[MS-SSTR]:
Smooth Streaming Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

Revision Summary

Date	Revision History	Revision Class	Comments
06/04/2010	0.1	Major	First Release.
07/16/2010	0.1	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	0.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	0.2	Minor	Clarified the meaning of the technical content.
11/19/2010	0.2	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	0.2	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	0.2	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	0.2	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	0.2.1	Editorial	Changed language and formatting in the technical content.
06/17/2011	0.3	Minor	Clarified the meaning of the technical content.
09/23/2011	0.3	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	1.0	Major	Significantly changed the technical content.
03/30/2012	2.0	Major	Significantly changed the technical content.
07/12/2012	2.1	Minor	Clarified the meaning of the technical content.
10/25/2012	3.0	Major	Significantly changed the technical content.
01/31/2013	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	4.0	Major	Significantly changed the technical content.
11/14/2013	5.0	Major	Significantly changed the technical content.

Release: Friday, October 25, 2013

Contents

1 Introduction

The Smooth Streaming Protocol describes the wire format used to deliver (via HTTP) live and on-demand digital **media**, such as audio and video, in the following manners: from an encoder to a web server, from a server to another server, and from a server to an HTTP client. The use of an MPEG-4 ([MPEG4-RA])-based data structure delivery over HTTP allows seamless switching in near-real-time between different quality levels of compressed media content. The result is a constant playback experience for the HTTP client end user, even if network and video rendering conditions change for the client computer or device.

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

1.1 Glossary

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

> **globally unique identifier (GUID)**
> **universally unique identifier (UUID)**

The following terms are specific to this document:

> **bit rate:** A measure of the average bandwidth required to deliver a **track**, in bits per second (bps).

> **composition time:** The time a sample needs to be presented to the client, as defined in [ISO/IEC-14496-12].

> **decode:** To decompress video or audio **samples** for playback.

> **decode time:** The time a sample is required to be decoded on the client, as defined in [ISO/IEC-14496-12].

> **Digital Video Recorder (DVR) content: Live** content not consumed at the **live position**.

> **DVR Window:** The length of time that content is available as DVR Content.

> **encode:** To compress raw video or audio into **samples** in a **media format**.

> **fresh:** A response stored on an HTTP cache proxy that has not expired, as defined in [RFC2616].

> **fragment:** An independently downloadable unit of **media** that comprises one or more **samples**.

> **live:** A **presentation** that is used to deliver an ongoing live event.

> **live position:** The latest content available for viewing in a **live presentation**.

> **HTTP cache proxy:** A proxy that can deliver a stored copy of a response to clients.

> **manifest:** Metadata about the **presentation** that allows a client to make requests for **media**.

> **media:** Compressed audio, video, and text data used by the client to play a **presentation**.

> **media format:** A well-defined format for representing audio or video as a compressed **sample**.

on-demand: A **presentation** that is available in its entirety when playback begins.

packet: A unit of audio **media** that defines natural boundaries for optimizing audio **decoding**.

parent track: A **track** with which one or more **sparse tracks** is associated, and which is used to transmit timing information for the sparse track. Parent stream fragments always contain the time stamp for the last sparse fragment.

presentation: The set of all **streams** and related metadata needed to play a single movie.

request: An HTTP message sent from the client to the server, as defined in [RFC2616].

response: An HTTP message sent from the server to the client, as defined in [RFC2616].

sample: The smallest fundamental unit (such as a frame) in which **media** is stored and processed.

sparse stream: A **stream** that comprises one or more **sparse tracks**.

sparse track: A **track** characterized by **fragments** that occur at irregular time intervals. It can be used to send metadata to clients to support scenarios such as ad-signaling. This contrasts with non-sparse streams (for example, audio, video) in which fragments are sent at regular time intervals. A sparse track is always associated with a non-sparse **parent track** that is used to transmit timing information for the sparse track. Each sparse fragment includes a reference to any sparse track fragments created immediately before it.

stream: A set of **tracks** interchangeable at the client when playing **media**.

track: A time-ordered collection of **samples** of a particular type (such as audio or video).

