
WMS-SendEvent-Request  = WMS-SE-Req-Line
                         WMS-SEReq-Headers
                         CRLF
                         WMS-SEReq-Body

WMS-SE-Req-Line        = "POST" SP Request-URI SP HTTP-Version CRLF

WMS-SEReq-Headers      = *( SEReq-Header-REQ
                         / SEReq-Header-OPT
                         / SEReq-Pragma
                         / HTTP-Header-Types )

SEReq-Header-REQ       = Content-Type              ; section 2.2.1.2
                         / User-Agent              ; section 2.2.1.8

SEReq-Header-OPT       = Cookie                    ; section 2.2.1.3
                         / Supported               ; section 2.2.1.7
                         / X-Accept-Authentication ; section 2.2.1.9
                         / X-Proxy-Client-Verb     ; section 2.2.1.11

SEReq-Pragma           = "Pragma: " #SEReq-Pragma-Types CRLF

SEReq-Pragma-Types     = SEReq-Token-OPT

SEReq-Token-OPT        = client-id               ; section 2.2.1.4.5
                         / no-cache              ; section 2.2.1.4.12
                         / proxy-client-agent    ; section 2.2.1.4.21
                         / request-context       ; section 2.2.1.4.23
                         / xClientGuid           ; section 2.2.1.4.33

WMS-SEReq-Body         = remote-event-message  ; section 2.2.5
```

The syntax of the SendEvent response is defined as follows:

```
WMS-SendEvent-Response = Status-Line
                         WMS-SEResp-Headers CRLF

WMS-SEResp-Headers     = *( SEResp-Header-REQ
```

```
                          / SEResp-Header-OPT
                          / SEResp-Pragma
                          / HTTP-Header-Types )

SEResp-Header-REQ       = Cache-Control           ; section 2.2.1.1
                          / Server                ; section 2.2.1.5

SEResp-Header-OPT       = Set-Cookie              ; section 2.2.1.6
                          / Supported             ; section 2.2.1.7

SEResp-Pragma           = "Pragma: " #SEResp-Pragma-Types CRLF

SEResp-Pragma-Types     = SEResp-Token-REQ
                          / SEResp-Token-OPT

SEResp-Token-REQ        = no-cache                ; section 2.2.1.4.12

SEResp-Token-OPT        = client-id               ; section 2.2.1.4.5
                          / timeout               ; section 2.2.1.4.29
```

The following example shows a SendEvent request:

```
POST /welcome.asf HTTP/1.0
Accept: */*
User-Agent: NSPlayer/10.0.0.3802
Host: SampleServer
Pragma: xClientGUID={11223344-1122-1122-1122-AABBCCDDEEFF}
X-Accept-Authentication: Negotiate, MS-NLMP, Digest, Basic
Content-Length: 3273
Content-Type: application/x-wms-sendevent
```

2.2.2.10 Stop Request

The purpose of the Stop request is to request that the server stop streaming content.

The Stop request MUST NOT be sent when the non-pipelined mode of the protocol is used.<50>

The Stop request MUST be sent using the HTTP POST method. It MUST include a Pragma header with a xStopStrm (section 2.2.1.4.38) token. The message body of the POST request MUST be zero length, and the Content-Type header MUST NOT be specified.

The Stop request MUST NOT include any of the following tokens on a Pragma header: log-line (section 2.2.1.4.10), pipeline-request (section 2.2.1.4.16), stream-switch-entry (section 2.2.1.4.27), xKeepAliveInPause (section 2.2.1.4.34).

A server that receives a POST request MUST treat it as a Stop request if a Pragma header includes the xStopStrm token.

If the server is currently transmitting multimedia data to the client, the server MUST transmit a zero-length chunk, as specified in the HTTP 1.1 protocol [RFC2616] section 3.6.1, before responding to the Stop request.

The response to the Stop request MUST contain a zero-length message body.

The Stop request MUST be sent using the HTTP 1.1 protocol, as specified in [RFC2616].

The syntax of the Stop request is defined as follows.

```
WMS-Stop-Request        = WMS-Stop-Req-Line
                          WMS-StopReq-Headers CRLF

WMS-Stop-Req-Line       = "POST" SP Request-URI SP "HTTP/1.1" CRLF

WMS-StopReq-Headers     = *( StopReq-Header-REQ
                          / StopReq-Header-OPT
                          / StopReq-Pragma
                          / HTTP-Header-Types )

StopReq-Header-REQ      = "Content-Length: 0"
                          / User-Agent             ; section 2.2.1.8

StopReq-Header-OPT      = Cookie                   ; section 2.2.1.3
                          / Supported              ; section 2.2.1.7
                          / X-Accept-Authentication; section 2.2.1.9
                          / X-Proxy-Client-Verb    ; section 2.2.1.11

StopReq-Pragma          = "Pragma: " #StopReq-Pragma-Types CRLF

StopReq-Pragma-Types    = StopReq-Token-REQ
                          / StopReq-Token-OPT

StopReq-Token-REQ       = client-id              ; section 2.2.1.4.5
                          / xStopStrm            ; section 2.2.1.4.38

StopReq-Token-OPT       = no-cache               ; section 2.2.1.4.12
                          / proxy-client-agent   ; section 2.2.1.4.21
                          / request-context      ; section 2.2.1.4.23
                          / xClientGuid          ; section 2.2.1.4.33
```

The syntax of the Stop response is defined as follows.

```
WMS-Stop-Response   = Status-Line
                          WMS-StopResp-Headers CRLF

WMS-StopResp-Headers    = *( StopResp-Header-REQ
                          / StopResp-Header-OPT
                          / StopResp-Pragma
                          / HTTP-Header-Types )

StopResp-Header-REQ     = Cache-Control          ; section 2.2.1.1
                          / Server               ; section 2.2.1.5

StopResp-Header-OPT     = Set-Cookie             ; section 2.2.1.6
                          / Supported            ; section 2.2.1.7

StopResp-Pragma         = "Pragma: " #StopResp-Pragma-Types CRLF

StopResp-Pragma-Types   = StopResp-Token-REQ
                          / StopResp-Token-OPT
```

```
StopResp-Token-REQ      = no-cache              ; section 2.2.1.4.12

StopResp-Token-OPT      = client-id             ; section 2.2.1.4.5
                        / timeout               ; section 2.2.1.4.29
```

The following example shows a Stop request:

```
POST /simple.wsx?WMCache=0 HTTP/1.1
Accept: */*
User-Agent: NSPlayer/9.0.0.2833
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: client-id=2754698341
Pragma: xStopStrm=1
Content-Length: 0
```

2.2.3 Packet Types

This section defines the packet types used by the Windows Media HTTP Streaming Protocol. The packets are sent by the server to the client. The packets will appear in the message body of any response sent by the server to an HTTP GET or POST request, as long as it is not a zero-length message body.

When the non-pipelined mode of the protocol is used, the packets will appear in the server's response to the Describe (section 2.2.2.1) and Play (section 2.2.2.6) requests. If the pipelined mode of the protocol is used, the packets will typically also appear in the message body of the Pipeline (section 2.2.2.5), PlayNextEntry (section 2.2.2.7), and SelectStream (section 2.2.2.8) requests.

The remainder of this section is organized as follows: Section 2.2.3.1 defines data structures and field definitions that are common to multiple packet types. Sections 2.2.3.2 through 2.2.3.8 define individual packet types.

2.2.3.1 Common Definitions

All integer fields are transmitted in **little-endian** byte order. If a field is set to an invalid value, clients are free to handle that situation in an implementation-specific manner.<51>

2.2.3.1.1 Framing Header

The Framing Header is used by all packet types. The syntax of the framing header is defined below.

B (1 bit): A 1-bit flag. This flag SHOULD be set to 1 if the next packet will be sent immediately after this packet is sent. In this context, "immediately" means that the server does not intentionally introduce a delay (such as a pacing delay) between the transmission of the two packets. In all other cases, the flag MUST be 0.<52>

Frame (7 bits): A 7-bit field. This field MUST have the value 0x24. (If the **B** and Frame fields are treated as a single byte, the value of this byte will be 0x24 when the B field is 0, and 0xA4 when the B field is 1.)

Value	Meaning
0x24	If treated as a 7-bit field.
0xA4	If the **B** and Frame fields are treated as a single byte.

PacketID (1 byte): A 1-byte field that specifies the type of the packet.

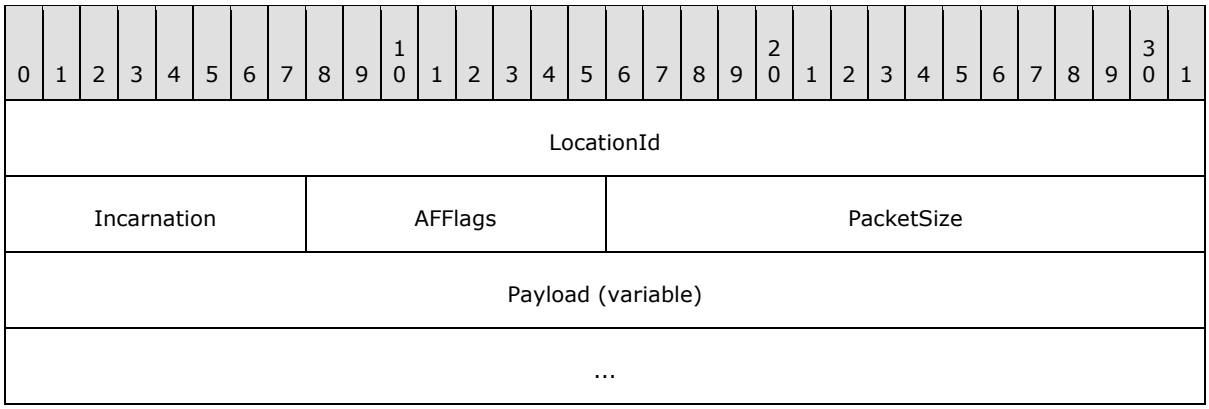
PacketLength (2 bytes): A 16-bit integer field that specifies the size of the MMS data packet (section 2.2.3.1.2), if any, plus the size of the **Reason** field, if any. Unless otherwise specified, the value of this field MUST be equal to the number of bytes in the packet, counted starting from the end of this field. If neither the MMS data packet nor the **Reason** field is present, PacketLength MUST be set to 0.

Reason (4 bytes): An optional 32-bit integer field. Unless specified otherwise, if this field is present, its value MUST be set to an HRESULT code, as specified in [MS-ERREF].

2.2.3.1.2 MMS Data Packet

The MMS data packet structure is a part of the $D (Data) (section 2.2.3.3), $H (Header) (section 2.2.3.5), and $M (Metadata) (section 2.2.3.6) packet types. The header portion of the MMS data packet is sometimes referred to as AF_HEADER_TYPE0.

The maximum size of an MMS data packet is 65,535 bytes. If the object being transferred would cause this size limit to be exceeded, it MUST be transferred as multiple MMS packets. For example, if the size of an ASF header is 100,000 bytes, it MUST be transferred using at least two $H packets. If the first $H packet contains a 65,535-byte MMS packet, the second $H packet would contain a 34,481-byte MMS packet.



LocationId (4 bytes): A 32-bit integer field which specifies the index of the **Payload** field into the complete object that is being transferred. For $M and $H packets, **LocationId** MUST be 0 for the first payload and MUST increment by 1 for each payload. For $D packets, the entire ASF file is considered the object that is being transferred. Thus, for $D packets, this field MUST be set to the payload's ASF packet number. The first ASF packet in an ASF file MUST have **LocationId** 0, the second ASF packet in the file MUST have **LocationId** 1, and so on. Note that since a server may skip ASF packets, the value of the **LocationId** field may not be sequential from one MMS data packet to the next. If the server does not have access to the ASF file (for example, in case of "live" content), the server MUST assume a "virtual" ASF file, incrementing **LocationId** (or decrementing it, when rewinding the content) exactly as if a real ASF file existed.

Incarnation (1 byte): An 8-bit sequence number. The initial value of this field SHOULD be 0. For $M and $H packets, the server MAY increment the **Incarnation** field by 1 each time it responds to a Play (section 2.2.2.6) or PlayNextEntry (section 2.2.2.7) request. If the server chooses to increment the **Incarnation** field, it MUST do it for both $M and $H packets, so that the **Incarnation** field for both packet types stays synchronized.

AFFlags (1 byte): An 8-bit integer field. For $D packets, the field MUST be treated as a sequence number. The initial value MUST be 0 and it MUST increment by 1 for each $D packet that is transmitted. If the MOST recently transmitted $D packet had an **AFFlags** field value of 254, and the server receives a Play (section 2.2.2.6) or PlayNextEntry (section 2.2.2.7) request, then the first $D packet transmitted in the response SHOULD have the **AFFlags** field set to 0. For $M and $H packets, the **AFFlags** field MUST be set to 0x04 if the object being transferred is split into multiple MMS packets and this is the first MMS data packet in the sequence. AFFlags MUST be set to 0x08 if this is the last MMS data packet in the sequence, and AFFlags MUST be set to 0x0C if this is both the first and the last packet in the sequence (that is, the object is fully contained in the **Payload** field of the current MMS data packet).

PacketSize (2 bytes): A 16-bit integer field. This field MUST be set to the total size of the MMS packet, in bytes.

Payload (variable): A variable size block of bytes. The content of this field is different for each packet type. For more information, see all packets from $C (Stream Change Notification) (section 2.2.3.2) through $T (Test Data Notification) (section 2.2.3.8).

2.2.3.2 $C (Stream Change Notification) Packet

The $C (Stream Change Notification) packet is sent by the server when it switches to the next entry in a server-side playlist. The purpose of this packet is to notify the client of that event. The $C packet is defined as a Framing header (section 2.2.3.1.1), with the following additional details:

B (1 bit): As specified in section 2.2.3.1.1.

Frame (7 bits): As specified in section 2.2.3.1.1.

PacketID (1 byte): This field MUST be set to the character "C" (0x43).

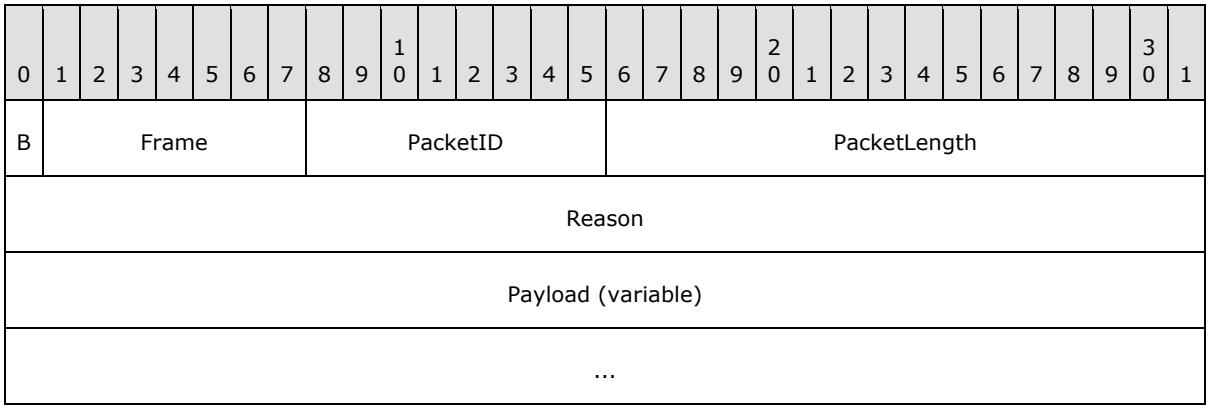
PacketLength (2 bytes): This field MUST be set to either 4 or 8.

Reason (4 bytes): This field MUST be present. If the reason the $C packet is sent is because the server is switching to the previous entry in a server-side playlist, that is, the server is streaming the playlist in reverse, then this field SHOULD be set to 0x400D14BE. Otherwise, it SHOULD be set to 0.

2.2.3.3 $D (Data) Packet

The $D (Data) packet is used by the server to transfer an ASF packet to the client.

The $D packet MUST start with a Framing header (section 2.2.3.1.1), followed by an MMS data packet (section 2.2.3.1.2), with the following additional details:



B (1 bit): As specified in section 2.2.3.1.1.

Frame (7 bits): As specified in section 2.2.3.1.1.

PacketID (1 byte): This field MUST be set to the "D" (0x44) character.

PacketLength (2 bytes): As specified in section 2.2.3.1.1

Reason (4 bytes): This field MUST NOT be present.

Payload (variable): This field MUST contain exactly one complete ASF packet. If the ASF packet contains a **Padding Data** field, as specified in [ASF] section 5.2.4, that field SHOULD be removed before encapsulating the ASF packet in the $D packet, except if the client

identifies itself as "NSServer" with a version earlier than 5.0 on the User-Agent (section 2.2.1.8) header. If the **Padding Data** field is removed, the **Padding Length** field in the ASF payload parsing information section, as specified in [ASF] section 5.2.2, MUST be updated to indicate a non-existent **Padding Data** field.

2.2.3.4 $E (End-of-Stream Notification) Packet

The C is used by the server to specify that the last $D (Data) packet (section 2.2.3.3) for the content has been transmitted. The $E packet also specifies if this was the last content in a server-side playlist or if the client should expect to receive a $C (Stream Change Notification) packet (section 2.2.3.2).

The $E packet is defined as a Framing header (section 2.2.3.1.1), with the following additional details:

B (1 bit): As specified in section 2.2.3.1.1.

Frame (7 bits): As specified in section 2.2.3.1.1.

PacketID (1 byte): This field MUST be set to the "E" (0x45) character.

PacketLength (2 bytes): This field MUST be set to either 4 or 8.

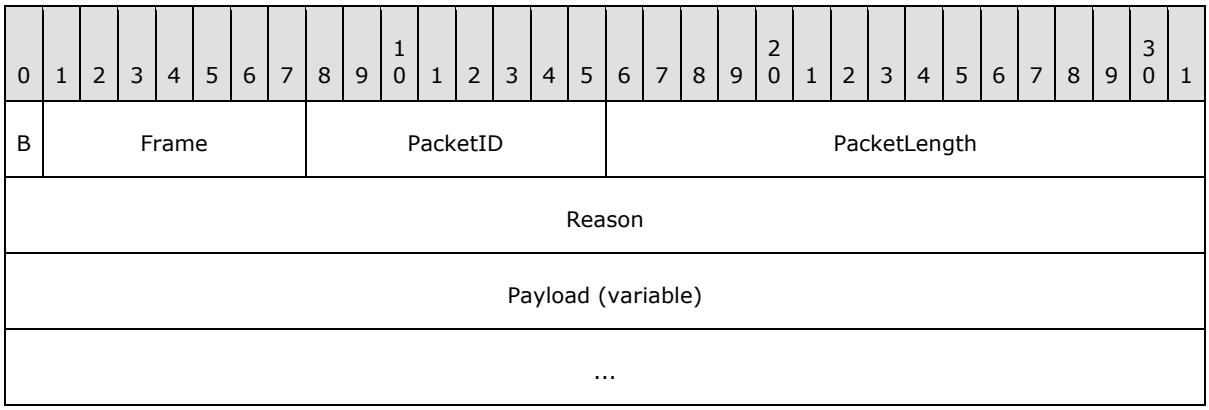
Reason (4 bytes): This field MUST be present. The HRESULT code specifies the error, if any, that caused the server to send the $E packet. HRESULT codes that have special meaning in the context of a $E packet are defined in the table below.

Value	Meaning
S_OK 0x00000000	The server has finished streaming and no more $D packets (section 2.2.3.3) will be transmitted until the next Play request (section 2.2.2.6).
S_FALSE 0x00000001	The server has finished streaming the current playlist entry. Other playlist entries still remain to be streamed. The server will transmit a $C packet when it switches to the next entry.
NS_S_EOS_RECEDING 0x000D2F09	The server was rewinding the content (streaming the content backward) and has reached the beginning of the current playlist entry. No more $D packets will be transmitted until the next Play request.

2.2.3.5 $H (Header) Packet

The $H (Header) packet is used by the server to transfer an ASF header to the client. The packet is sent in the response to Describe (section 2.2.2.1), Play (section 2.2.2.6), and PlayNextEntry requests (section 2.2.2.7), and after a $C packet (section 2.2.3.2).

The $H packet MUST start with a Framing header (section 2.2.3.1.1), followed by an MMS data packet (section 2.2.3.1.2), with the following additional details:



B (1 bit): As specified in section 2.2.3.1.1.

Frame (7 bits): As specified in section 2.2.3.1.1.

PacketID (1 byte): This field MUST be set to the character "H" (0x48).

PacketLength (2 bytes): As specified in section 2.2.3.1.1.

Reason (4 bytes): This field MUST NOT be present.

Payload (variable): This field MUST contain an ASF header. The ASF header consists of the entire ASF Header Object, as specified in [ASF] section 3.1, plus the 50-byte fixed initial portion of the ASF Data Object, as specified in [ASF] section 5.1. If the size of the ASF header would cause the maximum size of the MMS data packet (section 2.2.3.1.2) to be exceeded, the ASF header MUST be broken into multiple smaller pieces, and each piece must be transmitted as a separate $H packet, as specified in section 2.2.3.1.2.

2.2.3.6 $M (Metadata) Packet

The $M (Metadata) packet is used by the server to transfer metadata for the current playlist entry to the client.

The *playlist-gen-id* parameter in the $M packet MUST specify the identifier of the playlist entry that is described by the $M packet. The *playlist-gen-id* parameter is specified in section 2.2.1.4.18.

The value of the *broadcast-id* parameter in the $M packet MUST uniquely identify the source of "live" content within the scope of the current server-side playlist. For example, if the same live content source is used in multiple entries in a single playlist, the value of the *broadcast-id* parameter will be the same in each of those entries. There is no requirement that the values of the *broadcast-id* parameter should be unique across different server-side playlists. If a playlist entry is not using a live content source, the value of the *broadcast-id* parameter MUST be 0.

The *features* parameter in the $M packet MUST specify the properties and capabilities of the playlist entry that is described by the $M packet. The *features* parameter is specified in section 2.2.1.4.8.

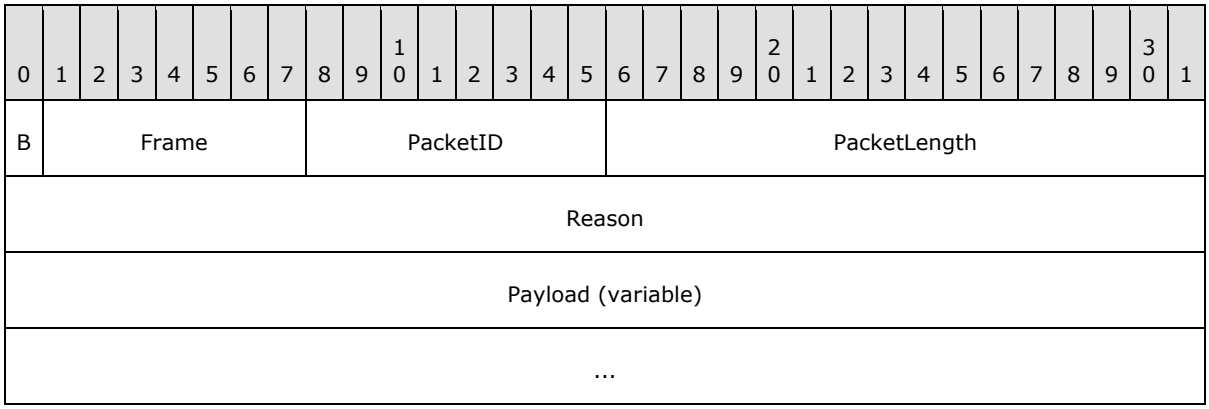
The $M packet SHOULD also contain at least one Content Description List (CDL) (section 2.2.4). A CDL can, for example, specify the author's name, the copyright date, and any extra items that the server specifies. It is structured in name-value pair format, as specified in 2.2.4.

The $M packet MUST NOT be sent to a client with a version number less than 9.0, as specified by the client on the User-Agent (section 2.2.1.8) header.

If the server identifies itself as "Cougar" with a version number greater than or equal to 9.0, as specified by the server on the Server (section 2.2.1.5) header, then it MUST send the $M packet to any clients with version number greater than or equal to 9.0, as specified by the client on the User-Agent header.<53>

If the $M packet is sent by the server, then it MUST be sent immediately before the $H packet, that is, the server MUST NOT transmit packets of any other type between the $M and $H packets.

The $M packet MUST start with a Framing Header (section 2.2.3.1.1), followed by an MMS data packet (section 2.2.3.1.2), with the following additional details:



B (1 bit): As specified in section 2.2.3.1.1.

Frame (7 bits): As specified in section 2.2.3.1.1.

PacketID (1 byte): This field MUST be set to the character "M" (0x4D).

PacketLength (2 bytes): As specified in section 2.2.3.1.1

Reason (4 bytes): This field MUST NOT be present.

Payload (variable): The field MUST adhere to the ABNF syntax for Metadata-Payload, defined below:

```
Metadata-Payload = playlist-gen-id     ; section 2.2.1.4.18
                   "," SP
                   "broadcast-id=" 1*10DIGIT
                   "," SP
                   features                  ; section 2.2.1.4.8
                   %x00
                   *content-description-list ; section 2.2.4
```

The value of the *broadcast-id* parameter MUST be an integer in the range from 0 through 4,294,967,295.

2.2.3.7 $P (Packet-Pair) Packet

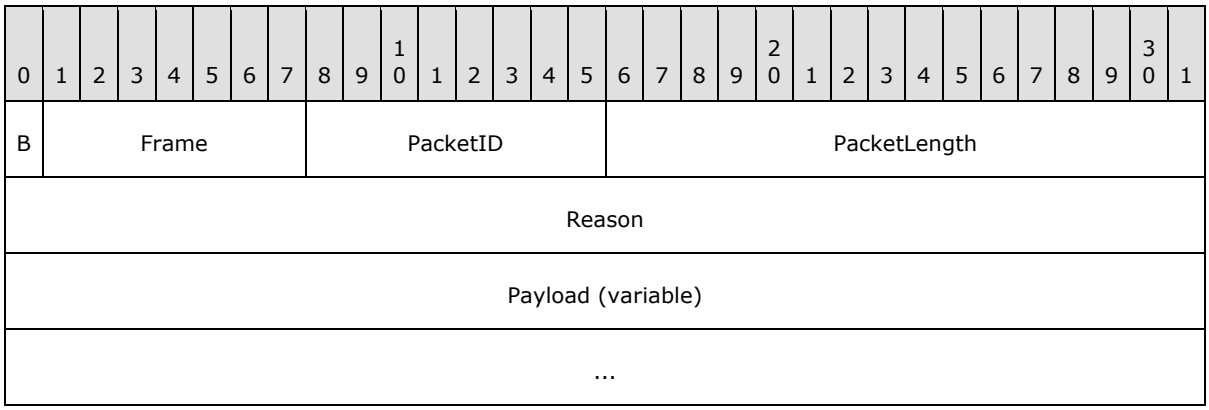
The $P (Packet-Pair) packet SHOULD be sent by the server in the response to a Play request (section 2.2.2.6), if the server specifies the packet-pair-experiment (section 2.2.1.4.14) token on a Pragma (section 2.2.1.4) header, and the value of the token is 1.

If a server sends a $P packet, the first packet in the response to the Play request MUST be a $P packet, and it MUST be followed by two more $P packets with sizes as defined in this section. All three $P packets MUST be sent before any $M or $H packets are sent.

The server SHOULD send the HTTP headers of the response to the Play request and the first $P packet together in a single TCP segment. The second $P packet SHOULD be sent in a TCP segment independently of other $P packets.

A $P packet MUST NOT be sent to a client with version number less than 9.0, as specified by the client on the User-Agent (section 2.2.1.8) header.<54>

A $P packet is defined as a Framing header (section 2.2.3.1.1), with the following additional details:



B (1 bit): As specified in section 2.2.3.1.1.

Frame (7 bits): As specified in section 2.2.3.1.1.

PacketID (1 byte): This field MUST be set to the "P" (0x50) character.

PacketLength (2 bytes): This field MUST be set according to one of the following values:

Value	Meaning
504	This field MUST be set to this value for the first and second $P packets.
1048	This field MUST be set to this value for the third $P packet.

Reason (4 bytes): This field MUST be present and is treated as an unsigned integer. The field MUST be set to the smaller of 504 and the number of bytes in the header portion of the HTTP response, counted from the first byte of the response up to the start of the first $P packet.

Payload (variable): This field MUST consist of random bytes with highly entropic values. For the first $P packet the size of the **Payload** field MUST be 504 bytes minus the value of the **Reason** field. However, if **Reason** is equal to 504, the size of the **Payload** field MUST be 0 bytes. For the second $P packet, the size of the **Payload** field MUST be 504 bytes. For the

third packet, the size of the **Payload** field MUST be 1,048 bytes plus the value of the **Reason** field.

2.2.3.8 $T (Test Data Notification) Packet

The $T (Test Data Notification) packet is sent by the server in the response to the first in a series of two Pipeline requests (section 2.2.2.5).<55>

B (1 bit): As specified in section 2.2.3.1.1.

Frame (7 bits): As specified in section 2.2.3.1.1.

PacketID (1 byte): This field MUST be set to the character "T" (0x54).

PacketLength (2 bytes): This field MUST be set to 0.

Reason (4 bytes): This field MUST NOT be present.

2.2.4 Content Description List Format

Metadata about the content that is sent by the server to the client in the $M (Metadata) (section 2.2.3.6) packet is formatted by using a syntax called the Content Description List (CDL). Any number of information items can be stored in the CDL such as copyright information, playlist information, duration of the content, size of the content, or content description.

A CDL consists of at least one Content Description. Each Content Description specifies metadata in a particular language. The Content Description consists of a series of name-value pairs.

Although it is not a general requirement of Content Descriptions, when a Content Description is used in a CDL, the first name-value pair of each Content Description MUST have the name "language". The value MUST be either a language tag, as specified in [RFC2616] section 3.10, or a zero-length string. If the length of the language tag is nonzero, it identifies the language of the metadata in the Content Description. Otherwise, the language of the Content Description is unspecified.

The remaining name-value pairs in the Content Description can have arbitrary names. There can be any number of name-value pairs, and each name and value can be of an arbitrary length.

The following table defines the individual fields used to represent a single name-value pair in the Content Description.

Term	Description
<name length>	The length, in bytes, of the name specified in the name-value pair.
<name>	The name specified for the name-value pair, expressed using UTF-8 characters, as specified in [RFC3629].

Term	Description
<value type>	Numeric value indicating the value's type, as specified in 2.2.4.1.
<value length>	Length in bytes of the value.
<value>	A string representation of the actual data of the name-value pair, using UTF-8 characters.

The format for a name-value pair is as follows:

```
<name length>,<name>,<value type>,<value length>,<value>
```

An example of a name-value pair is:

```
9,copyright,31,30,)) 1999 Microsoft Corporation
```

This example represents a 9-character name-value pair (copyright) which is of type 31 (a wide character string).

The format is more formally defined using ABNF as follows:

```
cd-length                    = 1*10DIGIT
cd-string                    =  *VCHAR

cd-namevalue                 = cd-length "," cd-string ","
                                 1*3DIGIT ","
                                 cd-length "," cd-string

content-description          = cd-namevalue
                                 *( "," cd-namevalue ) CRLF

language-content-description = "7,language,31," cd-string ","
                                 language-tag   ; [RFC2616] section 3.10
                                 *( "," cd-namevalue ) CRLF

content-description-list     = 1*( language-content-description )
```

2.2.4.1 CDL Value Types

The following table shows supported values for the **<value type>** field in the Content Description, using data types as specified in [MS-DCOM] section 7.1.

Any type not listed in the table MUST NOT be used by servers. If a client encounters a CDL value that uses a value type not listed in the table below, the CDL value MUST be ignored.

Data Type	<value type>
VT_BOOL	11

Data Type	<value type>
VT_BSTR	8
VT_CY	6
VT_ERROR	10
VT_I1	16
VT_I2	2
VT_I4	3
VT_INT	22
VT_LPWSTR	31
VT_UI1	17
VT_UI2	18
VT_UI4	19
VT_UINT	23

2.2.5 Remote Event Format

The Remote Event format is used in the message body of the SendEvent request (section 2.2.2.9).

The syntax of the Remote Event format is defined by using ABNF as follows.

```
EncodingFormat   = 1
EncodingVersion      = 1
EventType    = ( 28 / 29 / 30 )
EventReason   = 1*10DIGIT