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

1.2 References

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

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

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[ISO/IEC-14496-12] International Organization for Standardization, "Information technology -- Coding of audio-visual objects -- Part 12: ISO Base Media File Format", ISO/IEC 14496-12:2008, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51533

[ISO/IEC-14496-3] International Organization for Standardization, "Information technology -- Coding of audio-visual objects -- Part 3: Audio", ISO/IEC 14496-3:2009, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53943

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

[MPEG4-RA] The MP4 Registration Authority, "MP4REG", http://www.mp4ra.org

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

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

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

[SMIL2.1] Bulterman, D., Grassel, G., Jansen, J., Koivisto, A., Layaida, N., et al., Eds., "Synchronized Multimedia Integration Language", W3C Recommendation, December, 2005, http://www.w3.org/TR/SMIL2/

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

1.2.2   Informative References

[ISO/IEC-14496-15] International Organization for Standardization, "Information technology -- Coding of audio-visual objects -- Part 15: Advanced Video Coding (AVC) file format", ISO 14496-15, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38573

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

[MSDN-VIH] Microsoft Corporation, "VIDEOINFOHEADER structure", http://msdn.microsoft.com/en-us/library/dd407325(VS.85).aspx

[RFC2326] Schulzrinne, H., Rao, A., and Lanphier, R., "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998, http://www.ietf.org/rfc/rfc2326.txt

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

[RFC5234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, http://www.rfc-editor.org/rfc/rfc5234.txt

[VC-1] Society of Motion Picture and Television Engineers, "VC-1 Compressed Video Bitstream Format and Decoding Process", SMPTE 421M-2006, April 2006, http://standards.smpte.org/content/978-1-61482-555-5/st-421-2006/SEC1.body.pdf+html?sid=dc1cd243-8c31-45a2-87c6-1695c5bc63e5

Note There is a charge to download the specification.

[WFEX] Microsoft Corporation, "Augmented Multiple Channel Audio Data and WAVE Files", March 2007, http://www.microsoft.com/whdc/device/audio/multichaud.mspx

1.3 Overview

The IIS Smooth Streaming Transport Protocol provides a means of delivering media from encoders to servers (in the case of live streaming) and from servers to clients in a way that can be cached by standard HTTP cache proxies in the communication chain. Allowing standard **HTTP cache proxies** to respond to requests on behalf of the server increases the number of clients that can be served by a single server.

The following figure depicts a typical communication pattern for the protocol:
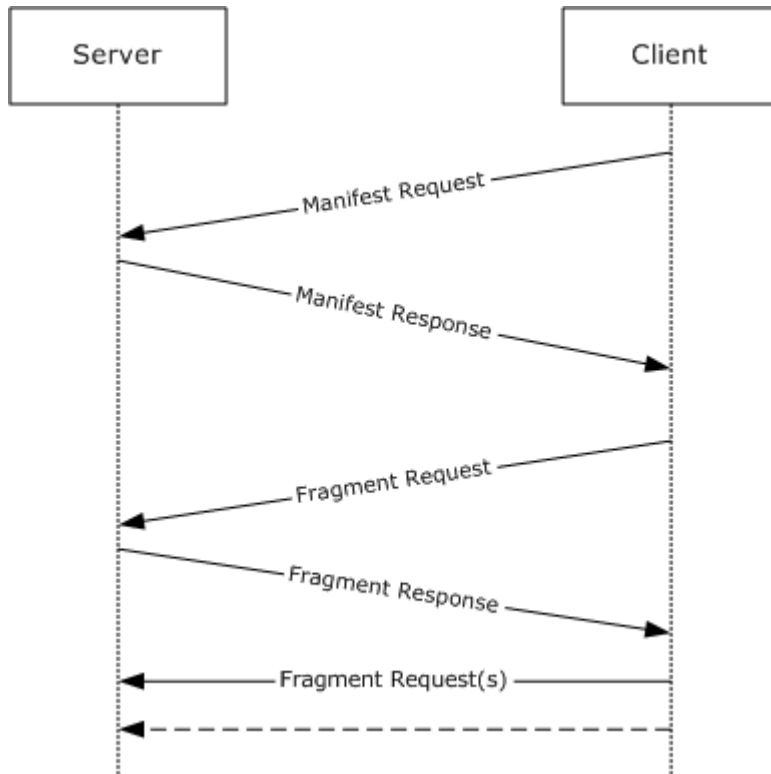
Figure 1: Typical communication sequence for the IIS Smooth Streaming Transport Protocol