remote-event-message = EncodingFormat CRLF
     EncodingVersion ","
     EventType ","
     EventReason CRLF
     [ 2(content-description) ]  ; section 2.2.4
```

The value of the *EventType* parameter MUST be set to one of the following three values: 28 to specify the remote-open event, 29 to specify the remote-close event, or 30 to specify the remote-log event.

The value of the *EventReason* parameter MUST be an integer in the range from 0 through 4,294,967,295. It MUST be treated as an HRESULT code, as specified in [MS-ERREF].

The two content-description parameters MUST be present if the value *EventType* parameter is 30 (that is, for remote-log events) and MUST NOT be present otherwise.

The first content-description MUST contain at least the following fields:

<name>	<value-type tag>	<value>
context-type	VT_I4	2
WMS_USER_AGENT	VT_LPWSTR	A string adhering to the syntax for client-token, or similar, as specified in User-Agent (section 2.2.1.8) header.
WMS_USER_GUID	VT_LPWSTR	A string adhering to the syntax for GUID value, as specified in xClientGUID (section 2.2.1.4.33) token.

The second content-description MUST contain at least the following fields:

<name>	<value-type tag>	<value>
context-type	VT_I4	5
@WMS_COMMAND_CONTEXT_URL	VT_LPWSTR	A string adhering to the syntax for httpurl, as specified in [RFC1738].
@.REMOTE_LOG	VT_LPWSTR	A "rendering" log in XML format, as specified in [MS-WMLOG]

For more information about the syntax for content-description, see section 2.2.4.

3 Protocol Details

3.1 Client Details

Unless specified otherwise, the protocol reports the occurrence of an error to the higher layer, stops all timers, and stops processing further messages. Possible errors include the following: failure to connect to the server, the connection to the server is unexpectedly closed, the response to a request indicates an error, the **Reason** field in the $E or $C packets indicates an error, or an unexpected packet (such as receiving a $P packet when waiting for a $H packet) is received.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Client-ID: The value of this variable is an identifier assigned by the server for the current streaming "session". Because the client-ID value is provided by the server, the initial value is undefined.

Client-Token: This variable stores the token used by the client on the User-Agent header. Three values are possible: "NSPlayer", "NSServer", and "WMCacheProxy".

Keepalive-timeout: This variable stores the frequency at which the client will send KeepAlive requests (section 2.2.2.3). The default value is 60 seconds.

Mode: This variable specifies if the protocol is in pipelined mode or non-pipelined mode.

Pipeline-Test-Allowed: A flag that, if set to 1, indicates that the server is willing to let the client test if the pipelined mode of the protocol can be used.

Playlist-gen-id: The value of this variable is an identifier assigned by the server to identify the current playlist entry. The default value is 0.

Server-features: This variable stores the capabilities that the server specified on the most recently-received Supported header.

Server-Version: This variable stores the server product name (the value of server-token) and major and minor version number that the server specified on the most recently received Server header.

State: This variable stores the client's state. Possible values are INIT, STREAMING, and IDLE.

Note The above conceptual data can be implemented by using a variety of techniques. This protocol does not prescribe or advocate any specific implementation technique.

3.1.2 Timers

Keepalive: This timer is used for sending KeepAlive requests (section 2.2.2.3). The time-out period is controlled by the Keepalive-timeout variable specified in section 3.1.1. The minimum allowed value for the time-out period is 10 seconds.

3.1.3 Initialization

Initialization of the protocol occurs as the result of a higher layer asking for information about multimedia content located on a server. That event is described in section 3.1.4.2.

The variables defined by the abstract data model MUST initially assume their default values. Variables that do not have a default defined MUST be initialized as follows:

- The State variable MUST be set to INIT.
- The Client-Token variable MUST be set to "WMCacheProxy" if the client is acting as a caching proxy server. If the client is a non-caching server acting on behalf of one or more clients, then the Client-Token variable MUST be set to "NSServer". Otherwise, Client-Token MUST be set to "NSPlayer".
- The Mode variable MUST be set to non-pipelined mode.
- The Server-features variable MUST be set to specify support for com.microsoft.wm.predstrm (section 2.2.1.7.2), com.microsoft.wm.srvppair (section 2.2.1.7.3), and for com.microsoft.wm.sswitch (section 2.2.1.7.4).

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Request to Retrieve Caching Information

This event can occur when the application is a caching proxy server. The event can occur if the higher layer wants to check if content is available, and if it can be cached, but does not necessarily want to stream it.

The higher layer MUST provide the URL that will be used in the request.

If this will be the first request that is sent by the client, the client MUST perform the initialization of the protocol as specified in section 3.1.3.

The client MUST then establish a TCP connection to the server by using the IP address and port number obtained by parsing the URL.

If the value of the Client-Token variable in the abstract data model is not "WMCacheProxy", then the client MUST send a Describe request (section 2.2.2.1).

Otherwise, the client MUST send a GetContentInfo request to the server.

The GetContentInfo request must adhere to the syntax specified in section 2.2.2.2.

In addition, the common processing steps specified in section 3.1.5.1 MUST be followed when sending the GetContentInfo request.

After having sent the request, the client MUST wait for the response to be received as specified in section 3.1.5.3.

3.1.4.2 Request to Retrieve Content Information

This event causes the client to send a Describe request (section 2.2.2.1) to the server. The most common scenarios in which an application would ask the client for information about multimedia content are the following:

1. A media player application that intends to play multimedia content, which will be streamed from a server. The media player knows the URL to the content and it might already know at what time position and at what rate it intends to play the content. However, before it can start playing the content, it needs to retrieve information about what audio and video streams are included in the content, what decoders will be needed to decompress the content, and other information.

2. A cache that already has a copy of the content but must retrieve information about the content from the server to determine if the cached copy is still fresh.

3. A server or intermediate device, such as a non-caching proxy, which is asking for information on behalf of another client.

The higher layer MUST provide the URL that will be specified in all requests sent by the client.

If this will be the first request that is sent by the client, the client MUST perform the initialization of the protocol as specified in section 3.1.3.

The client MUST then establish a TCP connection to the server by using the IP address and port number obtained by parsing the URL. Next, the client MUST send the Describe request to the server, as specified in the next section, 3.1.4.2.1.

3.1.4.2.1 Sending the Describe Request

The Describe request must adhere to the syntax specified in section 2.2.2.1.

In addition, the common processing steps specified in section 3.1.5.1 MUST be followed when sending the Describe request.

If the client implementation supports the pipelined mode of the protocol, the Describe request SHOULD include a pipeline-experiment (section 2.2.1.4.15) token on the Pragma header, with the value set to 1, as specified in 2.2.1.4.14.

If the client implementation would like to use $P (Packet-Pair) packets to measure the bottleneck bandwidth between the client and the server, and if the value of the Server-Version variable is not uninitialized and is less than 5.0, and if the value of the Server-features variable indicates that the com.microsoft.wm.srvppair feature is supported, then the client SHOULD specify the packet-pair-experiment (section 2.2.1.4.14) token on the Pragma header with the value set to 1. Otherwise, the packet-pair-experiment token SHOULD NOT be specified. For example, a client that intends to select all of the streams in the ASF file, or that does not support the stream-switch-entry token, is not likely to have any use for $P packets and thus does not need to specify the packet-pair-experiment token, as specified in 2.2.1.4.14.

After having sent the request, the client MUST wait for the response to be received, as specified in section 3.1.5.4.

3.1.4.3 Request to Start Streaming Content

This higher-layered triggered event can occur when the client is not currently streaming from the server. The event causes the client to send a Play request (section 2.2.2.6) to the server. The most common scenarios in which an application would ask the client to request the server to start streaming content are the following:

1. A media player application that has examined the ASF header that was received from the server (as specified in 3.1.5.7) and determined that it is capable to decompress and play the multimedia content.

2. A cache that has determined that the currently-cached copy of the content, if any, is either stale or incomplete.

3. A server or intermediate device, such as a non-caching proxy, which is asking for content to be streamed on behalf of another client.

If the TCP connection that is used for sending the most recent Describe (section 2.2.2.1), Play (section 2.2.2.6), or PlayNextEntry (section 2.2.2.7) request is still open, the connection MUST either be used for sending the next request or it MUST be closed at this time. If the TCP connection is closed, the client MUST then establish a TCP connection to the server by using the IP address and port number obtained by parsing the URL.

If the value of the Pipeline-Test-Allowed variable is 1, and the Mode variable specifies that the protocol is currently in non-pipelined mode, then the client SHOULD send two Pipeline requests (section 2.2.2.5). The Pipeline requests must adhere to the syntax defined in section 2.2.2.5. After having sent the two requests, the client MUST wait for the response to the first Pipeline request to be received, as specified in section 3.1.5.8.

Next, the client MUST send the Play request to the server, as specified in the next section 3.1.4.3.1.

3.1.4.3.1 Sending a Play Request

If the TCP connection that is used for sending the most recent Describe, Play, or PlayNextEntry request is still open, the connection MUST either be used for sending the Play request or it MUST be closed at this time. If the TCP connection is closed, the client MUST then establish a TCP connection to the server by using the IP address and port number obtained by parsing the URL.

The Play request must adhere to the syntax specified in section 2.2.2.6.

In addition, the common processing steps specified in section 3.1.5.1 MUST be followed when sending the Play request.

Because the ASF header will typically specify multiple streams, the higher layer MUST select exactly which of the streams should be streamed from the server. If a subset of the streams listed in the ASF header are selected to be streamed, or if the value of the Client-Token variable is not "NSServer", or if the client's version number is greater than or equal to 9.0, then the client MUST specify the complete selection state of all streams by including a stream-switch-entry (section 2.2.1.4.27) token on a Pragma header in the Play request. In this case, the stream-switch-entry token MUST specify all available streams, and each stream MUST either be marked as on by specifying the thinning level parameter as 0, or marked as off by specifying the destination stream parameter as "ffff", depending on the selection made by the higher layer.

If the higher layer specifies that the server to skip to the next or previous playlist entry, and the Server-Version variable is greater than, or equal to, 5.0, then the client SHOULD send this information to the server using the pl-offset (section 2.2.1.4.20) token on the Pragma header.

The higher layer MUST also provide either the time position, or the ASF packet number, from which the server should be asked to start streaming. If a time position is provided, the client MUST send this information using the stream-time (section 2.2.1.4.28) token on the Pragma header. If an ASF packet number is provided, the client MUST send this information using either the stream-offset (section 2.2.1.4.25) token or the packet-num (section 2.2.1.4.13) token on the Pragma header.

The higher layer MUST specify the playlist entry ID for which the above mentioned time position or ASF packet number applies to. Usually this will be the same playlist entry indicated by the client's Playlist-gen-id variable. But it can happen that the client has recently received a $M packet and updated its Playlist-gen-id variable, and that the higher layer has not yet processed the fact that the

playlist entry has changed. The client MUST compare the value of its Playlist-gen-id variable against the playlist entry ID specified by the higher layer. If the values do not match, and the higher layer's playlist entry ID is nonzero, then the client MUST specify a playlist-seek-id (section 2.2.1.4.19) token on the Pragma header, and use the higher layer's playlist entry ID as the value for that token.

The higher layer MUST specify at what rate the multimedia content should be played back. For example, if the higher layer wants to play the multimedia content in reverse, it MUST specify this, and the rate of playback. The client MUST send this information by using the rate (section 2.2.1.4.22) token on the Pragma header.

The higher layer SHOULD specify an amount of data that should be streamed faster than real time and the bit rate at which the server should stream this data. The higher layer SHOULD also specify the bit rate that can be used for streaming between the server and the client. If the value of the Server-Version variable is greater than, or equal to, 5.0, then the client SHOULD send this information to the server using the AccelBW (section 2.2.1.4.1), AccelDuration (section 2.2.1.4.2), and LinkBW (section 2.2.1.4.9) tokens on the Pragma header.

If the value of the Client-Token variable is "NSServer" or "WMCacheProxy", then the higher layer SHOULD specify an alternate amount of data that should be streamed faster than real time and an alternate bit rate at which the server should stream this data. These alternate amounts are for the client's own behalf, as opposed to the AccelDuration and AccelBW values which are on the behalf of another (external) client. If the value of the Server-Version variable is greater than, or equal to, 5.0, then the client SHOULD send this information to the server by using the BurstDuration (section 2.2.1.4.4) and BurstBW (section 2.2.1.4.3) tokens on the Pragma header.

If the client supports the X-StartupProfile (section 2.2.1.12) header, and the value of the Server-features variable indicates that the server supports the com.microsoft.wm.startupprofile feature, then the client SHOULD specify the com.microsoft.wm.startupprofile token on the Supported header when it sends a Play request. A client that does not support the X-StartupProfile header MUST NOT include this token in the Supported header.

The higher layer SHOULD specify that the entire content should be streamed faster than real time at some transmission rate chosen by the higher layer. If the Server-Features variable indicates that the server supports the com.microsoft.wm.fastcache feature (2.2.1.7.1) and the Server-Version variable is greater than, or equal to, 5.0, then the client SHOULD send this information to the server by using the Speed (section 2.2.1.4.24) token on the Pragma header. The Speed token MUST NOT be included in the request unless the server has explicitly specified that it supports the com.microsoft.wm.fastcache feature.

If the value of the Mode variable specifies that the pipelined mode of the protocol is used, then the Play request MUST be sent using the HTTP 1.1 protocol, and the client MUST specify the version11-enabled (section 2.2.1.4.30) token on the Pragma header, and the value of the token MUST be 1.

After having sent the request, the client MUST wait for the response to be received, as specified in section 3.1.5.10.

3.1.4.4 Request to Change the Currently Selected Streams

This event occurs when the higher layer wants to change the streams that are currently being streamed. For example, the higher layer may have decided to switch from an English language audio stream to a Spanish language one, or it may have decided to switch to a stream with higher quality video.

If the pipelined mode of the protocol is used and the TCP connection used for the most recent Play or PlayNextEntry request is open, then the client MUST send a SelectStream request (section 2.2.2.8) on this connection.

Otherwise, the client MUST establish a new TCP connection to the server using the IP address and port number obtained by parsing the URL used for the Describe request. A SelectStream request MUST be sent on this new connection, as specified in SelectStream Request (section 2.2.2.8).

The client SHOULD NOT send a SelectStream request to the server unless the value of the Server-features variable in the abstract data model indicates that the server supports the com.microsoft.wm.sswitch feature (2.2.1.7.4).

The stream-switch-entry (section 2.2.1.4.27) token on the Pragma header of the SelectStream request MUST specify the streams that are affected by the change that was requested by the higher layer. Streams that are unaffected SHOULD NOT be specified on the stream-switch-entry token.

If the value of the Mode variable specifies that the pipelined mode of the protocol is used, then the SelectStream request MUST be sent using the HTTP 1.1 protocol, and the client MUST specify the version11-enabled (section 2.2.1.4.30) token on the Pragma header, and the value of the token MUST be 1.

After having sent the request, the client MUST be prepared to receive the response, as specified in section 3.1.5.16. If the value of the State variable is STREAMING, the client MUST also be prepared to receive any packet that would normally have been received, such as a $D packet or a $E packet.

3.1.4.5 Selection of Streams to Play from the New Playlist

This event occurs after the higher layer has received the ASF header for a new playlist entry and the higher layer is ready to start processing the ASF packets for the new playlist entry. This event is the higher layer's opportunity to select the streams to receive from the new playlist entry. Because the bit rate needed to stream each playlist entry depends on how the content was encoded and which streams are selected, the higher layer may also specify new values for the parameters that control how much faster than real time (if at all) the content is streamed.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, or if the TCP connection used for the most recent Play (section 2.2.2.6) or PlayNextEntry (section 2.2.2.7) request is no longer open, then the client MUST establish a new TCP connection to the server by using the IP address and port number obtained by parsing the URL used for the Describe request (section 2.2.2.1).

If the value of the Mode variable specifies that the pipelined mode of the protocol is used, the client MUST send a PlayNextEntry request (section 2.2.2.7). The request MUST be sent on either the same TCP connection that was used for the most recent Play or PlayNextEntry request, or if that connection is no longer open, on the TCP connection that was newly established according to the rules above.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, the client MUST send a SelectStream request (section 2.2.2.8). The request MUST be sent on the TCP connection that was newly established according to the rules above.

If the client that wants the server to use Predictive Stream Selection for the current playlist entry, then it MUST send the Supported header with the com.microsoft.wm.predstrm token (as specified in section 2.2.1.7.2) in the Play, SelectStream, or PlayNextEntry request, as appropriate. A client that does not want the server to use Predictive Stream Selection for the current playlist entry MUST send a Supported header without the com.microsoft.wm.predstrm token in the Play, SelectStream, or PlayNextEntry request.

If the client is sending a PlayNextEntry request, it SHOULD contain a stream-switch-entry (section 2.2.1.4.27) token on a Pragma header (the token is already mandatory in the case of a SelectStream request).

If the stream-switch-entry token is included on a Pragma header, then the token MUST specify the complete selection state of all streams. In other words, the stream-switch-entry token MUST specify all available streams, and each stream MUST either be marked as on by specifying the thinning level parameter as 0 or marked as off by specifying the destination stream parameter as "ffff", depending on the selection made by the higher layer.

The higher layer SHOULD specify an amount of data that should be streamed faster than real time and the bit rate at which the server should stream this data. The higher layer SHOULD also specify the bit rate that can be used for streaming between the server and the client. If the value of the Server-Version variable is greater than, or equal to, 5.0, then the client SHOULD send this information to the server by using the AccelBW (section 2.2.1.4.1), AccelDuration (section 2.2.1.4.2), and LinkBW (section 2.2.1.4.9) tokens on the Pragma header.

If the value of the Client-Token variable is "NSServer" or "WMCacheProxy", then the higher layer SHOULD specify an alternate amount of data that should be streamed faster than real time and an alternate bit rate at which the server should stream this data. These alternate amounts are for the client's own use, as opposed to the AccelDuration and AccelBW values which are for the use of another (external) client. If the value of the Server-Version variable is greater than, or equal to, 5.0, then the client SHOULD send this information to the server using the BurstDuration (section 2.2.1.4.4) and BurstBW (section 2.2.1.4.3) tokens on the Pragma header.

The higher layer SHOULD specify that the entire content should be streamed faster than real time at some transmission rate chosen by the higher layer. If the Server-features variable indicates that the server supports the com.microsoft.wm.fastcache feature (as specified in section 2.2.1.7.1) and the Server-Version variable is greater than, or equal to, 5.0, then the client SHOULD send this information to the server by using the Speed (section 2.2.1.4.24) token on the Pragma header.

After having sent the request, the client MUST wait for the response to be received. If the request is SelectStream, section 3.1.5.16 specifies how to process the response. If the request is PlayNextEntry, section 3.1.5.17 specifies how to process the response.

3.1.4.6 Request to Stop Streaming

This event occurs if the higher layer wants to stop streaming. The end user may have requested that streaming should stop. Or the end user may have requested to seek to some position in the content while the client is currently streaming multimedia content from a different position.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used and the TCP connection used for the most recent Play request is open, then the client MUST close that TCP connection.

If the value of the Mode variable specifies that the pipelined mode of the protocol is used and the TCP connection used for the most recent Play or PlayNextEntry request is open, then the client MUST send a Stop request (section 2.2.2.10) on that connection.

If the value of the State variable is STREAMING, and the value of the Mode variable specifies that the pipelined mode of the protocol is used, then the client MUST send a Log request (section 2.2.2.4) to the server on the same TCP connection that the Stop request was sent on.

If the value of the State variable is STREAMING, and the non-pipelined mode of the protocol is used, then the client MUST establish a new TCP connection to the server using the IP address and port number obtained by parsing the URL used for the Describe request. The client MUST then send a Log request to the server on the new TCP connection.

If the server's product name is "Cougar", and its version number is greater than or equal to 9.0, as determined by the Server-Version variable, then the above mentioned Log request SHOULD contain

a logging message in XML format as defined in [MS-WMLOG]. If the client will submit remote-log remote events using the SendEvent request (section 2.2.2.9), then the logging information included in the Log request MUST be a "streaming" log, as defined in [MS-WMLOG]. Otherwise, the logging information MUST be a "legacy" style log, as defined in [MS-WMLOG].

Next, the value of the State variable MUST be set to IDLE.

If the Keepalive timer is running, it MUST be restarted. If it is not running, it MUST be started.

If a Stop request was sent, the client MUST wait for the response to be received, as specified in section 3.1.5.18. After having received the Stop response, the client MUST wait for the Log response to be received.

If only a Log request was sent, then the client MUST wait for the response to be received, as specified in section 3.1.5.14.

3.1.4.7 Request to Change the Playback Position

This event occurs when the higher layer wants to start streaming from some specific position in the content.

If the value of the State variable is INIT or IDLE, this event MUST be treated the same way as a request to start streaming, as specified in section 3.1.4.3.

If the value of the State variable is STREAMING, the client MUST first request the server to stop streaming, as specified in section 3.1.4.6. After streaming has completed, the client MUST request to start streaming at the new playback position, as specified in section 3.1.4.3.

3.1.4.8 Playback of Content Has Finished

This event occurs when the application software is a media player and it has finished rendering (that is, playing back) the content in the current playlist entry.

If the client specified the Speed (section 2.2.1.4.24) token on the Pragma header in the Play (section 2.2.2.6) or PlayNextEntry (section 2.2.2.7) request for the current playlist entry, and the server specified the remote-log remote event in the x-wms-event-subscription directive (as specified in section 2.2.1.1.10) on the Cache-Control header, then the client MUST establish a new TCP connection to the server using the IP address and port number obtained by parsing the URL used for the Describe request. The client MUST then send a SendEvent request (section 2.2.2.9) to the server on the new TCP connection.

After having sent the request, the client MUST wait for the response to be received, as specified in section 3.1.5.19.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Sending a Request (All Request Types)

This section specifies common steps that MUST be performed whenever the client sends a request of any of types specified in section 2.2.2 to the server.

If the Keepalive timer is running, it MUST be stopped.

The request sent by the client MUST NOT specify any of the headers and tokens defined in section 2.2.2 that are defined only for use in responses.

If the Server-Version variable is uninitialized, then the request sent by the client MUST NOT specify any of the headers and tokens defined in section 2.2.2.2 that are defined only for use in requests sent to servers with a version number higher than the version number indicated by the Server-Version variable.

The client MUST specify the User-Agent (section 2.2.1.8) header in the request. The client-token parameter on the User-Agent header MUST be set to the value of the Client-Token variable in the abstract data model. If the value of Client-Token is "NSServer" and the client is acting as a proxy server, then the User-Agent header MUST include the "; via WMCacheProxy" syntax element. Otherwise, that syntax element MUST NOT be included.

The client SHOULD specify the Cache-Control (section 2.2.1.1) header in the request.

The client MUST specify the Content-Type (section 2.2.1.2) header if it is sending a POST request.

The client SHOULD specify the Supported (section 2.2.1.7) header in the request if the version number specified by the Server-Version variable is undefined or is greater than or equal to 9.0. If the Supported header is specified, the header MUST correctly reflect the features that are supported by the client.

If the Supported header is specified in the request, it SHOULD list support for the following features: com.microsoft.wm.predstrm (section 2.2.1.7.2), and com.microsoft.wm.startupprofile (section 2.2.1.7.5). <56>

For example, if the client sends a SelectStream request (section 2.2.2.8) in the future, the Supported header MUST specify the com.microsoft.wm.sswitch (section 2.2.1.7.4) token.

The client SHOULD specify the xClientGUID (section 2.2.1.4.33) token on a Pragma header in the request and the token MUST specify the same GUID for all requests belonging to the same streaming session. The client MAY specify a different GUID on the xClientGUID token for different streaming sessions.<57>

If the Client-ID variable in the abstract data model has been assigned a value, the client MUST specify a client-id token on a Pragma header in the request. The value of the client-id token MUST be equal to the value of the Client-ID variable.

If the Playlist-gen-id variable in the abstract data model has a nonzero value, the client MUST specify a playlist-gen-id token on a Pragma header in the request. The value of the playlist-gen-id token MUST be equal to the value of the Playlist-gen-id variable.

The client SHOULD specify the X-Accept-Authentication (section 2.2.1.9) and X-Accept-Proxy-Authentication (section 2.2.1.10) headers in the request.

If the client is acting as a proxy server and relaying a request from another client, then the request MUST include the Via header (section 14.45 of [RFC2616]) in the request.

If the client is acting as a proxy server and relaying a request from another client, then the request SHOULD include the X-Proxy-Client-Verb (section 2.2.1.11) header in the request.

If the client is acting as a proxy server and relaying a request from another client, and if that request contained either a User-Agent (section 2.2.1.8) header, or a Pragma (section 2.2.1.4) header with the proxy-client-agent (section 2.2.1.4.21) token, then the client MUST include a Pragma header with the proxy-client-agent token in the request.

When the client is sending the Pragma header with the proxy-client-agent token in a request, the value specified by that token MUST be identical to the value of the proxy-client-agent token that was received in the original request (that is, the request that the client is relaying). If the original

request does not have a Pragma header with the proxy-client-agent token, then when the client sends the Pragma header with the proxy-client-agent token, the "user-agent-data" syntax element on that header MUST be identical to the "user-agent-data" syntax element of the User-Agent header in the original request. <58>

If there are any cookies to send for the URL specified on the HTTP request line, the Cookie (section 2.2.1.3) header MUST be included in the request.<59>

3.1.5.2 Receiving a Response (All Request Types)

This section specifies common steps that MUST be performed whenever the client receives the response to a request that it has sent. These steps MUST be performed prior to any processing that is specific to a particular request type.

Because requests and their responses do not have any kind of sequence number information, the client MUST assume that the responses are received in exactly the same order in which the requests were sent.

The client MUST inspect the Server (section 2.2.1.5) header in the response. If the Server header is missing, or if the server-token parameter on the Server header does not match any of the values used by this protocol, as specified in section 2.2.1.5, then the server is not compatible with the Windows Media HTTP Streaming Protocol and is probably a regular HTTP Web server. Because this protocol does not interoperate with incompatible servers, this MUST be treated as an error and reported as such to the higher layer.

Otherwise, the server version information (server-token and major and minor version number) on the Server header MUST be saved in the Server-Version variable in the abstract data model.

The client MUST process the Supported header, if present. Each feature token on the header MUST be added to the Server-features variable in the abstract data model. If the header is present, any feature token not listed on the header MUST be removed from the Server-features variable.

If the HTTP status code in the response is 401, this means that the server requires access authentication. Status code 407 means that a proxy server requires access authentication. The rules for access authentication specified in section 11 of [RFC2616] MUST be followed. The client SHOULD keep the TCP connection open, as it will be needed to resubmit the request. When the client is ready to resubmit the HTTP request with the authentication credentials that the server requested, the HTTP request MUST be sent on the same TCP connection on which the 401 or 407 response was received on, except if that TCP connection has been closed. If the TCP connection has been closed, the client MUST establish a new TCP connection to the server and send the HTTP request on that connection.<60>

If the HTTP status code in the response is not 401 or 407, then the client MUST process the client-id token on the Pragma header, if present. If it is present, the Client-ID variable in the abstract data model MUST be set to the value of the token.

The client MUST process the playlist-gen-id token on the Pragma header, if present. If it is present, the Playlist-gen-id variable in the abstract data model MUST be set to the value of the token.

The client MUST process the timeout token on the Pragma header, if present. If it is present, the Keepalive-timeout variable in the abstract data model MUST be set to the value of the token.

The client SHOULD adhere to the directives specified by the Cache-Control header and MUST NOT cache the content unless explicitly allowed by the appropriate directive (that is, public, proxy-public, or user-public).<61>

The client SHOULD ignore the version-info (section 2.2.1.4.31) and version-url (section 2.2.1.4.32) tokens.<62>

If the Set-Cookie header is present in the response, the cookies on that header MUST be processed as specified in 2.2.1.6.<63>

3.1.5.3 Receiving a GetContentInfo Response

The client MUST first follow the steps specified in section 3.1.5.2.

If the value of the Client-Token variable is "WMCacheProxy", and the HTTP status code in the response is equal to 400, and the server version is less than 9.0 according to the Server-Version variable, then the client MUST set the value of the Client-Token variable to "NSServer" and use a Describe request instead of a GetContentInfo request to get the server's Cache-Control header. To resubmit the request as a Describe request,it will be necessary to establish a new TCP connection to the server. The client MUST then continue by following the steps specified in section 3.1.4.2.1.

Otherwise, if the HTTP status code indicates that the request succeeded, the server SHOULD report the information in the Cache-Control header to the higher layer.

3.1.5.4 Receiving a Describe Response

The client MUST first follow the steps specified in section 3.1.5.2.

If the value of the Client-Token variable is "WMCacheProxy", and the HTTP status code in the response is equal to 400, and the server version is less than 9.0 according to the Server-Version variable, then the client MUST set the value of the Client-Token variable to "NSServer" and resubmit the Describe request. To resubmit the Describe request, it will be necessary to establish a new TCP connection to the server. The client MUST then continue by following the steps specified in section 3.1.4.2.1.

If the HTTP status code in the response is in the range 300-305, this means that the server is asking the client to connect to another server. The client MUST connect to the server specified in the response, by following the rules in section 10.3 of [RFC2616]. This is a brief summary of those rules: If the status code is 305, the URL on the Location header ([RFC2616], section 14.30) is for a proxy, and the URL used in the Describe request MUST remain unchanged. For status codes 300-304, the URL on the Location header MUST replace the URL used in the Describe request. The server MUST close the current TCP connection and establish a new TCP connection to the server, or proxy server, as appropriate depending on the status code. The client MUST then continue by following the steps defined in Sending the Describe Request (section 3.1.4.2.1)

Otherwise, if the HTTP status code indicates that the request succeeded, the client MUST perform the following steps:

- The variable Pipeline-Test-Allowed MUST be set to the value of the pipeline-experiment (section 2.2.1.4.15) token on the Pragma header. If that token is missing, the variable MUST be set to 0.
- If the Via header is present in the response, and it indicates that the response was processed by a HTTP 1.0 proxy server, then the client SHOULD set the value of the Pipeline-Test-Allowed variable to 0 as specified in [RFC2616] section 14.45.
- If the token packet-pair-experiment (section 2.2.1.4.14) is present on the Pragma header, and its value is 1, then the client MUST wait for the first $P (Packet-Pair) packet to be received, as specified in section 3.1.5.5.

Otherwise, if the server's product name is "Cougar" and the version number is greater than or equal to 9.0, according to the Server-Version variable, then the client MUST wait for the $M (Metadata) packet to be received, as specified in section 3.1.5.6.

Otherwise, if the server's version number is greater than or equal to 9.0, then the client MUST wait for either a $M (Metadata) packet or a $H (Header) packet to be received, as specified in sections 3.1.5.6 and 3.1.5.7, respectively.

Otherwise, the client MUST wait for a $H (Header) packet to be received, as specified in section 3.1.5.7.

3.1.5.5 Receiving a $P (Packet-Pair) Packet

The client MUST verify that the $P packet adheres to the syntax specified in section 2.2.3.7.

If this is the first $P packet received, the client SHOULD start measuring the time until the second $P packet is received. The client MUST now wait for the second $P packet to be received, then process the rules in section 3.1.5.5 again.

If this is the second $P packet received, the client can use the time lapsed between receiving the first $P packet and the second $P packet to compute the bit rate at which the second $P packet was transferred. The client SHOULD make this information available to a higher layer. The client MUST now wait for the third $P packet to be received, then process the rules in section 3.1.5.5 again.

If this is the third $P packet received and the server's product name is "Cougar", as determined by the Server-Version variable, then the client MUST wait for the $M (Metadata) packet to be received, as specified in section 3.1.5.6.

Otherwise, the client MUST wait for a $H (Header) packet to be received, as specified in section 3.1.5.7.

3.1.5.6 Receiving a $M (Metadata) Packet

The client MUST verify that the $M packet adheres to the syntax specified in section 2.2.3.6.

If the **AFFlags** field in the MMS data packet (section 2.2.3.1.2) specifies that the metadata is being transmitted as multiple $M packets, and this is not the last $M packet, then the client MUST wait until the next $M packet is received and then process the rules in section 3.1.5.6 again.

When the last $M packet has been received, the payloads in each $M packet MUST be reassembled before processing can continue.

The client MUST set the Playlist-gen-id variable in the abstract data model to the value specified by the playlist-gen-id field in the $M packet.

The client SHOULD make the metadata available to a higher layer.

Otherwise, the client MUST wait for a $H (Header) packet to be received, as specified in section 3.1.5.7.

3.1.5.7 Receiving a $H (Header) Packet

The client MUST verify that the $H packet adheres to the syntax specified in section 2.2.3.5.

If the **AFFlags** field in the MMS data packet (section 2.2.3.1.2) specifies that the ASF header is being transmitted as multiple $H packets, and this is not the last $H packet, then the client MUST wait until the next $H packet is received and then process the rules in section 3.1.5.7 again.

When the last $H packet has been received, the payloads in each $H packet MUST be reassembled before processing can continue.

The client SHOULD make the ASF header available to a higher layer.

If the value of the State variable is INIT or IDLE, the client MUST wait until a higher-layer triggered event occurs.

If the value of the State variable is STREAMING, the client MUST wait for a $D (Data) packet or a $E (End-of-Stream Notification) packet, as specified in sections 3.1.5.11 and 3.1.5.12, respectively.

3.1.5.8 Receiving a Pipeline Response

The client MUST first follow the steps specified in section 3.1.5.2.

If this is the response to the first Pipeline request, the client MUST now wait for a $T (Test Data Notification) packet to be received as specified in section 3.1.5.9.

If this is the response to the second Pipeline request, and the value of the pipeline-result (section 2.2.1.4.17) token on the Pragma header is 1, then the client SHOULD set the Mode variable to specify that the pipelined mode of the protocol is used.

Otherwise, the Mode variable MUST be set to specify that the non-pipelined mode mode of the protocol is used.

Next, the value of the Pipeline-Test-Allowed variable MUST be set to 0, and then the client MUST proceed with sending the Play request as specified in section 3.1.4.3.1.

3.1.5.9 Receiving a $T (Test Data Notification) Packet

The client MUST verify that the $T packet adheres to the syntax specified in section 2.2.3.8.

The client MUST then wait for the response to the second Pipeline request to be received as specified in section 3.1.5.8.

3.1.5.10 Receiving a Play Response

The client MUST first follow the steps specified in section 3.1.5.2.

The value of the State variable MUST be set to STREAMING.

If the response includes a Pragma header with an xResetStrm (section 2.2.1.4.37) token, the client SHOULD treat it the same way it would treat receiving a $C (Stream Change Notification) packet with a zero-value **Reason** field, as specified in section 3.1.5.13. However, unlike the case of a playlist entry change, the client MUST NOT send a PlayNextEntry request in this situation.

If the server's product name is "Cougar" and the version number is greater than or equal to 9.0, according to the Server-Version variable, then the client MUST wait for the $M (Metadata) packet to be received as specified in section 3.1.5.6.

Otherwise, if the server's version number is greater than or equal to 9.0, then the client MUST wait for either a $M (Metadata) packet or a $H (Header) packet to be received as specified in sections 3.1.5.6 and 3.1.5.7, respectively.

Otherwise, the client MUST wait for a $H (Header) packet to be received as specified in section 3.1.5.7.

3.1.5.11 Receiving a $D (Data) Packet

The client MUST verify that the $D packet adheres to the syntax specified in section 2.2.3.3.

If the Keepalive timer is running, it MUST be stopped.

The client SHOULD make the ASF packet that is contained in the $D packet available to the higher layer.

The client MUST be prepared to receive either another $D packet or a $E (End-of-Stream Notification) packet to be received. More information about how to process these packets is specified in sections 3.1.5.11 and 3.1.5.12, respectively. If a response to a request is pending, for example, a SelectStream response, the client MUST also be prepared to receive that response.

3.1.5.12 Receiving a $E (End-of-Stream Notification) Packet

The client MUST verify that the $E packet adheres to the syntax specified in section 2.2.3.4.

If the value of the State variable is STREAMING, or if the value of the Reason field in the $E packet specifies an error HRESULT code, then the client MUST report this event to the higher layer.

If the value of the State variable is STREAMING, and the value of the Mode variable specifies that the pipelined mode of the protocol is used, then the client MUST send a Log request (section 2.2.2.4) to the server on the same TCP connection that the $E packet was received on.

If the value of the State variable is STREAMING, and the non-pipelined mode of the protocol is used, then the client MUST establish a new TCP connection to the server. The new connection MUST be to the same TCP port as the connection on which the $E packet was received. The client MUST then send a Log request to the server on the new TCP connection.

If the server's product name is "Cougar", and its version number is greater than or equal to 9.0, as determined by the Server-Version variable, then the above-mentioned Log request SHOULD contain a logging message in XML format as specified in [MS-WMLOG]. If the client will submit remote-log remote events using the SendEvent request (section 2.2.2.9), then the logging information included in the Log request MUST be a "streaming" log, as specified in [MS-WMLOG]. Otherwise, the logging information MUST be a "legacy" style log, as specified in [MS-WMLOG].

Next, the value of the State variable MUST be set to IDLE.

If the Keepalive timer is running, it MUST be restarted. If it is not running, it MUST be started.

If the value of the **Reason** field in the $E packet is 1, the client MUST be prepared to receive a $C packet. More information about how to process that packet is specified in section 3.1.5.13.

The client MUST also be prepared to receive either another $E packet or a Log response. More information about how to process the $E packet is specified in section 3.1.5.12, and more information about how to process the Log response is specified in section 3.1.5.14.

3.1.5.13 Receiving a $C (Stream Change Notification) Packet

The client MUST verify that the $C packet adheres to the syntax specified in section 2.2.3.2.

If the value of the **Reason** field in the $C packet specifies an error **HRESULT** code, then the client MUST report this event to the higher layer.

If the value of the **Reason** field in the $C packet is equal to 0x400D14BE, then the client MUST set the value of the State variable to IDLE and report this event to the higher layer.

If the value of the **Reason** field in the $C packet is not equal to 0x400D14BE, and does not specify an error **HRESULT** code, and if the Server-Features variable specifies that the server will use the com.microsoft.wm.predstrm feature defined in section 2.2.1.7.2, then the value of the State variable MUST be set to STREAMING. In any other case, the value of the State variable MUST be IDLE.

If the value of the State variable is IDLE and the Keepalive timer is running, it MUST be restarted. If the value of the State variable is IDLE and the Keepalive timer is not running, it MUST be started.

If the value of the **Reason** field in the $C packet is equal to 0x400D14BE, then the client MUST wait for a higher-layer event. No packets are expected at this point.

Otherwise, if the server's product name is "Cougar" and the version number is greater than or equal to 9.0, according to the Server-Version variable, then the client MUST wait for the $M (Metadata) packet to be received. More information about how to process that packet is specified in section 3.1.5.6.

Otherwise, if the server's version number is greater than or equal to 9.0, then the client MUST wait for either a $M (Metadata) packet or a $H (Header) packet to be received. More information about how to process these packets is specified in sections 3.1.5.6 and 3.1.5.7, respectively.

Otherwise, the client MUST wait for a $H (Header) packet to be received. More information about how to process that packet is specified in section 3.1.5.7.

3.1.5.14 Receiving a Log Response

The client MUST first follow the steps specified in section 3.1.5.2.

If the non-pipelined mode of the protocol is used, the TCP connection on which the Log response was received SHOULD be closed.

Next, the client MUST continue waiting for the next response (if any are pending), next packet, or next higher-layer event.

3.1.5.15 Receiving a KeepAlive Response

The client MUST first follow the steps specified in section 3.1.5.2.

If the non-pipelined mode of the protocol is used, the TCP connection on which the KeepAlive response was received SHOULD be closed.

Next, the client MUST continue waiting for the next packet or next higher-layer event.

3.1.5.16 Receiving a SelectStream Response

The client MUST first follow the steps specified in section 3.1.5.2.

If the non-pipelined mode of the protocol is used, the TCP connection on which the SelectStream response was received SHOULD be closed.

Next, the client MUST continue waiting for the next response (if any are pending), next packet, or next higher-layer event.

3.1.5.17 Receiving a PlayNextEntry Response

The client MUST first follow the steps specified in section 3.1.5.2.

The client MUST wait for the next packet to be received. The next packet could be any packet type, except that $M packets are only expected if the server's version number is greater than or equal to 9.0.

3.1.5.18 Receiving a Stop Response

The client MUST first follow the steps specified in section 3.1.5.2.

The value of the State variable MUST be set to IDLE.

If the Keepalive timer is running, it MUST be restarted. If it is not running, it MUST be started.

Next, the client MUST continue waiting for the response to the Log request. Information about how to process the response is specified in section 3.1.5.14.

3.1.5.19 Receiving a SendEvent Response

The client MUST first follow the steps specified in section 3.1.5.2.

The TCP connection on which the SendEvent response was received SHOULD be closed.

Next, the client MUST continue waiting for the next response (if any are pending), next packet, or next higher-layer event.

3.1.6 Timer Events

3.1.6.1 KeepAlive Timer Expires

When the KeepAlive timer expires, the following actions MUST take place.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, or if the TCP connection used for the most recent Play (section 2.2.2.6) or PlayNextEntry request (section 2.2.2.7) is no longer open, then the client MUST establish a new TCP connection to the server and to the same TCP port used for the Play request.

The client MUST send a KeepAlive request (section 2.2.2.3). If the value of the Mode variable specifies that the pipelined mode of the protocol is used, the request MUST be sent on either the same TCP connection used for the most recent Play or PlayNextEntry request, or if that connection is no longer open, on the TCP connection that was newly established according to the rules above.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, the KeepAlive request MUST be sent on the TCP connection that was newly established according to the rules above.

The KeepAlive timer MUST be restarted.

After having sent the request, the client MUST wait for the response to be received. More information about how to process the response is specified in section 3.1.5.15.

3.1.7 Other Local Events

None defined.

3.2 Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Client-ID: The value of this variable is an identifier assigned by the server for the current streaming "session". The initial value is undefined.

Client-features: This variable stores the capabilities that the client specified on the most recently received Supported header.

Client-Version: This variable stores the client product name and the major and minor version number that the client specified on the most recently received User-Agent header.

Mode: This variable specifies whether the protocol is in pipelined mode or non-pipelined mode mode.

Playlist-gen-id: The value of this variable is an identifier assigned by the server to identify the current playlist entry. The default value is 0.

State: This variable stores the state of the streaming session identified by the Client-ID variable. Possible values are INIT, STREAMING, WAITING, and IDLE.

Note The above conceptual data can be implemented by using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

3.2.2 Timers

Idle-Timeout: This timer is used for cleaning up an unused session state. If no requests are received from the client, the Idle-Timeout timer will expire and the server is then free to delete the session state. The minimum allowed value for the time-out period is 10 seconds.

Pipeline-Test: This timer is used as part of the processing of pipeline requests. The minimum allowed value for the time-out period is 100 milliseconds.

3.2.3 Initialization

Initialization of the protocol occurs when a Describe (section 2.2.2.1) or Play (section 2.2.2.6) request is received and the request did not specify a client-id (section 2.2.1.4.5) token on the Pragma header or the value of the token did not match the value of the Client-ID variable.

The variables defined by the abstract data model MUST initially assume their default values. Variables that do not have a default defined MUST be initialized as follows:

The value of the Client-ID token SHOULD be assigned to a random number. If the server allows multiple simultaneous streaming sessions, each instance of the protocol state MUST use a different value for the Client-ID token.

The State variable MUST be set to INIT.

The Mode variable MUST be set to non-pipelined mode.

The Client-features variable MUST be set to specify support for com.microsoft.wm.predstrm (see section 2.2.1.7.2) and for com.microsoft.wm.sswitch (see section 2.2.1.7.4).

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Notification That the Last $D Packet Has Been Sent

When the higher layer notifies the server that the last $D packet has been sent, the server MUST send a $E (End-of-Stream Notification) packet.

The higher layer MUST specify if a new ASF header is forthcoming, that is, there are additional entries in the server-side playlist, or whether this was the last entry in the playlist, so that the Reason field in the $E packet can be filled in correctly.

If the value of the **Reason** field in the $E packet is 0, and the server is using Chunked Transfer Coding, then the server MUST follow the $E packet by a zero-length chunk.

If the value of the **Reason** field in the $E packet is 0, and the server is not using Chunked Transfer Coding, and the server intends to keep the TCP connection open after having sent the $E packet, then the server SHOULD follow the $E packet by at least 511 more $E packets. The additional 511 $E packets reduce the likelihood that intermediate proxy servers delay the delivery of the $E packet to the client.

The value of the State variable in the abstract data model MUST be changed to IDLE.

The Idle-Timeout timer MUST be started.

3.2.4.2 Notification That a New ASF Header Is Available

As a prerequisite for this event, the higher layer MUST already have notified the server that it has sent the last $D packet for the previous playlist entry, as specified in section 3.2.4.1.

If the most recently received SelectStream request, if any, specified the client-lag token on a Pragma header, then the server SHOULD delay sending the $C (Stream Change Notification) packet by the time amount specified by the client-lag token.

The server MUST now send a $C (Stream Change Notification) packet to the client.

The server MUST change the value of the Playlist-gen-id variable in the abstract data model, such that each playlist entry gets a different identifier.

If the client version is greater than or equal to 9.0, as indicated by the value of the Client-Version variable, and the server previously specified the value of the server-token parameter on the Server header as "Cougar", then the server MUST now send one or more $M (Metadata) packets (section 2.2.3.6). The **playlist-gen-id** field in the $M packet MUST be set to the value of the Playlist-gen-id variable in the abstract data model. For more information about $M packets, see section 2.2.3.6

The server MUST then send the ASF header of the new playlist entry, encoded as one or more $H (Header) packets (section 2.2.3.5).

If the value of the Client-features variable specifies that the client supports the com.microsoft.wm.predstrm (see section 2.2.1.7.2) feature, and the server has also specified that it supports this feature, and the server has received a Play request or a SelectStream request for the previous playlist entry, then the server MUST select suitable streams from the ASF header of the new playlist entry, and start streaming $D packets to the client. In this case, the State variable MUST be set to STREAMING and the Idle-Timeout timer MUST be stopped.

Otherwise, the State variable MUST be set to WAITING and if the Idle-Timeout timer is not running, it MUST be started.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Receiving a Request (All Request Types)

This section specifies common steps that MUST be performed whenever the server receives a request from a client. These steps MUST be performed prior to any processing that is specific to a particular request type.

The server MUST inspect the User-Agent (section 2.2.1.8) header in the request. If the User-Agent header is missing, or if the *client-token* parameter on the User-Agent header does not match any of the values used by this protocol, as specified in section 2.2.1.8, then the client is not compatible with the Windows Media HTTP Streaming Protocol and is probably a regular HTTP Web client. Because the Windows Media HTTP Streaming Protocol does not interoperate with incompatible clients, this situation MUST be treated as an error. The server MAY respond to the request with an HTTP error status code, or it MAY respond with a success code, but return data that redirects the Web client to some other suitable URL.<64>

The server MUST validate that the request is of one of the types specified in section 2.2.2 and that the request is using the appropriate HTTP request method. If the validation fails, the server MUST respond with some HTTP error status code, such as 400, or 501, as appropriate.

If the higher layer requires the client to authenticate itself, then the server MUST process the Authorization header (section 14.8 of [RFC2616]), if it is present in the request. If the server is acting as a proxy server, then it MUST process the Proxy-Authorization header (section 14.34 of [RFC2616]) instead of the Authorization header.

If it is necessary to send an authentication challenge to the client (for example, because the Authorization header was missing, or specified incorrect credentials), then the server SHOULD use one of the authentication schemes that the client listed on the X-Accept-Authentication (section 2.2.1.9) header in the request, if that header is present. If the server is acting as a proxy server, then it MUST use the X-Accept-Proxy-Authentication (section 2.2.1.10) header instead of the X-Accept-Authentication header.

If the server sends an authentication challenge to the client, it MUST be specified using the WWW-Authenticate header (section 14.47 of [RFC2616]), and the status code of the response MUST be 401. If the server is acting as a proxy server, then it MUST specify the Proxy-Authenticate header (section 14.33 of [RFC2616]) instead of the WWW-Authenticate header, and the status code of the response MUST be 407.

After having sent a response with status codes 401 or 407, if the authentication scheme used in the authentication challenge is NTLM [NTLM], then the server SHOULD NOT close the TCP connection to the client.

If the server is not sending a response with an error status code, and if the request has a client-id token on a Pragma header, the server MUST load the state that has a Client-ID variable with the same value as the value of the client-id token. If the matching state cannot be found, and the request type is not Describe or Play, then this MUST be treated as an error. The server MUST respond to the error with some suitable HTTP error status code, such as 400. For details about what to do in the case of Describe and Play requests, see sections 3.2.5.4 and 3.2.5.6, respectively.