The first message in the communication pattern is a Manifest Request, to which the server replies with a Manifest Response. The client then makes one or more Fragment Requests, and the server replies to each with a Fragment Response. Correlation between Requests and Responses is handled by the underlying Hypertext Transport Protocol (HTTP) [RFC2616] layer.

The server role in the protocol is stateless, allowing each request from the client to be potentially handled by a different instance of the server, or by one or more HTTP cache proxies. The following figure depicts the communication pattern for requests for the same **fragment**, indicated as "Fragment Request X", when an HTTP cache proxy is used:
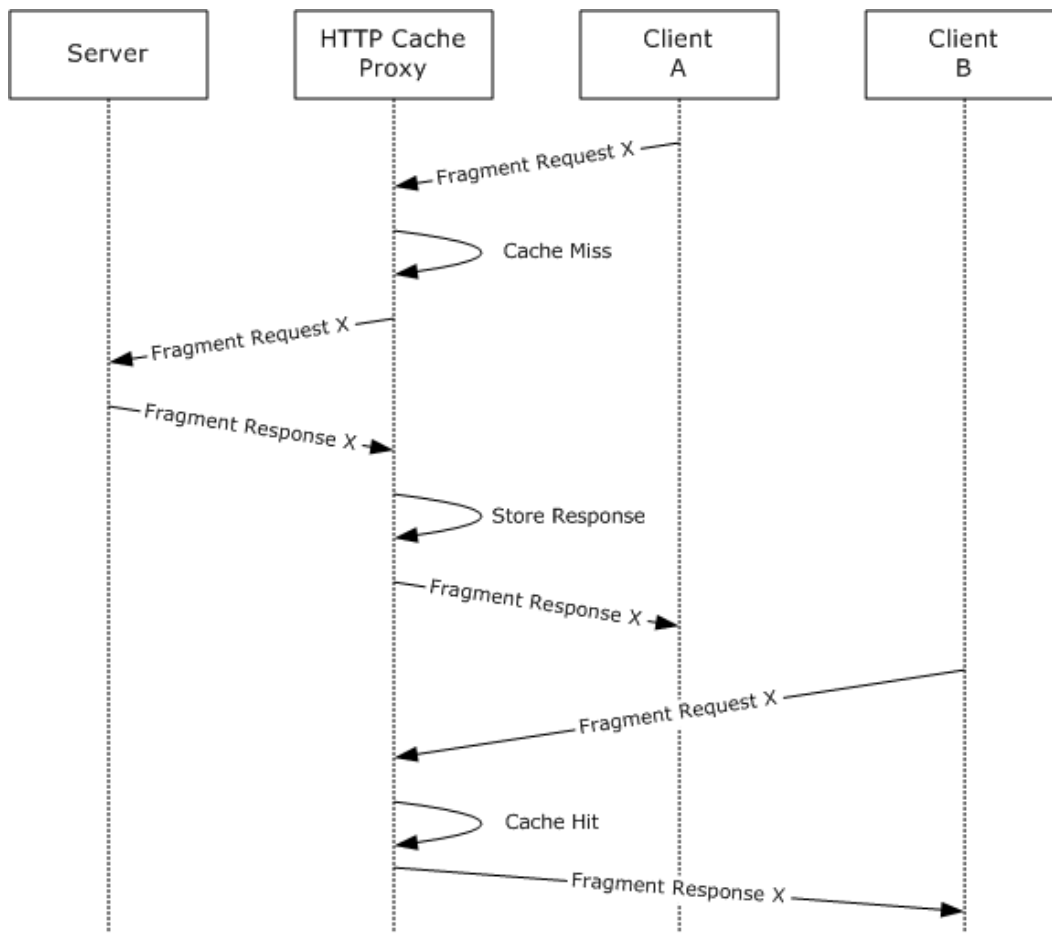
Figure 2: Typical communication pattern of requests for the same fragment

1.4 Relationship to Other Protocols

The IIS Smooth Streaming Transport Protocol uses HTTP [RFC2616] as its underlying transport.