If the request does not have a client-id token on a Pragma header, and the request type is not GetContentInfo, Describe, Play, or SendEvent, then this MUST be treated as an error. The server MUST respond to the error with some suitable HTTP error status code, such as 400.

The client version information (client-token and major and minor version number) on the User-Agent header MUST be saved in the Client-Version variable in the abstract data model.

The server MUST process Supported header, if present. Each feature token on the header MUST be added to the Client-features variable in the abstract data model. If the header is present, any feature token not listed on the header MUST be removed from the Client-features variable.

If the Idle-Timeout timer is running, it MUST be stopped.

3.2.5.2 Sending a Response (All Request Types)

This section specifies common steps that MUST be performed whenever the server sends a response to a request from the client.

The response sent by the server MUST NOT specify any of the headers and tokens defined in section 2.2.1 that are defined only for use in requests.

The response MUST NOT specify any of the headers and tokens specified in section 2.2.1 that are defined only for use in responses sent to clients with a version number higher than the version number specified by the Client-Version variable in the abstract data model.

The server MUST specify the Server (section 2.2.1.5) header in the response. If the server is acting as a live encoder, the value of the server-token parameter on the Server header MUST be set to "Rex". Otherwise, it MUST be set to "Cougar".

The server SHOULD specify the Supported (section 2.2.1.7) header in the response. If specified, the header MUST correctly reflect the features that are supported by the server in the response if the version number specified by the Client-Version variable is greater than or equal to 9.0. If the Supported header is specified, the header MUST correctly reflect the features that are supported by the server. The server MUST support the following feature: com.microsoft.wm.sswitch (section 2.2.1.7.4). The server SHOULD support the following features: com.microsoft.wm.fastcache (section 2.2.1.7.1), com.microsoft.wm.predstrm (section 2.2.1.7.2), com.microsoft.wm.srvppair (section 2.2.1.7.3), and com.microsoft.wm.startupprofile (section 2.2.1.7.5).<65>

The server SHOULD specify the Cache-Control (section 2.2.1.1) header in the response to prevent caching proxy servers that do not understand the Windows Media HTTP Streaming Protocol from caching the response.

The server SHOULD specify the Content-Type header if the response is to a GET request, unless the message body of the response will be empty.

If the Client-ID variable in the abstract data model has been assigned a value, the client MUST specify a client-id token on a Pragma header in the response. The value of the client-id token MUST be equal to the value of the Client-ID variable.

If the response includes a client-id token on the Pragma header, then the Pragma header SHOULD also include the timeout (section 2.2.1.4.29) token. The value of the timeout token MUST be set to a value less than or equal to the timeout interval of the Idle-Timeout timer. It is recommended that the value of the token be at least a few seconds less than the timeout interval, to allow for processing delays and network delays.

If the Playlist-gen-id variable in the abstract data model has a nonzero value, the client MUST specify a playlist-gen-id token on a Pragma header in the response. The value of the playlist-gen-id token MUST be equal to the value of the Playlist-gen-id variable.

The Idle-Timeout timer MUST be started, unless the value of the State variable in the abstract data model is STREAMING.

3.2.5.3 Receiving a GetContentInfo Request

The server MUST first follow the steps specified in section 3.2.5.1.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not, then this is an error, and the server MUST respond with some suitable HTTP error status code, such as 404.

The GetContentInfo request does not require any server state. Hence, if the client-id token on the Pragma header is missing, this MUST NOT be treated as an error.

The GetContentInfo response MUST follow the rules in section 3.2.5.2 and 2.2.2.2.

If the value of the State variable is STREAMING and the GetContentInfo request was received on the same TCP connection on which the server is currently sending $D packets, then server MUST continue sending $D packets and MUST send the GetContentInfo response only after the zero-length chunk has been sent.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, then the server SHOULD close the TCP connection after sending the GetContentInfo response.

3.2.5.4 Receiving a Describe Request

The server MUST first follow the steps specified in section 3.2.5.1.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not, then this is an error, and the server MUST respond with some suitable HTTP error status code, such as 404.

The server MUST check with the higher layer to determine if the client shall be redirected to a different server or to a proxy server. (The presence, or absence, of a Via header in the request can be used to determine if the request was delivered directly by the client or through a proxy server, as specified in [RFC2616] section 14.45.)

If the higher layer indicates that the client shall be redirected to another server, then the server MUST respond with status code 302. If the client shall be redirected to a proxy server, then the server SHOULD respond with status code 305. In both cases, the URL of the server, or proxy server, MUST be specified on the Location header in the response, as specified in [RFC2616] section 14.30.

After having sent a response with status codes 302 or 305, the server MUST delete the session state, if any, and close the TCP connection to the client.

If the server is not sending a response with an error status code (for example, 302 or 305), and if the request does not specify a client-id (section 2.2.1.4.5) token on a Pragma header, then the server MUST create new state by performing the initialization procedure specified in section 3.2.3.

If the request specifies a client-id token on a Pragma header, but the value of that token does not match the value of the Client-ID variable in any instance of the server's state, then the server SHOULD create new state by performing the initialization procedure specified in section 3.2.3. The response MUST specify the xResetStrm (section 2.2.1.4.37) token on a Pragma header.

If the request includes a pipeline-experiment (section 2.2.1.4.15) token on the Pragma header, with the value set to 1, and the server supports the pipelined mode of the protocol, then the server SHOULD specify the pipeline-experiment token on the Pragma header in the response, and the value of the token SHOULD be 1. Otherwise, the server SHOULD NOT specify the pipeline-experiment token on the Pragma header.

If the request includes a packet-pair-experiment (section 2.2.1.4.14) token on the Pragma header, with the value set to 1, and the server supports sending $P (Packet Pair) packets, then the server SHOULD specify the packet-pair-experiment token on the Pragma header in the response, and the value of the token SHOULD be 1. Otherwise, the server SHOULD NOT specify the packet-pair-experiment token on the Pragma header.

The Describe response MUST follow the rules as specified in section 3.2.5.2 and 2.2.2.1.

The server MAY specify the version-info (section 2.2.1.4.31) token and the version-url (section 2.2.1.4.32) token on a Pragma header in the response, but SHOULD NOT specify these tokens when the version number specified by the Client-Version variable is greater than or equal to 7.0.

If the server specified the packet-pair-experiment token on the Pragma header with a value of 1, the message body of the Describe response MUST begin with three $P packets, and the server SHOULD deliver the $P packets to the TCP layer such that each $P packet is transmitted in a separate TCP segment. Otherwise, the server MUST NOT send $P packets to the client.

If the client implementation supports the pipelined mode of the protocol, the Describe request SHOULD include a pipeline-experiment (section 2.2.1.4.15) token on the Pragma header, with the value set to 1.

If the version number specified by the Client-Version variable is greater than or equal to 9.0, and if the server is specifying the value of the server-token parameter on the Server header as "Cougar", then the server MUST send one or more $M (Metadata) packets (section 2.2.3.6).

The message body of the Describe response MUST end with the ASF header of the current playlist entry, encoded as one or more $H (Header) packets (section 2.2.3.5).

3.2.5.5 Receiving a Pipeline Request

The server MUST first follow the steps specified in section 3.2.5.1.

If this is the first Pipeline request (section 2.2.2.5), the Pipeline-Test timer MUST be started. The recommended value for the timeout of this timer is 1 second.

The Pipeline response MUST follow the rules as specified in section 3.2.5.2 and 2.2.2.5.

If this is the first Pipeline request, the response MUST use Chunked Transfer Coding and the message body of the response MUST contain a $T (Test Data Notification) packet (section 2.2.3.8). The response MUST NOT specify a zero-length chunk.

If this is the second Pipeline request, the Pipeline response MUST specify the pipeline-result (section 2.2.1.4.17) token on a Pragma header, and the value of the token MUST be either 1 if the Pipeline-Test timer is still running, or 0 if the Pipeline-Test timer is not running.

If this is the second Pipeline request, and the Pipeline-Test timer is running, then the timer MUST be stopped. Also, in this case, the server MUST send the zero-length chunk, which will terminate the response to the first Pipeline request.

Details about Chunked Transfer Coding are as specified in [RFC2616] section 3.6.1.

For more information about the Pipeline request, see section 2.2.2.5.

3.2.5.6 Receiving a Play Request

The server MUST first follow the steps specified in section 3.2.5.1.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not, then this is an error, and the server MUST respond with some suitable HTTP error status code, such as 404.

If the request specifies a client-id (section 2.2.1.4.5) token on a Pragma header, and the value matches the value of a Client-ID variable in some instance of the server's state, and the value of the corresponding State variable is STREAMING, the server SHOULD treat this as an error and fail the request. The reason is that a Play request for a session that is already streaming could be an attempt to hijack a session.

If the request does not specify a client-id token on a Pragma header, then the server MUST create new state by performing the initialization procedure defined in section 3.2.3.

If the request specifies a client-id token on a Pragma header, but the value of that token does not match the value of the Client-ID variable in any instance of the server's state, then the server MUST either create new state by performing the initialization procedure specified in section 3.2.3 or fail the request. Also, in this case, if the request is successful, the response MUST specify the xResetStrm (section 2.2.1.4.37) token on a Pragma header.

The server MUST process the stream selection information specified on the stream-switch-entry (section 2.2.1.4.27) token on the Pragma header, if any.

If the request does not contain a stream-switch-entry token on a Pragma header, and the client version is less than or equal to 5.0, as specified by the client on the User-Agent header, and the value of the client-token parameter on the User-Agent header is "NSServer", then the server MUST assume that all streams in the content are selected.

Otherwise, if the request does not contain a stream-switch-entry token on a Pragma header, the server MUST assume that none of the streams in the content are selected.

The Play response MUST follow the rules as specified in section 3.2.5.2 and 2.2.2.6.

If the request specifies the version11-enabled (section 2.2.1.4.30) token on a Pragma header and the value of the token is 1, then the response SHOULD use HTTP 1.1 and Chunked Transfer Coding. Otherwise, Chunked Transfer Coding MUST NOT be used.

The value of the Mode variable in the abstract data model MUST be updated to record if the pipelined or non-pipelined mode of the protocol is used.

The server SHOULD specify the AccelBW (section 2.2.1.4.1) token, the AccelDuration (section 2.2.1.4.2) token, the BurstBW (section 2.2.1.4.3) token, and the BurstDuration (section 2.2.1.4.4) token on a Pragma header in the response if the client sent the corresponding tokens on a Pragma header in the Play request, and if the server supports changing the transmission rate based on the respective tokens in the request. Otherwise, a server SHOULD NOT specify the tokens in the response.

If the client has specified support for the com.microsoft.wm.startupprofile feature on the Supported header, as specified in section 2.2.1.7.5, then the server SHOULD specify the X-StartupProfile (section 2.2.1.12) header in the response.

The server MUST NOT specify the X-StartupProfile header in the response unless the client has specified support for the com.microsoft.wm.startupprofile feature.

The value of the State variable in the abstract data model MUST be set to STREAMING.

If the client version is greater than or equal to 9.0, as specified by the client on the User-Agent header, and the server is specifying the value of the server-token parameter on the Server header as "Cougar", then the server MUST send one or more $M (Metadata) packets (section 2.2.3.6).

The server MUST send the ASF header of the current playlist entry, encoded as one or more $H (Header) packets (section 2.2.3.5).

If the server specified the xResetStrm token on a Pragma header in the response, and the client version is less than 7.0, as specified by the client on the User-Agent header, and the value of the client-token parameter on the User-Agent header is "NSPlayer", and the client specified the request-context (section 2.2.1.4.23) token on the Pragma header, and the value of that token is 2, then the server MUST send a $E (End-Of-Stream Notification) packet with the **Reason** field set to 1, followed by a $C (Stream Change Notification) packet with the **Reason** field set to 0, followed by the ASF header of the current playlist entry, encoded as one or more $H packets.

If the server specified the xResetStrm token on a Pragma header in the response, and the client version is less than 7.0, as specified by the client on the User-Agent header, and the value of the client-token parameter on the User-Agent header is "NSServer", and the client specified the request-context token on the Pragma header, and the value of that token is 3, then the server MUST send a $E packet with the **Reason** field set to 1, followed by a $C packet with the **Reason** field set to 0, followed by the ASF header of the current playlist entry, encoded as one or more $H packets.

The server MUST then send the ASF packets of the current playlist entry, encoded as $D (Data) packets (section 2.2.3.3) packets. The ASF payloads in the ASF MUST be filtered such that only ASF payloads that belong to streams that have been selected by the client are included in the ASF packets.

3.2.5.7 Receiving a SelectStream Request

The server MUST first follow the steps specified in section 3.2.5.1.

The server MUST process the stream selection information specified on the stream-switch-entry (section 2.2.1.4.27) token on the Pragma header, if any. Any new streams that are transmitted as a result of the stream-switch-entry token MUST be transmitted beginning with an ASF key-frame payload. Any streams that are replaced by different streams MUST continue to be transmitted until the first ASF key-frame payload of the new stream is transmitted. How to determine if an ASF payload contains a key-frame is as specified in [ASF].

If the value of the State variable is WAITING, and the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, then the State variable MUST be set to STREAMING and the server MUST start sending $D (Data) packets on the TCP connection that was used for the most recently sent Play response.

The SelectStream response MUST follow the rules in sections 3.2.5.2 and 2.2.2.8.

If the value of the Mode variable specifies that the pipelined mode of the protocol is used, the SelectStream response SHOULD use Chunked Transfer Coding.

If the value of the State variable is STREAMING and the SelectStream request was received on the same TCP connection on which the server is currently sending $D packets, then server MUST continue sending $D packets after sending the SelectStream response.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, then the server SHOULD close the TCP connection after sending the SelectStream response.

3.2.5.8 Receiving a KeepAlive Request

The server MUST first follow the steps specified in section 3.2.5.1.

The KeepAlive response MUST follow the rules in section 3.2.5.2 and 2.2.2.3.

If the value of the State variable is STREAMING and the KeepAlive request was received on the same TCP connection on which the server is currently sending $D packets, then server MUST continue sending $D packets and MUST send the KeepAlive response only after the zero-length chunk has been sent.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, then the server SHOULD close the TCP connection after sending the KeepAlive response.

3.2.5.9 Receiving a PlayNextEntry Request

The server MUST first follow the steps specified in section 3.2.5.1.

The server MUST process the stream selection information specified on the stream-switch-entry (section 2.2.1.4.27) token on the Pragma header, if any. Any new streams that are transmitted as a result of the stream-switch-entry token MUST be transmitted beginning with an ASF key-frame payload. Any streams that are replaced by different streams MUST continue to be transmitted until the first ASF key-frame payload of the new stream is transmitted. Details about how to determine if an ASF payload contains a key-frame, are as specified in [ASF].

The PlayNextEntry response MUST follow the rules in section 3.2.5.2 and 2.2.2.7.

The server SHOULD specify the AccelBW (section 2.2.1.4.1) token, the AccelDuration (section 2.2.1.4.2) token, the BurstBW (section 2.2.1.4.3) token, and the BurstDuration (section 2.2.1.4.4) token, on a Pragma header in the response if the client sent the corresponding token(s) on a Pragma header in the Play request, and if the server supports changing the transmission rate based on the respective token(s) in the request. Otherwise, a server SHOULD NOT specify the token(s) in the response.

If the value of the State variable is WAITING, and if the client has specified support for the com.microsoft.wm.startupprofile feature on the Supported header, as specified in section 2.2.1.7.5, then the server SHOULD specify the X-StartupProfile (section 2.2.1.12) header in the response.

The server MUST NOT specify the X-StartupProfile header in the response unless the client has specified support for the com.microsoft.wm.startupprofile feature.

If the value of the State variable is WAITING, then the State variable MUST be set to STREAMING and the server MUST start sending $D (Data) packets.

3.2.5.10 Receiving a Stop Request

The server MUST first follow the steps defined in section 3.2.5.1.

If the value of the State variable in the abstract data model is STREAMING, then the server MUST stop sending $D packets and MUST send a zero-length chunk.

The State variable MUST be set to IDLE.

The Stop response MUST follow the rules in sections 3.2.5.2 and 2.2.2.10.

3.2.5.11 Receiving a Log Request

The server MUST first follow the steps defined in section 3.2.5.1.

The server SHOULD communicate the logging information submitted by the client to the higher layer.

The Log response MUST follow the rules in sections 3.2.5.2 and 2.2.2.4.

If the value of the State variable is STREAMING and the Log request was received on the same TCP connection on which the server is currently sending $D packets, then server MUST continue sending $D packets and MUST send the Log response only after the zero-length chunk has been sent.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, then the server SHOULD close the TCP connection after sending the Log response.

3.2.5.12 Receiving a SendEvent Request

The server MUST first follow the steps defined in section 3.2.5.1.

The server MUST check with the higher layer that the URL that the client specified in the request is valid. If it is not, then this is an error, and the server MUST respond with some suitable HTTP error status code, such as 404.

The SendEvent request does not require any server state. Hence, if the client-id token on the Pragma header is missing, this MUST NOT be treated as an error.

The server SHOULD communicate the logging information submitted by the client to the higher layer.

The SendEvent response MUST follow the rules in sections 3.2.5.2 and 2.2.2.9.

If the value of the State variable is STREAMING and the SendEvent request was received on the same TCP connection on which the server is currently sending $D packets, then server MUST continue sending $D packets and MUST send the SendEvent response only after the zero-length chunk has been sent.

If the value of the Mode variable specifies that the non-pipelined mode of the protocol is used, then the server SHOULD close the TCP connection after sending the SendEvent response.

3.2.6 Timer Events

3.2.6.1 Pipeline-Test Timer Expires

When the Pipeline-Test timer expires, the server MUST send a zero-length chunk to terminate the response to the first pipeline request (section 2.2.2.5).

The Pipeline-Test timer MUST remain in a stopped state.

3.2.6.2 Idle-Timeout Timer Expires

When the Idle-Timeout timer expires, the server MUST close any TCP connections that are still open to the client. After that, the server MUST delete the session state, invalidating the Client-ID variable.

3.2.7 Other Local Events

3.2.7.1 TCP Connection Is Closed by the Client

If the value of the State variable in the abstract data model is STREAMING, and the TCP connection that was closed is the one that was used for sending $D packets to the client, then the State variable MUST be set to IDLE, and the Idle-Timeout timer MUST be started.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Windows Media HTTP Streaming Protocol.

4.1 Server States in Non-Pipelined Mode

The following diagram shows the server states:

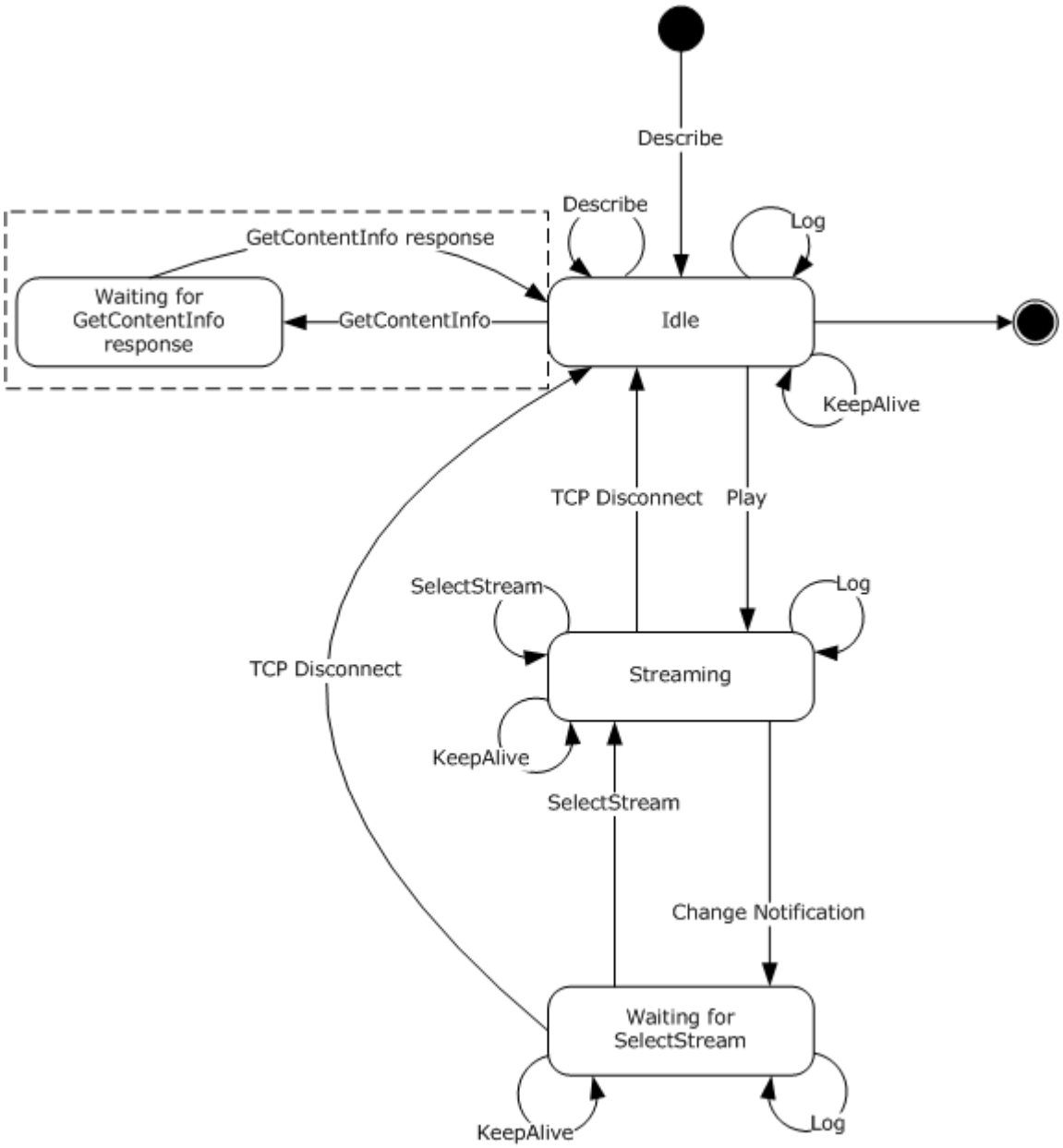


Figure 1: WMS HTTP 1.0 Caching Proxy Server States and Sequencing

Understanding the server states diagram:

- All requests illustrated in the diagram, with the exception of Change Notification, are sent by the client to the server in the context of an HTTP request message using either the GET or POST methods. All server responses are in the context of an HTTP response message.
- Requests that loop back on a given state indicate that the request is sent by the client to the server on a separate TCP connection. These requests do not interrupt other requests or sessions and can be sent multiple times without changing the server state.
- The presence of a caching proxy server introduces an additional state, indicated by the dotted box in the above server states diagram. This state is not applicable during direct server and client interaction. For more information, see section 4.12.
- A TCP connection is initiated with a server when a client issues its first Describe request containing a Uniform Resource Identifier (URI) corresponding to the virtual publishing point for content on the server (the content can be stored or live.) This first Describe request initiates the server Idle state.
- In response to the initial Describe request, a client-id token value is created by the server and is used by the client throughout that individual streaming session. However, the server can send a new client-id that must be subsequently used by the client.
- TCP Disconnect occurs when the connection used for the Play request and server response is closed. The server may close the TCP connection after sending the response or because of an error or time-out that occurred on the server. Clients may also close the TCP connection in stop and resume scenarios. For more information, see section 4.7. A TCP Disconnect causes the server to change from its current state to the Idle state.
- Change Notification, which contains the $C (Stream Change Notification) packet, is sent by the server on the same TCP connection as the Play request. A client usually sends Describe, Log, SelectStream, and KeepAlive requests on separate connections. A TCP Disconnect on one of these separate connections does not cause a state transition. All transitions from Idle state to final state are server-specific implementations.
- By default when the non-pipelined mode of the protocol is used, TCP connections are not reused. Therefore, the connection used for the Describe request is closed by the server when the server sends its response. However, the client can keep this connection open by including a Connection: Keep-Alive header with the request. If the server can comply, it notifies the client by including a Connection: Keep-Alive header in response. In this case, the client can send subsequent requests, such as the Play request, on the same connection. The presence of the Connection: Keep-Alive header does not trigger a state transition or otherwise change the state of the protocol. The Connection: Keep-Alive header is described in Hypertext Transfer Protocol 1.1, [RFC2616].

For more information about sequencing associated with the various server states, see section 3.2.

4.2 Server States in Pipelined Mode

The following illustration shows the server states and sequencing:

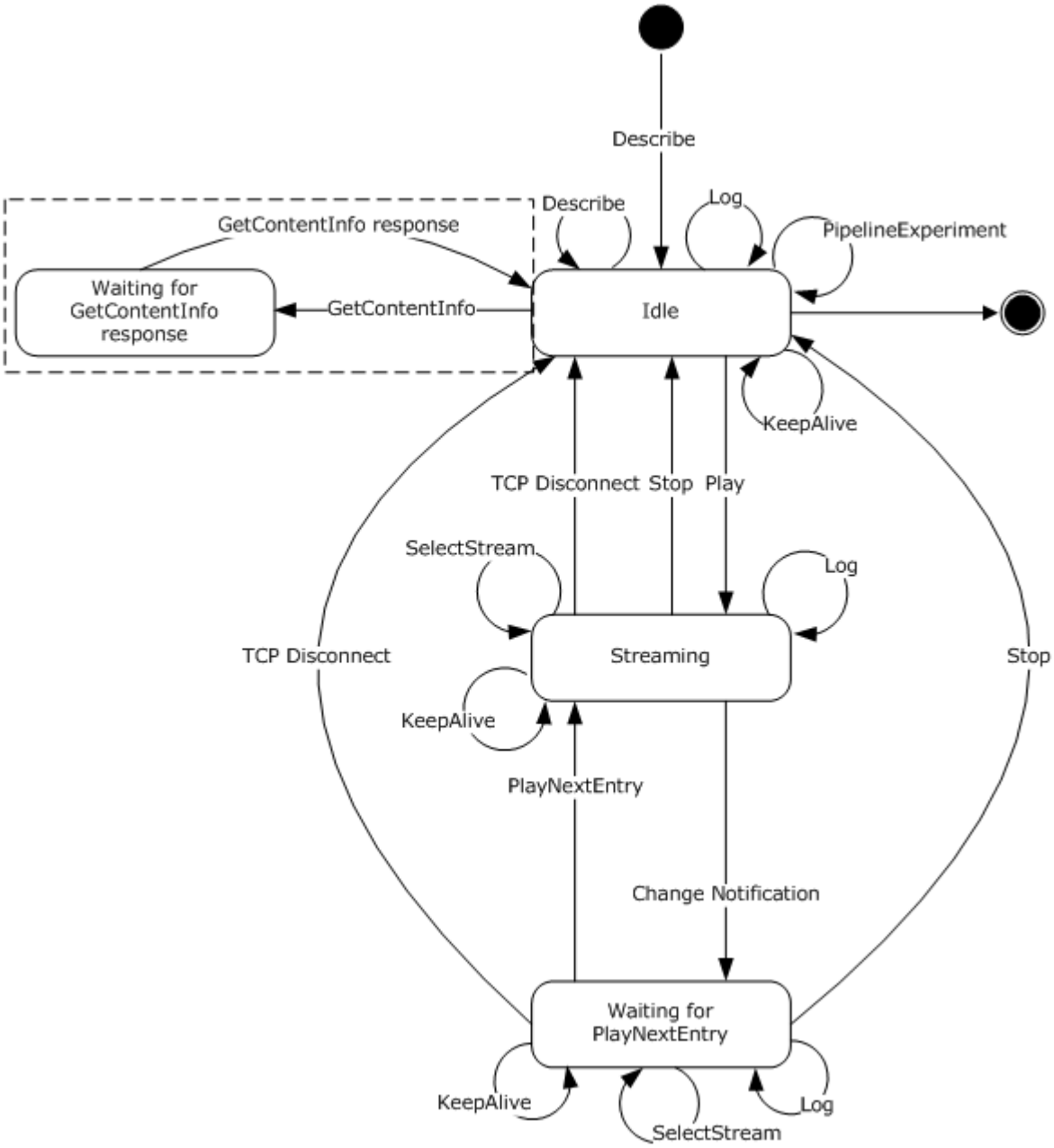


Figure 2: WMS HTTP 1.1 Caching Proxy Server States and Sequencing

Understanding the server states diagram:

- All requests illustrated in the diagram are sent by the client to the server in the context of an HTTP request message by using either the GET, POST, or OPTIONS methods. All server responses are in the context of an HTTP response message. For more information about the request types and the associated methods, see section 2.2.2.2.

- Requests that loop back on a state indicate that the request is sent by the client to the server on the same TCP connection. The response to the client is not returned until the previous request is terminated by the server.
- The presence of a caching proxy server introduces an additional state, indicated by the dotted box in the above server states diagram. This state is not applicable during direct server and client interaction. For more information, see section 4.12.
- A TCP connection is initiated with a server when a client issues its first Describe request containing a URI corresponding to the virtual publishing point for content on the server. (The content can be stored or live.) This first Describe request initiates the server Idle state.
- In response to the initial Describe request, a client-id token value is created by the server and is used by the client throughout that individual streaming session. However, the server can send a new client-id that MUST be subsequently used by the client.
- By default when the pipelined mode of the protocol is used, the TCP connection created with the initial Describe request is kept open. If the server closes the connection, it notifies the client by including a Connection: Close header with the response. For more information about Connection: Close header, see HTTP 1.1, as specified in [RFC2616]. A TCP Disconnect causes the server to change from its current state to the Idle state; however, TCP connections are not required to be disconnected for the protocol to be in the Idle state. All transitions from Idle state to final state are server-specific implementations.
- In an auto-reconnect scenario, the server maintains the session state. The client must maintain the point or offset from which to resume playback. This offset is sent in the stream-offset token on the Pragma header with the next Play request. The server does not expect a new Describe request because it has already sent the $H (Header) packet for the media stream. Also, the server does not need to conduct the packet-pair experiment again to determine the bit rate of the new connection. The sequence is identical to a server resuming from a Pause state. For more information, see section 4.8.

For more information about sequencing associated with the various server states, see section 3.2.

4.3 Packet-Pair Bandwidth Estimation

Packet-Pair is a technique for estimating the bandwidth of a streaming media connection over the Internet.

To estimate bandwidth, the server sends two or more consecutive packets of highly entropic data, and the client estimates the bandwidth by measuring the difference between the times that it receives the packets. This method is usually reliable; however, if the client traverses a Network Address Translation (NAT), firewall, or proxy server, the packet-pair bandwidth measurement might be inaccurate.

This technique is not necessary for devices with known bandwidths, such as cellular phones.
Note The client's use of the packet-pair data to estimate the network bandwidth is implementation-specific.

The following sequence occurs between a client and server when conducting the packet-pair experiment. The sequencing applies to both the pipelined mode and non-pipelined modes of the protocol.

1. The client sends a Describe request and includes:
   - The Pragma header field with the token packet-pair-experiment=1.

- The Supported header with the token com.microsoft.wm.srvppair.

2. If the server is able to conduct the packet-pair experiment, the server response includes:
   - The Pragma header field with the token packet-pair-experiment=1.
   - The Supported header with the token com.microsoft.wm.srvppair.
3. The server sends the following packets in the response body:
   - $P (Packet-Pair) packets (section 2.2.3.7).
   - A $M (Metadata) packet.
   - A $H (Header) packet.

The following illustration shows the sequence described above.

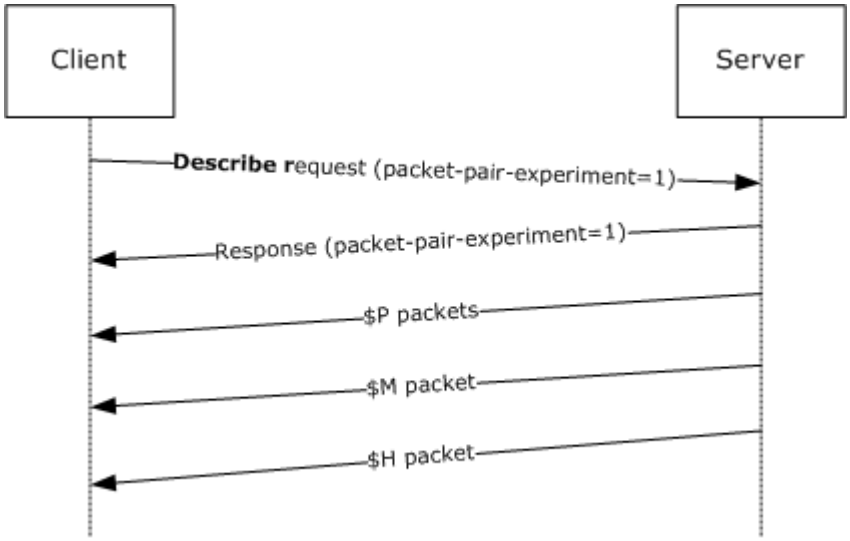


Figure 3: Packet-Pair sequence

The following example shows a client's packet-pair experiment request:

```
GET /test.asf HTTP/1.0
Accept: */*
User-Agent: NSPlayer/10.0.0.3802
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest, Basic
Pragma: no-cache,rate=1.000,stream-time=0,stream-offset=0:0,
packet-num=4294967295,max-duration=0
Pragma: packet-pair-experiment=1
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.predstrm, com.microsoft.wm.startupprofile
```

The following example shows the server response:

```
HTTP/1.0 200 OK
Content-Length: 5227
Content-Type: application/vnd.ms.wms-hdr.asfv1
Server: Cougar/9.01.01.3814
Pragma: packet-pair-experiment=1, no-cache, client-id=2064325698, xResetStrm=1,
features="seekable,stridable", timeout=60000
Cache-Control: no-cache, x-wms-content-size=638066, x-wms-event-subscription="remote-log"
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
com.microsoft.wm.predstrm, com.microsoft.wm.fastcache,
com.microsoft.wm.startupprofile

$P......<packet-pair payload>
$P......<packet-pair payload>
$P......<packet-pair payload>
$M......<metadata payload>
$H......<header payload>
```

4.4 Playlist Streaming

The following sequence occurs between a client and server during playlist streaming using predictive stream selection. The sequencing applies to both the pipelined mode and non-pipelined mode of the protocol.

1. The client sends a Describe request to retrieve the ASF header.
2. The server responds with a $H (Header) packet.
3. The client sends a Play request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets of the first playlist element.
5. After all data packets of the first playlist element have been sent to the client, the server sends an $E (End-of-Stream Notification) packet, and the server sends a $C (Stream Change Notification) packet.
6. The client sends a Log request for the first element of the playlist.

   Note When the non-pipelined mode of the protocol is used, the client uses a separate TCP connection to send the Log request to the server; therefore, the server can respond to the Log request while still sending stream data through the previously opened connection.

   Note When the pipelined mode of the protocol is used, the client sends requests to the server on the same TCP connection. The response to the client is not returned until the previous request is terminated by the server.
7. The server sends $H (Header) and $D (Data) packets of the next element in the playlist.
8. The client notifies the server that it is rendering the second element of the playlist by sending a SelectStream request. The client might select other streams than those predictably selected by the server. For information about Predicative Stream Selection, see section 4.5.2.

   Note When the non-pipelined mode of the protocol is used, the client uses a separate TCP connection for the SelectStream request. The server responds immediately to this request while still sending stream data through the previously opened connection.

Note When the pipelined mode of the protocol is used, the client uses the same TCP connection; therefore, the server does not respond to the SelectStream request until the previous request is terminated by the server.

9. The server responds to the Log and SelectStream requests.

10. The server continues looping playlist element data packets (step 5) until the end of the playlist is reached.

The following illustration shows the sequence described above:

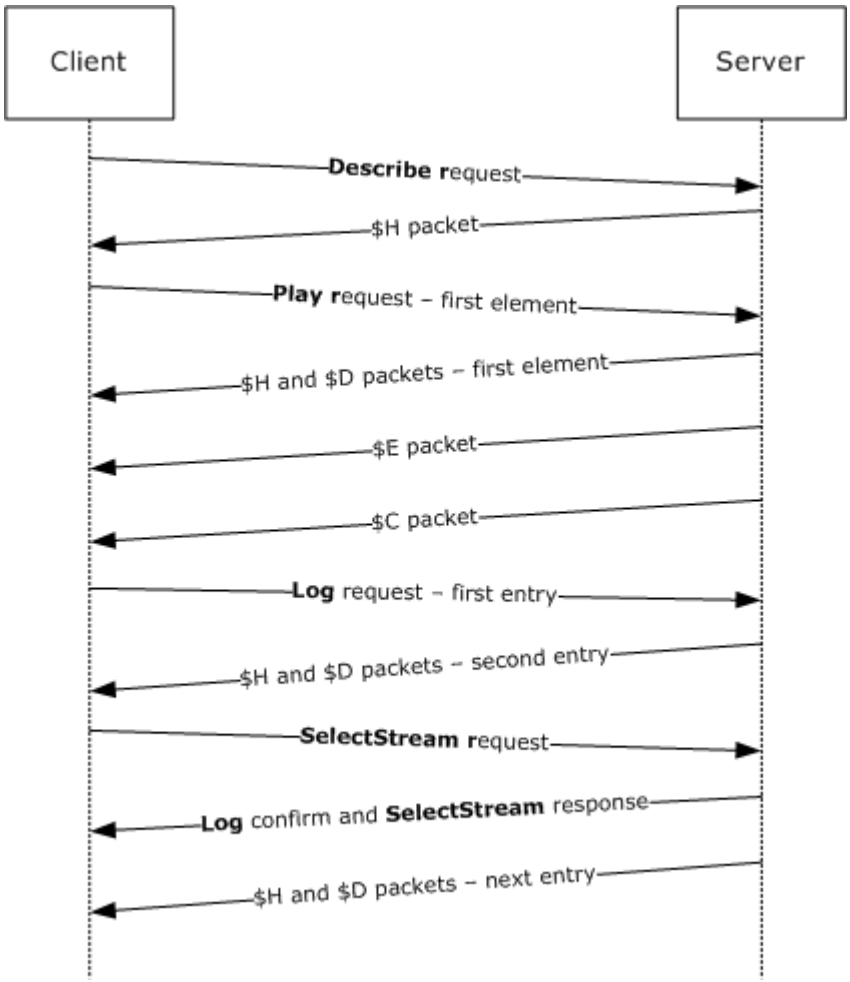


Figure 4: Sequencing during normal playlist streaming

4.5 Server-Side Playlist Streaming

The following sequence occurs between a client and server during a server-side playlist switch using either the pipelined mode or the non-pipelined mode of the protocol.

The client can seek and skip to a new entry within a server-side playlist. For more information, see section 4.5.1.

1. The client sends a Describe request to retrieve the ASF header.
2. The server responds with a $H (Header) packet.
3. The client sends a Play request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets of the first playlist element.
5. After all data packets of the current playlist element have been sent to the client, the server sends an $E (End-of-Stream Notification) packet, with the **Reason** field set to S_FALSE (0x00000001).
6. The server sends a $C (Stream Change Notification) packet to the client. This signal, like the data stream, is sent as part of the response body for the last GET request sent by the client.
7. The server sends the $H (Header) and $M (Metadata) packets for the next playlist entry.
8. The client sends a Log request for the previous entry in the playlist.

   Note When the non-pipelined mode of the protocol is used, the client sends this POST request over a separate TCP connection rather than the one created with the client's initial GET request. The POST request contains the client identifier so that the server can associate the log with the client.

   Note When the pipelined mode of the protocol is used, the client sends this POST request over the same TCP connection that was created with a client's initial GET request. The response to the POST request is not returned until the previous GET request is terminated by the server.
9. The client starts rendering the stream by sending a GET request.

   Note When the non-pipelined mode of the protocol is used, the client sends a SelectStream request. This request contains the stream information in the stream-switch-entry token on the Pragma header. The client might select other streams than those predictably selected by the server. Predictive stream selection provides a way for the server to predict the streams that the client is going to request so the server can begin streaming those streams immediately. For more information, see section 4.5.2.

   Note When the pipelined mode of the protocol is used, the client sends a PlayNextEntry request with the xPlayNextEntry token on the Pragma header over the same connection to the server. This request also contains the stream selection for the next entry in the stream-switch-entry token on the Pragma header. The xPlayNextEntry token indicates that the client can begin streaming the data for the next entry.
10. The server terminates the Play request for the previous entry in the playlist.
11. The server sends the response for the Log request.
12. The server sends $H (Header) and $D (Data) packets for the next entry in the playlist.
13. The server continues looping playlist element data packets (step 5) until the end of the playlist is reached. The end is indicated by the server sending a $E (End-of-Stream Notification) packet with the **Reason** field set to S_OK (0x00000000).

The following illustration shows the sequence described above:

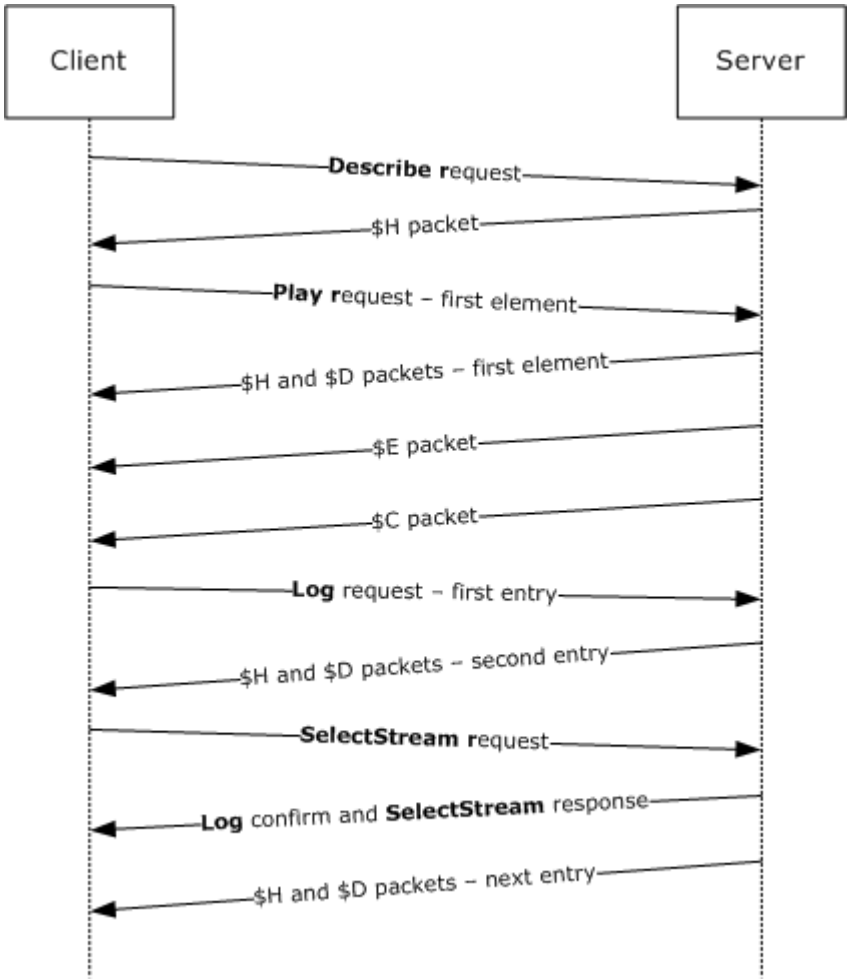


Figure 5: Sequencing during normal playlist streaming

4.5.1 Seeking and Skipping in Server-Side Playlists

The following sequence occurs between a client and server during seeking or skipping in server-side playlists. This sequence applies to both the pipelined mode and non-pipelined mode of the protocol.

1. The client stops streaming the current entry.

   Note When the non-pipelined mode of the protocol is used, the client stops streaming by terminating the TCP connection.

   Note When the pipelined mode of the protocol is used, the client sends the Stop request over the same TCP connection that was created with the client's initial GET request. The response for this POST request is not sent until the server terminates the previous GET request.

2. The client sends a Log request for the previous play.

   Note When the non-pipelined mode of the protocol is used, the client sends the Log request over a new TCP connection. This request contains the client identifier in the client-id token on the

Pragma header so that the server can associate the log with the client. The response for the POST request is returned immediately.

Note When the pipelined mode of the protocol is used, the client sends the Log request over the same TCP connection that was created with the client's initial GET request. The response to the POST request is not returned until the previous GET request is terminated by the server.

3. The client sends a Play request to the server. This GET request carries the seeking or skipping information. The playlist-seek-id token on the Pragma header indicates the current entry being rendered by the client. The pl-offset token on the Pragma header contains the information about whether the client requested to skip to the previous or the next entry. The following are possible offsets:

   - pl-offset=1 indicates that the client should skip to the next entry. The server sends a GET response and then generates the playlist change signal and pushes the $H (Header) packet and $M (Metadata) packet for the next header. From this point forward, the sequence follows a playlist change switch. For more information, see section 4.5.
   - pl-offset=1 indicates that the client should skip to the previous entry. See pl-offset=1 above for detailed server information.
   - pl-offset=0 indicates that the client should seek within the current entry. If the pl-offset=0 token is specified and the server is not predicting streams, then the server sends a GET response and starts streaming the data for the current entry. If the pl-offset=0 token is specified and the server predicts the stream that the client is going to request, then the server would send the GET response and immediately send a $C (Stream Change Notification) packet. The server then follows with $H (Header) and $M (Metadata) packets for the requested playlist entry. From that point on, the sequence of steps would be exactly the same as for a playlist change switch. For more information about predictive streaming, see section 4.5.2.

The following example shows a Stop request for the pipelined mode of the protocol:

```
POST /simple.wsx?WMCache=0 HTTP/1.1
Accept: */*
User-Agent: NSPlayer/9.0.0.2833
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: client-id=2754698341
Pragma: xStopStrm=1
Content-Length: 0
```

The following example shows a skip to next entry:

```
GET /simple.wsx?WMCache=0 HTTP/1.1
Accept: */*
User-Agent: NSPlayer/9.0.0.2833
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: no-cache, pl-offset=1,rate=1.000,stream-time=0,
        stream-offset=4294967295:4294967295,packet-num=4294967295,
        max-duration=0
Pragma: xPlayStrm=1
Pragma: client-id=3342451229
```

```
Pragma: LinkBW=2147483647, AccelBW=2147483647, AccelDuration=10000
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
           com.microsoft.wm.predstrm
Pragma: playlist-seek-id=115
Pragma: stream-switch-count=4
Pragma: stream-switch-entry=ffff:1:0 ffff:2:2 ffff:4:2 ffff:5:0
Accept-Language: en-us, *;q=0.1
```

4.5.2 Server-Side Playlist Streaming with Predictive Stream Selection

The following sequence occurs between a client and server during a server-side playlist switch using either the pipelined mode or the non-pipelined mode of the protocol.

For more information about streaming server-side playlist without predictive streaming, see section 4.5.

1. The client sends a Describe request to retrieve the ASF header.
2. The server responds with 'Supported: com.microsoft.wm.predstrm'.
3. The client sends a Play request for the file, selecting one or more streams with the Supported header.
4. The server responds with a $H (Header) packet and $D (Data) packets of the first playlist entry.
5. The server sends a playlist change signal to the client by sending a $C (Stream Change Notification) packet.
6. The client sends a Log request for the previous entry in the playlist.

   Note When the non-pipelined mode of the protocol is used, the client sends the Log request over a new TCP connection. This request contains the client identifier in the client-id token on the Pragma header so that the server can associate the log with the client.

   Note When the pipelined mode of the protocol is used, the client sends the Log request over the same TCP connection that was created with the client's initial GET request. The response to the POST request is not returned until the previous GET request is terminated by the server.
7. The server predicts the stream selection for the next playlist entry and immediately starts sending $H (Header) and $D (Data) packets for the next entry.

   The client starts receiving the data for the next entry while it is rendering the data for the previous entry. By the time the client finishes rendering, it has accumulated enough data for the next entry to make a seamless switch.
8. If the client wants to override the streams selected by the server, the client sends a GET request to start streaming a new entry.

   Note When the non-pipelined mode of the protocol is used, the client sends a SelectStream request. This request contains the stream-switch-entry token on a Pragma header that specifies the next entry for which the server must send data packets.

   Note When the pipelined mode of the protocol is used, the client sends a PlayNextEntry request with the xPlayNextEntry token on the Pragma header over the same connection to the server. This request also contains the stream-switch-entry token on the Pragma header for the next

entry, which would override the server prediction of streams. The xPlayNextEntry token indicates that the client can continue streaming the data for the next entry. If the client is lagging behind the server, it can send a client-lag token on a Pragma header with a value specified in milliseconds. Upon receipt, the server would pause the predictive stream selection for that duration.

9. The server terminates the Play request for the previous entry in the playlist, as described in step 3.

10. The server sends the response for the Log request, as described in step 6.

11. The server sends the response for the GET request for the new entry, as described in step 8.

12. The server sends $H (Header) and $D (Data) packets for the next entry.

The following example shows a Log request:

```
POST /simple.wsx?WMCache=0 HTTP/1.1
Accept: */*
User-Agent: NSPlayer/9.0.0.2833
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: client-id=2380927133
Content-Length: 3012
Content-Type: application/x-wms-LogStats;charset=UTF-8
```

The following example shows a PlayNextEntry request:

```
GET /simple.wsx?WMCache=0 HTTP/1.1
Accept: */*
User-Agent: NSPlayer/9.0.0.2833
Host: SampleServer
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: client-id=2380927133
Pragma: xPlayNextEntry=1
Pragma: no-cache,rate=1.000
Pragma: client-lag=0
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
           com.microsoft.wm.predstrm
Accept-Language: en-us, *;q=0.1
Pragma: stream-switch-count=4
Pragma: stream-switch-entry=ffff:1:0 ffff:2:2 ffff:4:2 ffff:5:0
```

4.6 Single File Streaming

The following sequence occurs between a client and server when streaming a single file. The sequencing applies to both the pipelined mode and the non-pipelined mode of the protocol.

1. The client sends a Describe request to retrieve the ASF header.

2. The server responds with a $H (Header) packet.

3. The client sends a Play request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets.
5. After all $D (Data) packets have been sent to the client, the server sends a $E (End-of-Stream Notification) packet to indicate that the end of the ASF file has been reached.
6. The client sends a Log request to the server.
7. The server responds with a confirmation that it received the log.

The following illustration shows the sequence described above:

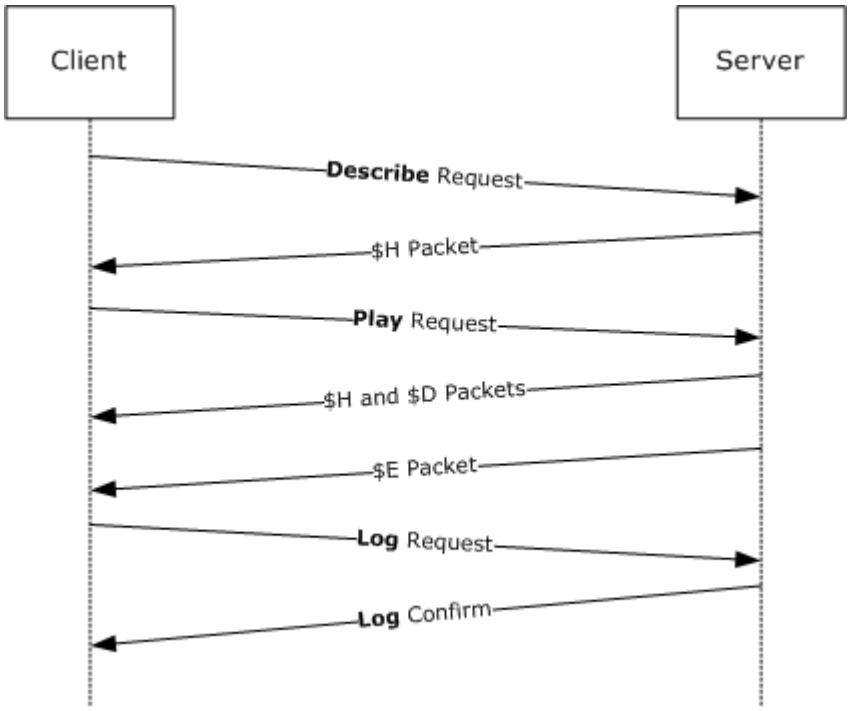


Figure 6: Sequencing during single-file streaming

4.7 Streaming and Stopping Playback by Using Non-Pipelined Mode

The sequence described in this topic occurs between a client and server while streaming a file, pausing playback, and then resuming playback using the non-pipelined mode of the protocol.

For more information about the packet types, see section 2.2.2.3.

1. The client sends a Describe request to retrieve the ASF header.
2. The server responds with a $H (Header) packet.
3. The client sends a Play request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets. The number and size of the $D (Data) packets are not necessarily known at the time of the Play request, so it is not possible for the server to send a Content-Length header. The server transmits $D (Data) packets for the stream until the TCP connection is closed.

5. The client stops streaming by terminating the TCP connection.

6. The client sends a KeepAlive request to the server.

7. The client sends a Log request to the server.

8. The server responds with a confirmation that it received the log.

9. The client sends a Play request for the same file, specifying the point or offset from which to resume playback. If an offset is not specified, then the client resumes playback from the beginning of the file.

10. The server responds with a $H (Header) packet and $D (Data) packets.

11. After all data packets have been sent to the client, the server sends a $E (End-of-Stream Notification) packet to indicate that the end of the ASF file has been reached.

12. The client sends a Log request to the server.

13. The server responds with a confirmation that it received the log.

Pause is implemented by the client terminating the existing TCP connection and issuing another Play request after playback has stopped. In the Play request, the client can specify the point or offset from which to resume playback by using the stream-offset token on a Pragma header. The client can use packet-num, stream-time, or the stream-offset tokens on a Pragma header to specify the offset. These headers are used to seek to a specific point within the ASF file that the client was streaming. If the offset is not provided, then the client resumes playback from the beginning of the file. The server responds to the Play request as described in step 4.

The following illustration shows the sequence described above:

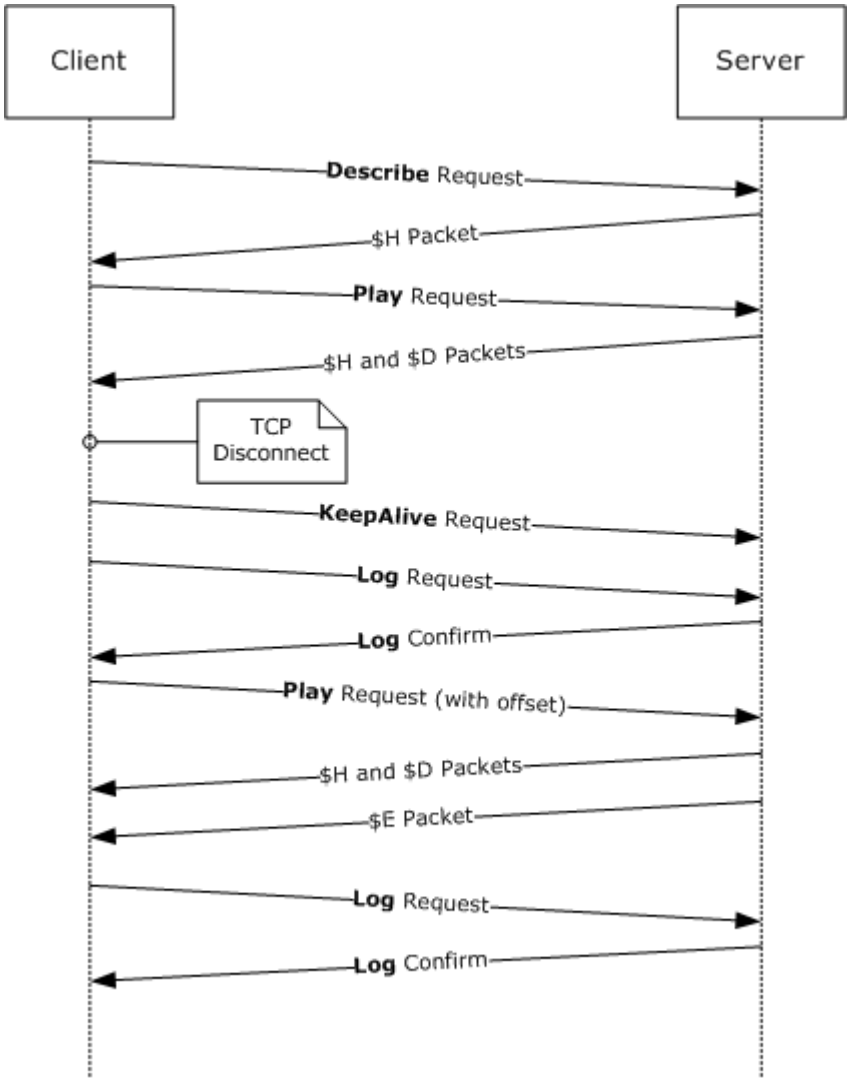


Figure 7: Stream, Pause and Resume packet sequencing

4.8 Streaming and Stopping Playback by Using Pipelined Mode

The sequence described in this topic occurs between a client and server while streaming a file, pausing playback, and then resuming playback using the pipelined mode of the protocol.

1. The client sends a Describe request to retrieve the ASF header.
2. The server responds with a $H (Header) packet.
3. The client sends a Play request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets. HTTP 1.1 Chunked Transfer Coding is used by the server to transmit the packets because the number and size of the $D (Data) packets are not necessarily known at the time of the Play response.

5. The client stops streaming by sending the xStopStrm token on a Pragma header in a Stop request to the server.

6. If a Stop request is received, the server acknowledges this request.

7. The client sends a Log request to the server.

8. The server responds with a confirmation that it received the log.

9. The client sends a Play request for the same file, specifying the point or offset from which to resume playback. If an offset is not specified, then the client resumes playback from the beginning of the file.

10. The server responds with a $H (Header) packet and $D (Data) packets.

11. After all data packets have been sent to the client, the server sends a $E (End-of-Stream Notification) packet to indicate that the end of the media file has been reached.

12. The client sends a Log request to the server.

13. The server responds with a confirmation that it received the log.

Pause is implemented as a Stop request immediately followed by a Play request. After the client has stopped playback, it can choose to resume playing by sending another Play request as described in step 3. In the GET request, the client can specify the point or offset from which to resume playback using the stream-offset token on a Pragma header. The client can use packet-num, stream-time, or the stream-offset tokens on Pragma headers to specify the offset. These headers are used to seek to a specific point within the ASF file that the client was streaming. If the offset is not provided, then the client resumes playback from the beginning of the file. The server responds to the Play request as described in step 4.

The following illustration shows the sequence described above:

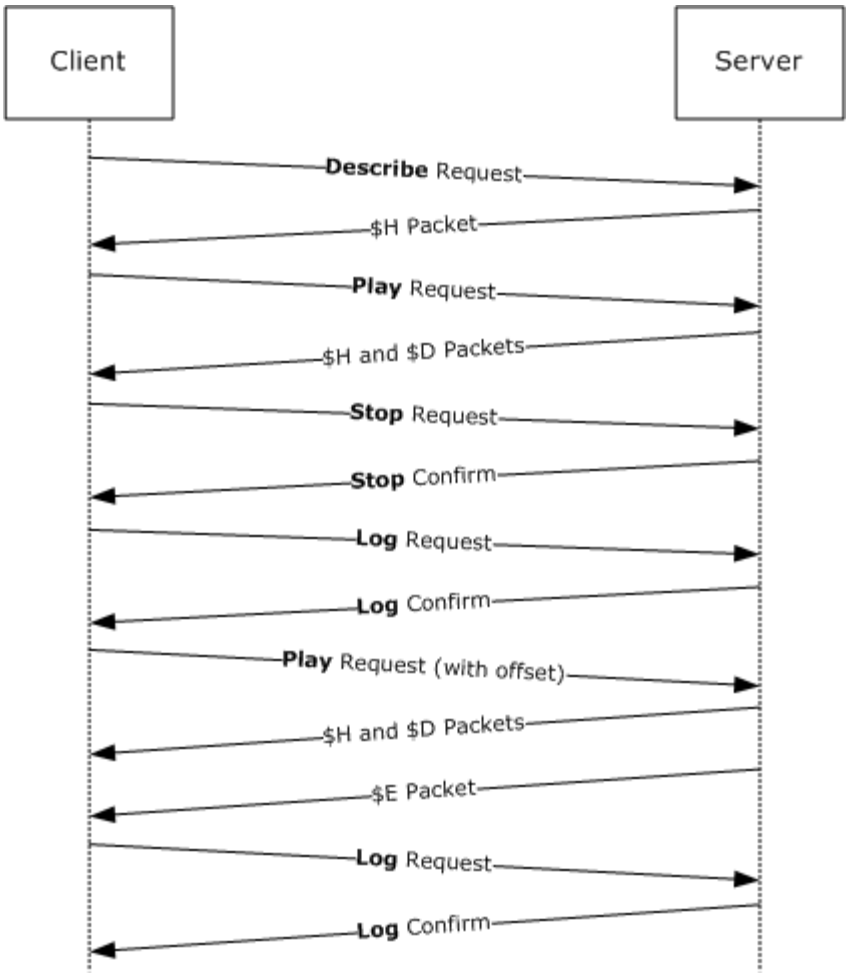


Figure 8: Stream, stop, and resume packet sequencing

To prevent an idle connection from being disconnected, the client sends periodic KeepAlive requests to the server.

4.9 Streaming, Stopping, and Striding Playback

The following sequence occurs between a client and server when streaming a file and then stopping it, followed by **striding** and stopping the file again by using either the pipelined or the non-pipelined mode of the protocol.

1. The client sends a Describe request to retrieve the ASF header.
2. The server responds with a $H (Header) packet.
3. The client sends a Play request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets.
5. The client stops streaming.

Note When the non-pipelined mode of the protocol is used, the client stops streaming the file by terminating the TCP connection.

Note When the pipelined mode of the protocol is used, the client stops streaming by sending the xStopStrm token on a Pragma header in a Stop request to the server.

6. When the pipelined mode of the protocol is used, the server acknowledges if a Stop request is received.
7. The client sends a Log request to the server.
8. The server responds with a confirmation that it received the log.
9. The client sends a Play request for the same file at any rate, fast-forward or rewind, selecting one or more streams. The client can specify the rate using the rate token on the Pragma header. If this header is not specified, the server assumes the rate is 1.0.

   Note Playback rate for a stream can be changed only if the stream supports striding. If striding is supported, the server reports it in the features token on the Pragma header. If $M (Metadata) is present, it overrides the features token on a Pragma header.

10. The server responds with a $H (Header) packet and $D (Data) packets.

The following illustration shows the sequence described above:

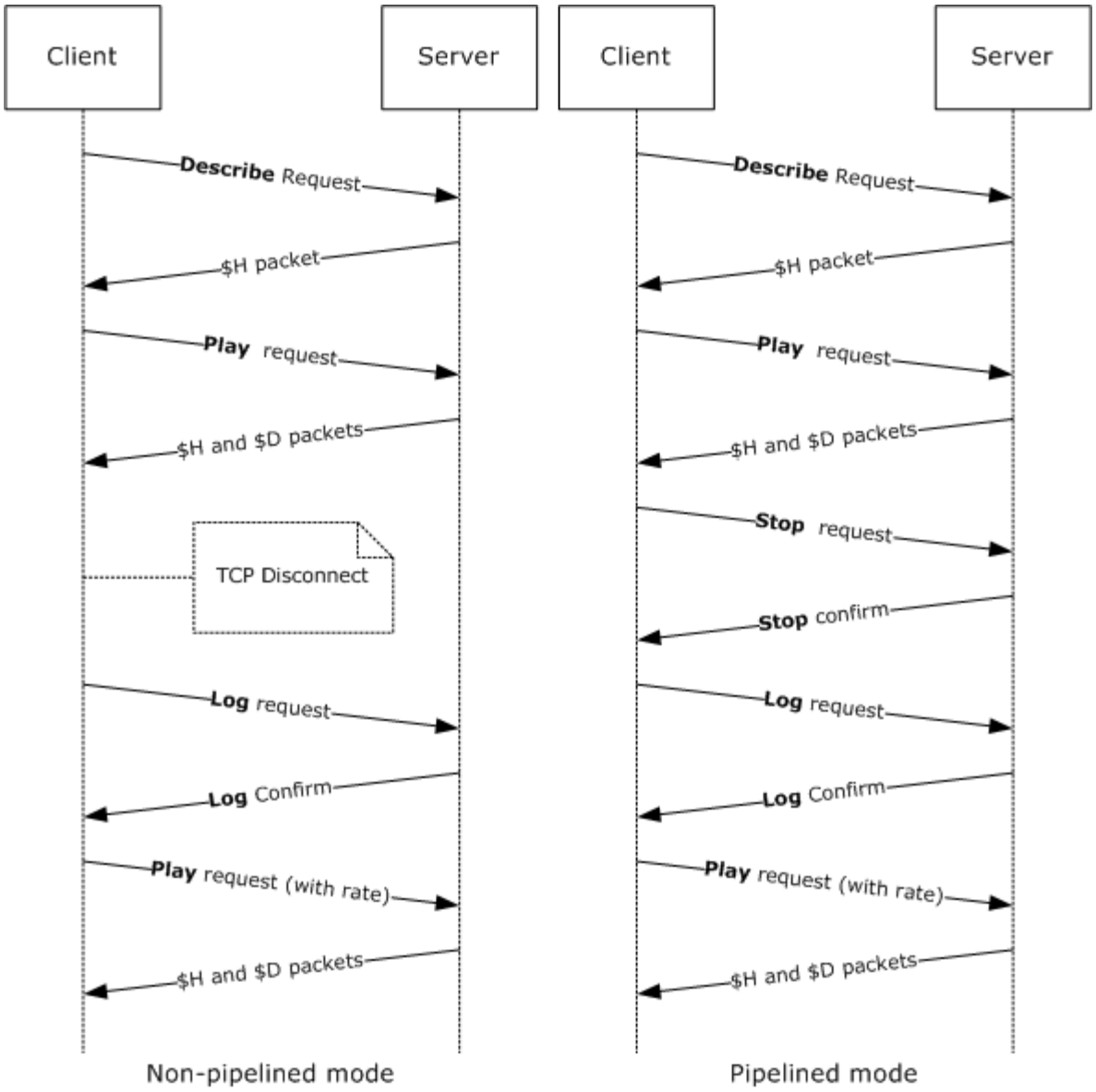


Figure 9: Stream, stop, stride, and stop packet sequencing

4.10 Stream Selection

The SelectStream request is sent by the client to notify the server which streams it requests. If a file contains only one stream, the SelectStream request will request that the server send the default streams for that file. A SelectStream request will always accompany a Play request to denote which audio and/or video streams the client wants to play based on preferences and bandwidth considerations. As with Log and KeepAlive requests, SelectStream request can be sent multiple times during a session, and they are always sent as a POST request on a separate TCP connection.

The following scenarios are examples of when a SelectStream request would be sent by the client:

- When streaming a server-side playlist, the client will send a SelectStream request between each playlist entry.
- When streaming a multi-language Windows Media Audio (.wma) file, the client will send a SelectStream request whenever the client changes from one language to another.
- When streaming a multiple bit rate ASF file, the client will send a SelectStream request whenever it experiences a prolonged change in bandwidth conditions.

The sequencing applies to both the pipelined and the non-pipelined mode of the protocol.

1. The client sends a Describe request to retrieve the ASF header.
2. The server responds with a $H (Header) packet.
3. The client sends a Play request for the file, selecting one or more streams.
4. The server responds with a $H (Header) packet and $D (Data) packets.
5. To change streams, the client passes the stream selection information using a SelectStream request with the stream-switch-entry token on a Pragma header.
6. The server responds by acknowledging the client request and switching to the requested stream.

   Note To avoid disruptions in a video stream, the server will switch the streams at a key frame.
7. The client detects the stream change in the data stream and renders the new stream.

   Note The client will not send a message to the server as a result of this stream change.

The following illustration shows the sequence described above:

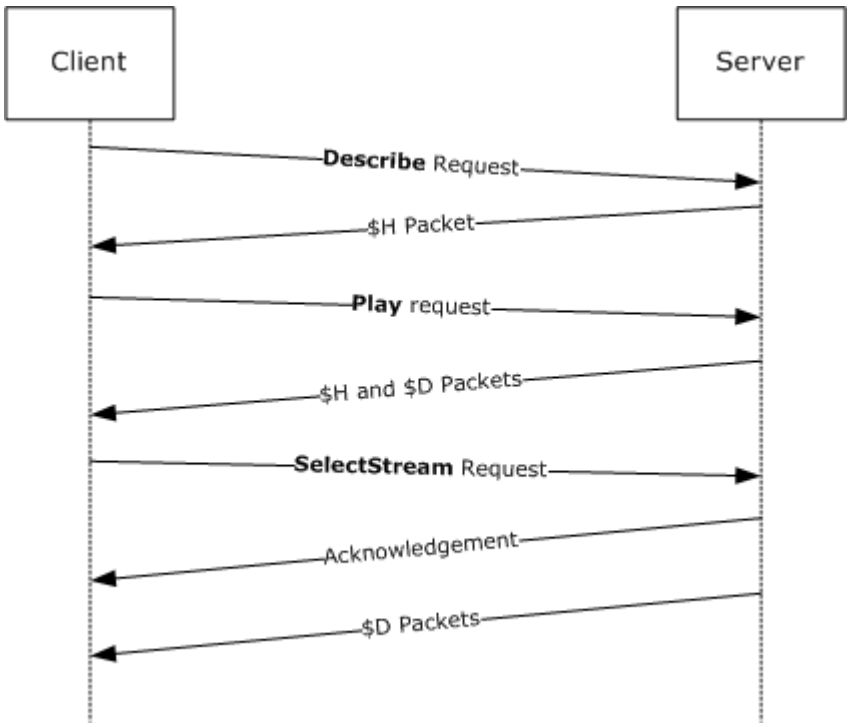


Figure 10: Using the SelectStream Request packet sequencing

4.11 Windows Media Encoder to Windows Media Server Pull Distribution

Pull Distribution is a method by which streaming content is transmitted from a source, such as Windows Media Encoder or Windows Media Services server, to a requesting server using the HTTP protocol. The connection to the encoder and transmission of the content is initiated and managed by the requesting server. When using pull distribution, the server acts as a logical client. Similarly, an encoder behaves as a logical server.

The following sequence occurs between a server and encoder when streaming a single file:

1. The server sends a Describe request to retrieve the ASF header.
2. The encoder responds with a $H (Header) packet.
3. The server sends a Play request for the file, selecting one or more streams.
4. The encoder responds with a $H (Header) packet and $D (Data) packets.
5. After all $D (Data) packets have been sent to the server, the encoder sends a $E (End-of-Stream Notification) packet to indicate that the end of the ASF file has been reached.
6. The server sends a Log request to the encoder.
7. The encoder responds with a confirmation that it received the log.

The following illustration shows the sequence described above:

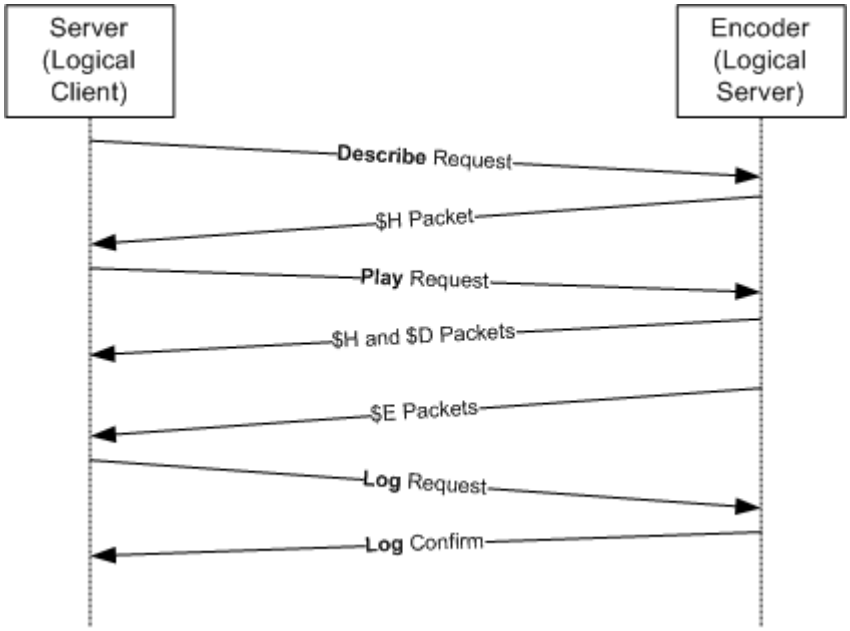


Figure 11: Windows Media Encoder to Windows Media server Pull Distribution

The following example shows a typical Describe request in a Pull:

```
Distribution scenario:GET / HTTP/1.1
```

```
Accept: */*
User-Agent: NSServer/9.0.0.3171
Host: MyServer
Max-Forwards: 9
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: no-cache,rate=1.000,stream-time=0,stream-offset=0:0,
        packet-num=4294967295,max-duration=0
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
           com.microsoft.wm.predstrm
Pragma: xClientGUID={00000000-0000-0000-0000-000000000000}
Accept-Language: *
```

The following example shows a possible response from the encoder:

```
HTTP/1.0 200 OK
Server: Rex/9.0.0.2837
Cache-Control: no-cache
Pragma: no-cache
Pragma: client-id=1543854732
Pragma: features="broadcast,playlist"
Content-Type: application/vnd.ms.wms-hdr.asfv1
Content-Length: 2680
Connection: Keep-Alive

<body of the response containing the headers ($H packets)>
```

The server sends a Play request for one or more streams, as shown in the following example:

```
GET / HTTP/1.0
Accept: */*
User-Agent: NSServer/9.0.0.3171
Host: SampleServer
Max-Forwards: 9
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Pragma: no-cache,rate=1.000,stream-time=0,
        stream-offset=4294967295:4294967295,
        packet-num=4294967295,max-duration=0
Pragma: xPlayStrm=1
Pragma: client-id=1543854732
Pragma: LinkBW=2147483647, BurstBW=3500000, BurstDuration=5000
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
           com.microsoft.wm.predstrm
Pragma: xClientGUID={00000000-0000-0000-0000-000000000000}
Pragma: stream-switch-count=2
Pragma: stream-switch-entry=ffff:1:0 ffff:2:0
Accept-Language: *
```

The encoder responds with a "200 OK" message followed by the $H (Header) and $D (Data) packets for the selected streams.

If the headers change during the course of streaming, the encoder uses the $C (Stream Change Notification) packet followed by the new header data and then the ASF data. When the encoder has finished sending the data, it replies with the $E (End-of-Stream Notification) packet.

Streams pulled from an encoder are considered broadcast content – "trick modes" such as fast-forward, rewind, seek, or pause are not available.

For more information about the Pragma headers in these examples, see section 2.2.1.4.

4.12 Windows Media Services HTTP Proxy Server Interaction

A server that is configured to operate as a proxy server provides the service of routing client requests to one or more origin servers that publish the streaming media content. In this case, the proxy server behaves as a client to the origin server. A proxy server might support caching of content. When the client requests content from the a caching proxy server, it can either transmit the content from its local cache or obtain the content from the origin server and then transmit it to the requesting client.

A Cache-Control header contains directives about the content that indicates to the proxy server how it must handle the content. For example, the x-wms-stream-type directive is used to determine whether the requested content is broadcast or on-demand. The header may include one or more directives and can be passed from the origin server to client through the intermediate proxy server. When the proxy server is caching content, the Cache-Control header MUST be saved. When streaming this content to the requesting client, the proxy server MUST add the previously saved Cache-Control header to the relevant response messages. For more information about the Cache-Control header and the directives, see section 2.2.1.1.

When a client requests on-demand content from a caching proxy server, the proxy server SHOULD first check if the content exists locally and if the content is valid. If both conditions are true, then the proxy server MAY transmit the content from its local cache to the client. If the content has expired, the proxy server SHOULD establish a connection to the origin server to determine if the cached copy of the content is still valid. If the proxy server is able to determine that the cached content is still valid, then the proxy server is allowed to transmit the content to the client. If the cached copy of the content is invalid and caching of the content is allowed, then the proxy server might replace its cached copy by downloading the content from the origin server into the cache. The proxy server would then be able to transmit the content to the requesting client.

For broadcast content, a proxy server might provide the capability to proxy live content from an origin server through another server. When a client requests live content, the proxy server should check whether it is already proxying the content. If so, it could split the streams and delivers the content to all of the requesting clients. If the proxy server is not proxying the content, it could establish a connection to the origin server and forward the content from the origin server to the client. For broadcast content, it is conceivable that only one TCP connection would exist between the proxy server and the origin server.

A proxy server, whether acting as a server or as a client, is largely identical in state to an origin server (as illustrated in sections 4.1 and 4.2.) That is, when streaming to a client, the proxy server is acting as a regular (origin) server. When acting as a logical client, the proxy server is forwarding requests to the origin server. As indicated in the below diagram, much of the decision matrix described above occurs as a result of the Describe request that causes the initial transition into the "Idle" state. One additional state - the "Waiting for GetContentInfo response" state - becomes available with the introduction of a caching proxy server. This state is only applicable if the caching proxy server is acting as a client to the origin server and only if the content on the cache has expired. The caching proxy server will remain in the "Waiting for GetContentInfo response" state until it receives the GetContentInfo response from the origin server. The response to the GetContentInfo request will determine whether the session is streamed from the cache or from the

origin server. In either case, both the origin server and the proxy server transition back to the "Idle" state.

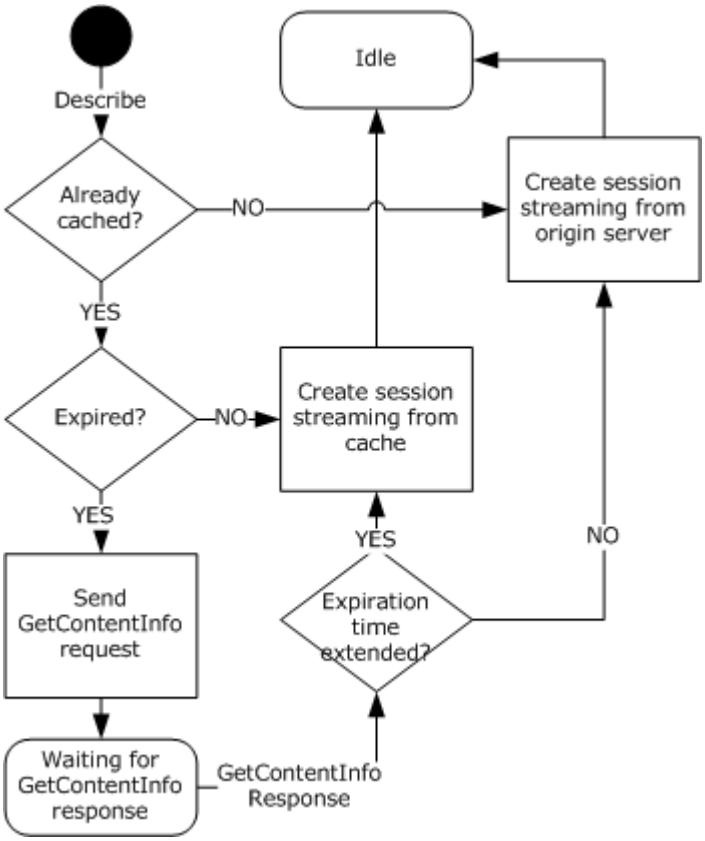


Figure 12: Caching Proxy Server States

Other request types have a negligible impact on state of a proxy server. As indicated in the flow diagram below, if a proxy server receives a rendering log from a client, it should forward it to the origin server regardless of the then current state of the proxy server. For example, if the proxy server is streaming content to a client and receives a rendering log, it should immediately transmit the log to the origin server. The proxy server does not have to wait for a response from the origin server.

On the contrary, if a proxy server receives a streaming log or a "legacy" log from a client, it should not forward it to the origin server. In the case of a "legacy" log, a proxy server implementation ought to synthesize a rendering log from the "legacy" log and transmit the synthesized rendering log to the origin server (assuming the origin server has indicated that it will accept rendering logs.)

Regardless of the type of log message received by the proxy server, there is no state change caused by the transmittal of a log from the proxy server to the origin server. More details on the different Windows Media Log types are specified in [MS-WMLOG].

In the case that the request received by the proxy server is not log-related, the proxy server will attempt to service the request locally. If it cannot, the request will be forwarded to the origin server. The proxy server will wait until it receives a response from the origin server before sending a response to the client that originated the request. It is important to note that this interaction does not trigger a state transition - the proxy server will remain in its then current state.

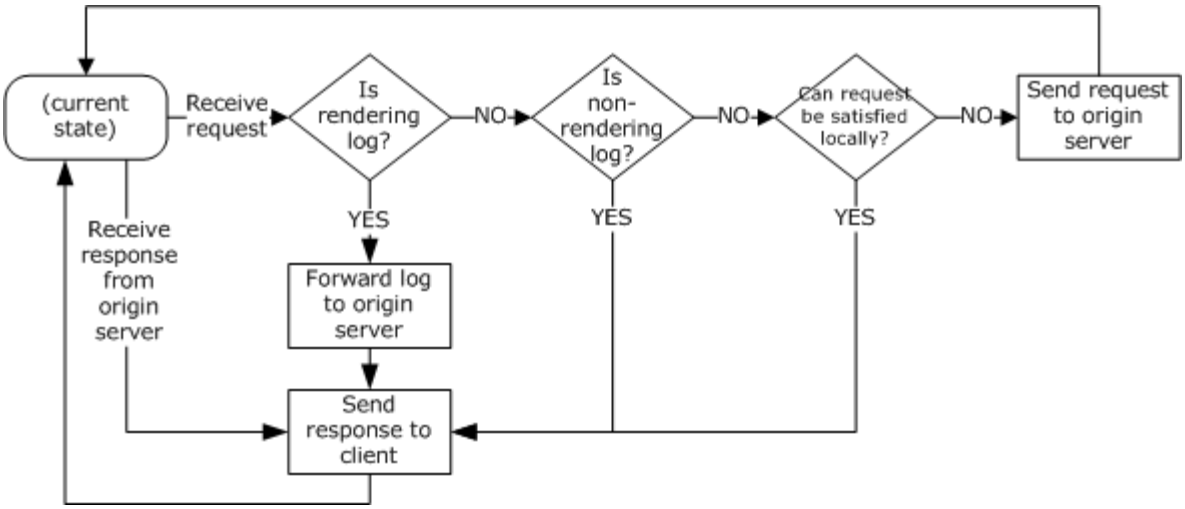


Figure 13: Caching Proxy Server State Flow Diagram

4.12.1 Sequencing

The general message sequence consists of the following steps:

1. The client sends a Describe request.
2. The cache/proxy server responds with a $H packet that contains the content description.
3. The client sends a Play request, selecting one or more streams.
4. The cache/proxy server determines whether the requested content is broadcast or on-demand (as indicated within the x-wms-stream-type directive on the Cache-Control header).
5. Depending on the type of the stream, the cache/proxy delivers the content to the client. These steps are described in sections 4.12.1.1 and 4.12.1.2.

4.12.1.1 On-Demand Content Delivery

The On-Demand Content Delivery consists of the following steps:

1. The cache/proxy server checks whether the requested stream is in its local cache.

   If the content is not in the cache, then the cache/proxy opens a connection with the origin server. These steps are described in section 4.12.1.3.

2. The cache/proxy server checks whether the content in the local cache is valid. If the content is not valid, the cache/proxy server revalidates the content.

   If the content is still not valid, then the cache/proxy opens a connection with the origin server. These steps are described in section 4.12.1.3.

3. The cache/proxy server sends the Cache-Control header to the client.
4. The cache/proxy server sends $H and $D packets for the requested stream from its local cache to the client.

5. The cache/proxy server communicates any subscribed-to events (as indicated within the x-wms-event-subscription directive on the Cache-Control header) to the origin server.

The following illustration shows the sequencing that occurs when the client requests a file from a media server that is configured as a cache/proxy server.

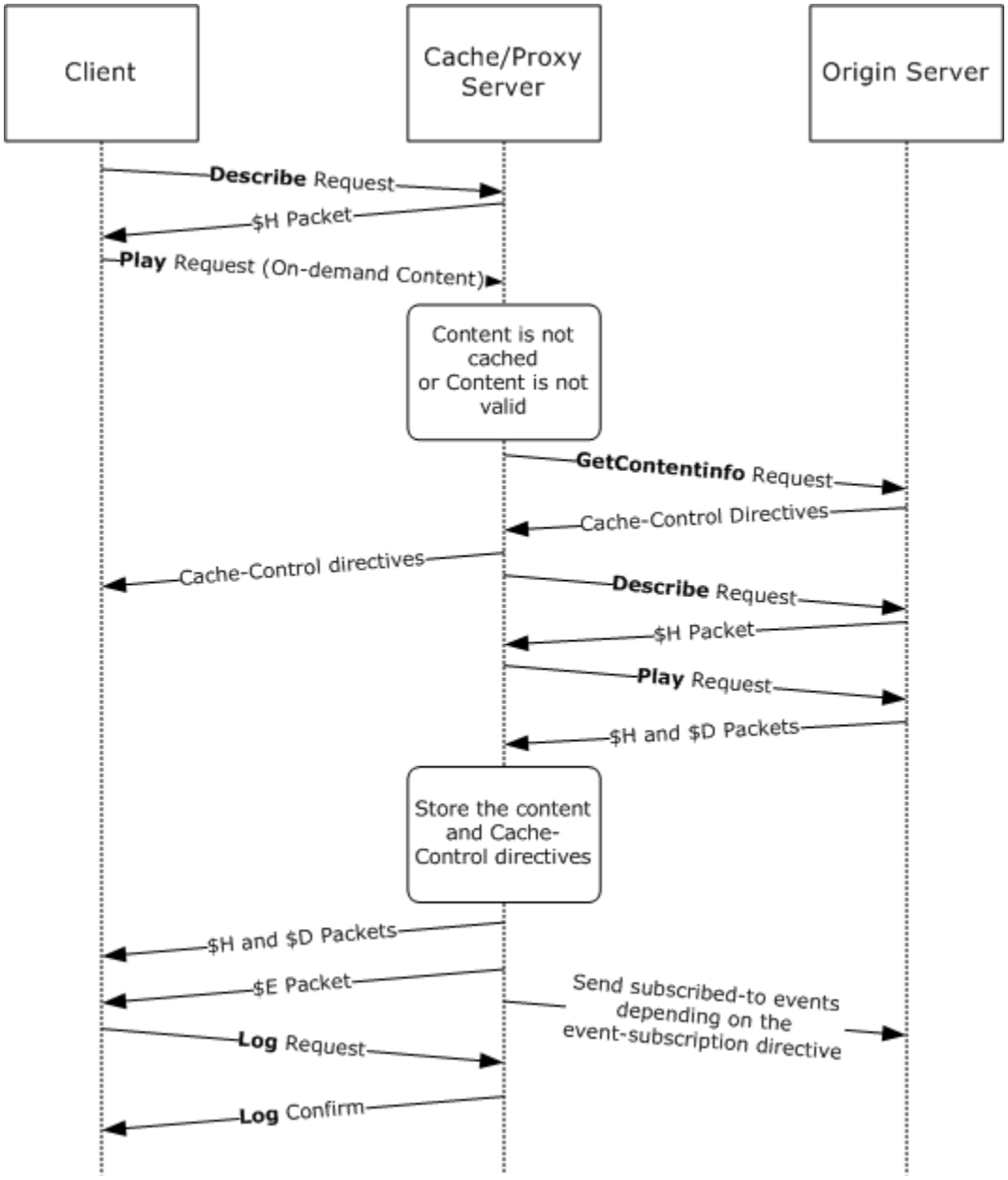


Figure 14: On-demand content delivery

4.12.1.2 Broadcast Content Delivery

The Broadcast Content Delivery consists of the following steps:

1. The cache/proxy server checks whether the stream can be split, as determined by the x-wms-proxy-split directive on the Cache-Control header.

   If the content cannot be split, then the cache/proxy opens a connection with the origin server. These steps are described in section 4.12.1.3.

2. The cache/proxy server checks whether it is currently receiving the content.

   If the content is not being received, then the cache/proxy opens a connection with the origin server. These steps are described in section 4.12.1.3.

3. The cache/proxy server checks whether the content is valid. If the content is not valid, the cache/proxy server revalidates the content.

   If the content is still not valid after revalidation, then the cache/proxy opens a connection with the origin server. These steps are described in section 4.12.1.3.

4. The cache/proxy server splits the content stream and sends the $H and $D packets, which it receives from the origin server, to the client.

5. The cache/proxy server communicates any subscribed-to events (as indicated within the x-wms-event-subscription directive on the Cache-Control header) to the origin server.

The following illustration shows the sequencing that occurs when the client requests broadcast content from a media server that is configured as a cache/proxy server.

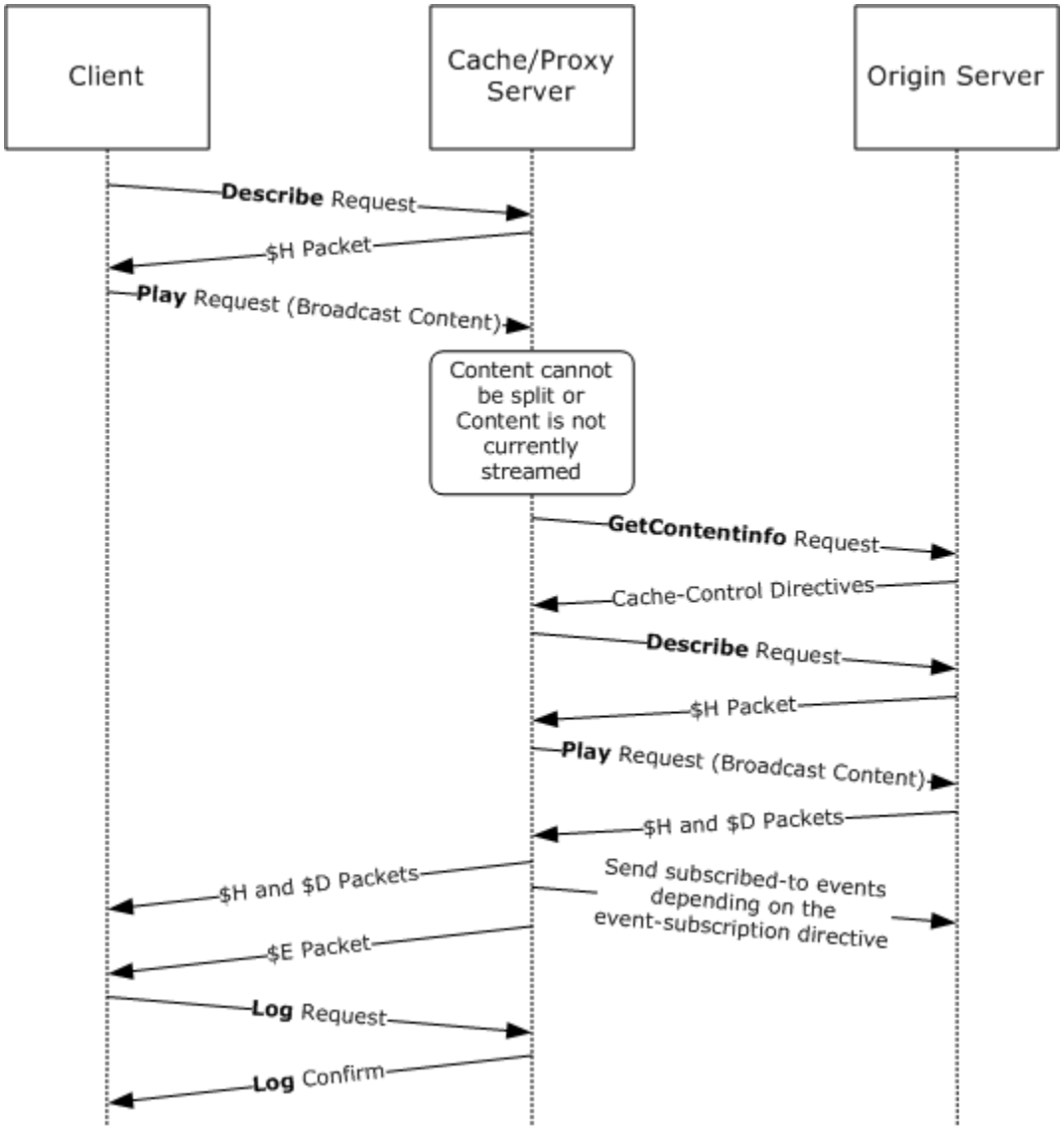


Figure 15: Broadcast content delivery

4.12.1.3 Cache/Proxy Server and Origin Server Communication

The following steps demonstrate the Cache/Proxy Server and Origin Server Communication:

1. The cache/proxy server requests content information from the origin server by sending a GetContentInfo request.
2. The origin server responds with the Cache-Control header for the content.
3. Optional. Additional handshaking can occur depending on the implementation of the origin server.
4. The cache/proxy server determines whether to cache the content that the origin server is about to stream. If the content is to be cached, the cache/proxy server stores the Cache-Control header information and the content in the local cache.

5. If the content is not fully cached in the cache/proxy server's local cache, or if the content will not be cached, the cache/proxy server sends a Describe request followed by a Play request to the origin server.

6. The cache/proxy sends the $H and $D packets for the requested content to the client. Depending on whether the content is already fully cached, the source is either the cache/proxy server's local cache or the origin server.

7. The cache/proxy server communicates any subscribed-to events (as indicated within the x-wms-event-subscription directive on the Cache-Control header) to the origin server.

The following shows a sample GetContentInfo request and the corresponding response.

Client to Server:

```
POST /welcome2.asf?WMCache=0 HTTP/1.1
User-Agent: WMCacheProxy/9.0.0.3177
Via: HTTP/1.1 WMCacheProxy (WMCacheProxy/9.0.0.3177)
Max-Forwards: 9
Accept-Charset: UTF-8, *;q=0.1
Pragma: xClientGUID={00000000-0000-0000-0000-000000000000}
X-Accept-Authentication: Negotiate, MS-NLMP, Digest
Content-Type: application/x-wms-getcontentinfo
Host: myhost:87
Content-Length: 1
Connection: Keep-Alive
```

Server to Client:

```
HTTP/1.1 200 OK
Server: Cougar/9.00.00.3178
Date: Wed, 31 Jul 2002 00:13:03 GMT
Pragma: no-cache, xResetStrm=1, timeout=60000
Cache-Control: no-cache, x-wms-content-size=2139795, max-age=86399,
               user-public, must-revalidate, proxy-public,
               proxy-revalidate
Last-Modified: Fri, 30 Jun 2000 09:37:24 GMT
Etag: "2139795"
Supported: com.microsoft.wm.srvppair, com.microsoft.wm.sswitch,
           com.microsoft.wm.predstrm, com.microsoft.wm.fastcache
Content-Length: 0
```

5 Security

The following sections specify security considerations for implementers of the Windows Media HTTP Streaming Protocol.

5.1 Security Considerations for Implementers

The Windows Media HTTP Streaming Protocol is vulnerable to a session hijacking attack in which the attacker guesses the value of the client-id (section 2.2.1.4.5) token on the Pragma header and the TCP port number used by the client. The attacker makes the server believe that the TCP connection to the client has been lost. Then the attacker establishes its own TCP connection to the server and sends a request with the victim's client-id value. To mitigate the attack, server implementations should use a good random number generator when creating client-id values. Also, if HTTP Access Authentication is used, the server should authenticate access at least once on each new URL or TCP connection, or, preferably, on each Play request.

5.2 Index of Security Parameters

Security Parameter	Section
HTTP Access Authentication	2.1

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Vista
- Windows Server 2003
- Windows XP
- Windows 2000
- Windows NT

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.1: Windows Media Player 6, Windows Media Format 7.0 SDK, Windows Media Format 7.1 SDK, and Windows Media Player for Windows XP support Basic authentication (as specified in [RFC2617] and [MS-NLMP]). Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista, support MS-NLMP, Digest (as specified in [RFC2617]), and Negotiate (as specified in [RFC4559]) authentication. Basic authentication is only supported when challenged by a proxy server. Windows Media Services on Windows NT Server 4.0 and on Windows 2000 Server supports Basic and [MS-NLMP] authentication. Windows Media Services on Windows Server 2003 supports [MS-NLMP], Digest, and Negotiate authentication. Authentication protocols on the server are disabled by default and may be selectively enabled by the server administrator.

<2> Section 2.2.1.1: This header is not supported by Windows Media Player 6, or by Windows Media Services on Windows NT Server 4.0 and on Windows 2000 Server.

<3> Section 2.2.1.1.9: This directive is only supported by Windows Media Services on Windows Server 2003.

<4> Section 2.2.1.1.10: Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista only support the "remote-log" event.

<5> Section 2.2.1.1.11: This directive is only supported by Windows Media Services on Windows Server 2003.

<6> Section 2.2.1.1.12: This directive is only supported by Windows Media Services on Windows Server 2003.

<7> Section 2.2.1.2: Windows Media Services on Windows NT 4.0 and Windows 2000 Server only sends the "application/octet-stream" content-type. Also, Windows Media Player 6 only sends the "text/plain" content-type in its POST requests.

<8> Section 2.2.1.2.4: This content-type is only supported by Windows Media Services on Windows Server 2003.

<9> Section 2.2.1.2.5: This content-type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<10> Section 2.2.1.2.6: This content-type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<11> Section 2.2.1.2.7: This content-type is only sent by Windows Media Player 6, Windows Media Format 7.0 SDK, Windows Media Format 7.1 SDK, and by Windows Media Player for Windows XP, and by Windows Media Services on Windows NT 4.0 and Windows 2000 Server.

<12> Section 2.2.1.4: When Windows Media Services is acting in the server role, it will typically respond to errors with "400 Bad Request" or a similar HTTP error code. Implementations of the client role on Windows (listed in the introduction to this section) will typically close the connection to the server and display an error message to the user.

<13> Section 2.2.1.4.1: This token is supported by Windows Media Player for Windows XP, Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. The token is also sent by Windows Media Format 7.0 SDK and by Windows Media Format 7.1 SDK. However, those products do not properly process the token in a response, hence the requirement to not send the token to clients with a version number less than 8.

<14> Section 2.2.1.4.2: This token is supported by Windows Media Player for Windows XP, Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. The token is also sent by Windows Media Format 7.0 SDK, and by Windows Media Format 7.1 SDK, but those products do not properly process the token in a response, hence the requirement to not send the token to clients with version number less than 8.

<15> Section 2.2.1.4.3: This token is supported by Windows Media Services on Windows Server 2003.

<16> Section 2.2.1.4.4: This token is supported by Windows Media Services on Windows Server 2003.

<17> Section 2.2.1.4.6: This token is supported by Windows Media Player for Windows XP, Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<18> Section 2.2.1.4.8.2: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<19> Section 2.2.1.4.8.4: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<20> Section 2.2.1.4.8.7: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<21> Section 2.2.1.4.8.8: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<22> Section 2.2.1.4.9: : This token is supported by Windows Media Format 7.0 SDK, Windows Media Format 7.1 SDK, Windows Media Player for Windows XP, Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. When supported, the value of the token is set to the link bandwidth if it has been

configured by the user or the link bandwidth if it was successfully determined through a packet-pair experiment. If the link bandwidth is unknown, the token is not transmitted. Windows Media Services on Windows Server 2003 uses the value of the token in determining which streams should be selected when it switches to the next entry (if any) in a server-side playlist.

<23> Section 2.2.1.4.11: This token is not ignored by Windows Media Services on Windows NT 4.0 and Windows 2000.

<24> Section 2.2.1.4.14: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<25> Section 2.2.1.4.15: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<26> Section 2.2.1.4.16: Section 2.2.1.4.16: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<27> Section 2.2.1.4.17: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<28> Section 2.2.1.4.18: : This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<29> Section 2.2.1.4.19: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<30> Section 2.2.1.4.20: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<31> Section 2.2.1.4.23: The first Play request sent by Windows Media Player 6 MUST specify a value for this token that is 2. For other GET requests sent by Windows Media Player 6, the token MUST be either omitted or have a value that is greater than 2. The first Play request sent by Windows Media Services on Windows NT 4.0 and Windows 2000 MUST specify a value for this token that is either 2 or 3. For other GET requests sent by Windows Media Services on Windows NT 4.0 and Windows 2000, the token MUST be either omitted or have a value that is greater than 3. If Windows Media Services on Windows Server 2003 receives a Play request, and the client is either Windows Media Player 6 or Windows Media Services with a version less than 5.0 (as specified by the client in the User-Agent header), then the server will use this token to determine if the request is the first Play request in the current session. If it is not the first Play request, and the $H packet that will be sent in response to the Play request is different from the most recently transmitted $H packet, then Windows Media Services will follow the $H packet with a $C packet followed by the same $H packet again.

<32> Section 2.2.1.4.24: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. Windows Media Services on Windows Server 2003 may respond to a request that specifies this token with a speed token that specifies a value that is less than what the client specified, but it will never be less than 1 if the client specified a value greater than 1.

<33> Section 2.2.1.4.29: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. Windows Media Services on Windows Server 2003 will disconnect a client that has been idle, but the actual time out used may be greater (but never smaller) than the time out specified by the token.

<34> Section 2.2.1.4.30: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<35> Section 2.2.1.4.31: This token is supported by Windows Media Player 6 and Windows Media Services on Windows NT 4.0 and Windows 2000. Versions of Windows Media Services that support this token do not normally send it, but they can be configured to do so.

<36> Section 2.2.1.4.32: This token is supported by Windows Media Player 6 and Windows Media Services on Windows NT 4.0 and Windows 2000. Versions of Windows Media Services that support this token do not normally send it, but they can be configured to do so.

<37> Section 2.2.1.4.34: This token is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. The client products that support this token will send a request with the token if the connection has been idle for the amount that the server specified in the timeout token. Windows Media Services on Windows Server 2003 will reset its idle timeout timer for the specified session whenever it receives a request with the xKeepAliveInPause token.

<38> Section 2.2.1.4.35: The pipelined mode of the protocol, and hence also this token, is only supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<39> Section 2.2.1.4.38: The pipelined mode of the protocol, and hence also this token, is only supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<40> Section 2.2.1.7: This header is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<41> Section 2.2.1.7.5: The X-StartupProfile (section 2.2.1.12) header is supported by Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003 SP1.

<42> Section 2.2.1.9: This header is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. Windows Media Player 6, Windows Media Format 7.0 SDK, Windows Media Format 7.1 SDK, and Windows Media Player for Windows XP support Basic (as specified in [RFC2617]) and MS-NLMP authentication (as specified in [MS-NLMP]). Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista support MS-NLMP (as specified in [MS-NLMP]), Digest (as specified in [RFC2617]), and Negotiate authentication (as specified in [RFC4559]). Basic authentication is only supported when challenged by a proxy server. Windows Media Services on Windows NT 4.0 and on Windows 2000 Server supports Basic and MS-NLMP authentication. Windows Media Services on Windows Server 2003 supports MS-NLMP, Digest, and Negotiate authentication. Authentication protocols on the server are disabled by default and may be selectively enabled by the server administrator.

<43> Section 2.2.1.10: : This header is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. Windows Media Player 6, Windows Media Format 7.0 SDK, Windows Media Format 7.1

SDK, and Windows Media Player for Windows XP support Basic (as specified in [RFC2617]) and MS-NLMP authentication (as specified in [MS-NLMP]). Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista support Basic, MS-NLMP, Digest (as specified in [RFC2617]), and Negotiate authentication (as specified in [RFC4559]). Windows Media Services on Windows NT 4.0 and on Windows 2000 Server supports Basic and MS-NLMP authentication. Windows Media Services on Windows Server 2003 supports MS-NLMP, Digest, and Negotiate authentication. Authentication protocols on the server are disabled by default and may be selectively enabled by the server administrator.

<44> Section 2.2.1.12: This header is supported by Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<45> Section 2.2.2.2: This request type is supported by Windows Media Services on Windows Server 2003.

<46> Section 2.2.2.3: This request type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<47> Section 2.2.2.5: This request type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<48> Section 2.2.2.7: This request type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<49> Section 2.2.2.9: This request type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<50> Section 2.2.2.10: This request type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<51> Section 2.2.3.1: Implementations on Windows (listed in the introduction to this section) typically close the connection to the server and display an error message to the user if a packet is received with a field set to an invalid value.

<52> Section 2.2.3.1.1: This field is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003. Windows Media Services sets this field to 1 on any packets that are transmitted faster than real time as the result of the client having specified the AccelBW (section 2.2.1.4.1) token on the Pragma header.

<53> Section 2.2.3.6: This packet type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<54> Section 2.2.3.7: This packet type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<55> Section 2.2.3.8: This packet type is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, and by Windows Media Services on Windows Server 2003.

<56> Section 3.1.5.1: The com.microsoft.wm.startupprofile (section 2.2.1.7.5) token is only specified by Windows Media Format 9.5 SDK, and Windows Vista. Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, Windows Vista, only specify the com.microsoft.wm.predstrm (section 2.2.1.7.2) token when the most recent Play (section 2.2.2.6) or PlayNextEntry (section 2.2.2.7) request did not include the speed token with a value other than 1 on a Pragma (section 2.2.1.4) header.

<57> Section 3.1.5.1: Whether the same GUID is used for all sessions or changed between different sessions is determined by how the user has configured Windows Media Player and/or the Windows Media Format SDK.

<58> Section 3.1.5.1: Only Windows Media Services on Windows Server 2003 supports sending a Pragma (section 2.2.1.4) header with the proxy-client-agent (section 2.2.1.4.21) token or a X-Proxy-Client-Verb (section 2.2.1.11) header.

<59> Section 3.1.5.1: The Cookie (section 2.2.1.3) header can be sent by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista.

<60> Section 3.1.5.2: Windows Media Player 6, Windows Media Format 7.0 SDK, Windows Media Format 7.1 SDK, and Windows Media Player for Windows XP, support Basic (as specified in [RFC2617]) and NTLM (as specified in [NTLM]) authentication. Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista, support NTLM, Digest (as specified in [RFC2617]) and Negotiate (as specified in [RFC4559]) authentication. Basic authentication is only supported when challenged by a proxy server. Windows Media Services on Windows NT 4.0 and on WS-Management Windows 2000 Server supports Basic and NTLM authentication. Windows Media Services on Windows Server 2003 supports NTLM, Digest, and Negotiate authentication.

<61> Section 3.1.5.2: Only Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista support caching of content and thus only those product versions will adhere to the directives on the Cache-Control header.

<62> Section 3.1.5.2: Windows Media Player 6 will use the URL specified in the value of the version-info token to download an update but only if the version number specified by the version-info token is greater than the current version number of Windows Media Player.

<63> Section 3.1.5.2: The Set-Cookie (section 2.2.1.6) header is supported by Windows Media Format 9 Series SDK, Windows Media Format 9.5 SDK, and Windows Vista.

<64> Section 3.2.5.1: Windows Media Services will respond to the request with HTTP status code 200 and the message response of the request will contain an ASX file that contains the same URL that the client provided in the request. Because ASX files are understood by Windows Media Player, it is expected that the Web client will invoke Windows Media Player to parse the ASX file.

<65> Section 3.2.5.2: The com.microsoft.wm.startupprofile (section 2.2.1.7.5) token is only specified by Windows Media Services on Windows Server 2003 SP1.

7 Index

S

T

U

V

W

X