The IIS Smooth Streaming Transport Protocol fulfills a similar function to established stateful media protocols, such as Real Time Streaming Protocol (RTSP) [RFC2326], with significantly greater scalability in Internet scenarios due to effective use of HTTP cache proxies.

1.5 Prerequisites/Preconditions

This protocol assumes HTTP [RFC2616] connectivity from the client to the server.

It is also assumed that the client is integrated with a higher-layer implementation that supports any **media format(s)** used and can otherwise play the media transmitted by the server.

1.6 Applicability Statement

This protocol is most appropriate for delivering media over the Internet or environments where HTTP cache proxies can be used to maximize scalability. It can be used on any network where HTTP [RFC2616] connectivity to the server is available.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Protocol Versions:** The IIS Smooth Streaming Transport Protocol is explicitly versioned using the **MajorVersion** and **MinorVersion** fields specified in section 2.2.2.1.

- **Security and Authentication Methods:** Security and authentication for the IIS Smooth Streaming Transport Protocol is performed at the underlying transport layer (HTTP) and does not restrict which of the mechanisms supported by HTTP can be used.

1.8 Vendor-Extensible Fields

The following fields in this protocol can be extended by vendors:

- **Custom Attributes in the Manifest Response:** This capability is provided by the **VendorExtensionAttributes** field, as specified in section 2.2.2. Implementers can ensure that extensions do not conflict by assigning extensions an XML Namespace unique to their implementation.

- **Custom Data Elements in the Manifest Response:** This capability is provided by the **VendorExtensionDataElement** fields, as specified in section 2.2.2.6.1. Implementers can ensure that extensions do not conflict by assigning extensions an XML Namespace unique to their implementation.

- **Custom Boxes in the Fragment Response:** This capability is provided by the **VendorExtensionUUID** field, as specified in section 2.2.4.

- **Custom Media Formats for Audio:** This capability is provided by the **AudioTag** and **CodecPrivateData** fields, as specified in section 2.2.2.5. Implementers can ensure that extensions do not conflict by assigning extensions a unique GUID (as specified in [MS-DTYP] section 2.3.4.1) embedded in the **CodecPrivateData** field, as specified in [WFEX].

- **Custom Descriptive Codes for Media Formats:** This capability is provided by the **FourCC** field, as specified in section 2.2.2.5. Implementers can ensure that extensions do not conflict by registering extension codes with the MPEG4-RA, as specified in [ISO/IEC-14496-12]

- **Custom HTTP Headers in the Manifest Response:** This capability is provided by the underlying transport layer (HTTP), as specified in [RFC2616] section 6.

- **Custom HTTP Headers in the Fragment Response:** This capability is provided by the underlying transport layer (HTTP), as specified in [RFC2616] section 6.

- **Custom HTTP Headers in the Fragment Request:** This capability is provided by the underlying transport layer (HTTP), as specified in [RFC2616] section 5.

- **Custom HTTP Headers in the Manifest Request:** This capability is provided by the underlying transport layer (HTTP), as specified in [RFC2616] section 5.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The Manifest Request and Fragment Request messages MUST be represented as HTTP Request messages, as specified by the Request rule of [RFC2616], subject to the following constraints:

- The Method MUST be "GET".

- For the Manifest Request message, the RequestURI MUST adhere to the syntax of the **ManifestRequest** field, specified in section 2.2.1.

- For the Fragment Request message, the RequestURI MUST adhere to the syntax of the **FragmentRequest** field, specified in section 2.2.3.

- The HTTP-Version SHOULD be HTTP/1.1.

The Manifest Response and Fragment Response messages MUST be represented as HTTP Response messages, as specified by the Response rule of [RFC2616], subject to the following constraints:

- The Status-Code SHOULD be 200.

- For the Manifest Response message, the message body MUST adhere to the syntax of the **ManifestResponse** field, specified in section 2.2.2.

- For the Fragment Response message, the message body MUST adhere to the syntax to the **FragmentResponse** field, specified in section 2.2.4.

- The HTTP-Version SHOULD be HTTP/1.1.

The Live Ingest Request MUST be represented as an HTTP Request message, as specified by the Request rule of [RFC2616], subject to the following constraints:

- The Method MUST be "POST".

- The "Transfer-Encoding: Chunked" header SHOULD replace the "Content-Length" header.

- The RequestURI MUST adhere to the syntax of the **LiveIngestRequest** field, specified in section 2.2.7.

- The HTTP-Version SHOULD be HTTP/1.1.

2.2 Message Syntax

The IIS Smooth Streaming Transport Protocol defines five types of messages:

- **Manifest Request** (section 2.2.1)

- **Manifest Response** (section 2.2.2)

- **Fragment Request** (section 2.2.3)

- **Fragment Response** (section 2.2.4)

- **Live Ingest Request** (section 2.2.7)

The following fields are commonly used across the message set. The syntax of each field is specified in ABNF [RFC5234].

TRUE: A case-insensitive string value for true, for use in XML attributes.

```
TRUE = "true"
```

FALSE: A case-insensitive string value for false, for use in XML attributes.

```
FALSE = "false"
```

STRING_UINT64: An unsigned decimal integer less than 2^64, written as a string.

```
STRING_UINT64 = 1*DIGIT
```

STRING_UINT32: An unsigned decimal integer less than 2^32, written as a string.

```
STRING_UINT32 = 1*DIGIT
```

STRING_UINT16: An unsigned decimal integer less than 2^16, written as a string.

```
STRING_UINT16 = 1*DIGIT
```

STRING_UINT8: An unsigned decimal integer less than 2^8, written as a string.

```
STRING_UINT8 = 1*DIGIT
```

S: Whitespace legal inside an XML Document, as defined in [XML].

```
S = 1* ( %x20 / %x09 / %x0D / %x0A )
```

Eq: An equality expression used for Attributes, as defined in [XML].

```
Eq = S "=" S
```

SQ: A single-quote character that contains Attributes, as defined in [XML].

```
SQ = %x27
```

DQ: A double-quote character that contains Attributes, as defined in [XML].

```
DQ = %x22
```

URL_SAFE_CHAR: A character that can safely appear in a URI, as specified in [RFC2396].

```
URL_SAFE_CHAR = <URL-safe character as defined in [RFC2616]>
```

URL_ENCODED_CHAR: A character encoded to safely appear in a URI, as specified in [RFC2396].

```
URL_ENCODED_CHAR = "%" HEXDIG HEXDIG
```

HEXCODED_BYTE: A hexadecimal coding of a byte, with the first character for the four high bits and the second character for the four low bits.

```
HEXCODED_BYTE = HEXDIG HEXDIG
```

XML_CHARDATA: XML data without Elements, as specified by the "CharData" field in [XML].

```
XML_CHARDATA = <XML character data as defined by CharData in [XML]>
```

IDENTIFIER: An identifier safe for use in data fields.

```
IDENTIFIER = *URL_SAFE_CHAR
```

IDENTIFIER_NONNUMERIC: A non-numeric identifier safe for use in data fields.

```
IDENTIFIER = ALPHA / UNDERSCORE *URL_SAFE_CHAR
UNDERSCORE = "_"
```

URISAFE_IDENTIFIER: An identifier safe for use in data fields part of a URI [RFC2396].

```
IDENTIFIER = *(URL_SAFE_CHAR / URL_ENCODED_CHAR)
```

URISAFE_IDENTIFIER_NONNUMERIC: A non-numeric identifier safe for use in data fields part of a URI [RFC2396].

```
IDENTIFIER = ALPHA / UNDERSCORE *(URL_SAFE_CHAR / URL_ENCODED_CHAR)
UNDERSCORE = "_"
```

2.2.1 Manifest Request

ManifestRequest and related fields contain data required to request a **manifest** from the server.

ManifestRequest (variable): The URI [RFC2396] of the Manifest resource.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
ManifestRequest = PresentationURI "/" "Manifest"
PresentationURI = [ "/" VirtualPath ] "/" PublishingPointName "." FileExtension
VirtualPath = URISAFE_IDENTIFIER
PublishingPointName = URISAFE_IDENTIFIER
FileExtension = "ism" / VendorExtensionFileExtension
VendorExtensionFileExtension = ALPHA *( ALPHA / DIGIT )
```

2.2.2 Manifest Response

ManifestResponse and related fields contain metadata required by the client to construct subsequent **FragmentRequest** messages and play back the data received.

ManifestResponse (variable): Metadata required by the client to play back the **presentation**. This field MUST be a Well-Formed XML Document [XML] subject to the following constraints:

- The Document's root Element is a **SmoothStreamingMedia** field.

- The Document's XML Declaration's major version is 1.

- The Document's XML Declaration's minor version is 0.

- The Document does not use a Document Type Definition (DTD).

- The Document uses an encoding that is supported by the client implementation.

- The XML Elements specified in this document do not use XML Namespaces.

Prolog (variable): The **Prolog** field, as specified in [XML].

Misc (variable): The **Misc** field, as specified in [XML].

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
ManifestResponse = prolog SmoothStreamingMedia Misc
```

2.2.2.1 SmoothStreamingMedia

SmoothStreamingMedia and related fields encapsulate metadata required to play the presentation.

SmoothStreamingMedia (variable): An XML Element that encapsulates all metadata required by the client to play back the presentation.

SmoothStreamingMediaAttributes (variable): The collection of XML attributes for the **SmoothStreamingMedia** Element. Attributes can appear in any order. However, the following

fields are required and MUST be present in **SmoothStreamingMediaAttributes**:
MajorVersionAttribute, **MinorVersionAttribute**, **DurationAttribute**.

MajorVersion (variable): The major version of the Manifest Response message. MUST be set to 2.

MinorVersion (variable): The minor version of the Manifest Response message. MUST be set to 0 or 2.

TimeScale (variable): The time scale of the Duration attribute, specified as the number of increments in one second. The default value is 10000000.

Duration (variable): The duration of the presentation, specified as the number of time increments indicated by the value of the **TimeScale** field.

IsLive (variable): Specifies the presentation type. If this field contains a TRUE value, it specifies that the presentation is a **live** presentation. Otherwise, the presentation is an **on-demand** presentation.

LookaheadCount (variable): Specifies the size of the server buffer, as an integer number of fragments. This field MUST be omitted for on-demand presentations.

DVRWindowLength (variable): The length of the **DVR window**, specified as the number of time increments indicated by the value of the **TimeScale** field. If this field is omitted for a live presentation or set to 0, the DVR window is effectively infinite. This field MUST be omitted for on-demand presentations.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
SmoothStreamingMedia = "<" SmoothStreamingMediaElementName S
                       SmoothStreamingMediaAttributes S ">"
                       S SmoothStreamingMediaContent S?
                       "</" SmoothStreamingMediaElementName ">"
SmoothStreamingMediaElementName = "SmoothStreamingMedia"
SmoothStreamingMediaAttributes = *(
                                  MajorVersionAttribute
                                  / MinorVersionAttribute
                                  / TimeScaleAttribute
                                  / DurationAttribute
                                  / IsLiveAttribute
                                  / LookaheadCountAttribute
                                  / DVRWindowLengthAttribute
                                  / VendorExtensionAttribute
                                 )
MajorVersionAttribute = S MajorVersionAttributeName S Eq S
                        (DQ MajorVersion DQ) / (SQ MajorVersion SQ) S?
MajorVersionAttributeName = "MajorVersion"
MajorVersion = "2"
MinorVersionAttribute = S MinorVersionAttributeName S Eq S
                        (DQ MinorVersion DQ) / (SQ MinorVersion SQ) S?
MinorVersionAttributeName = "MinorVersion"
MinorVersion = "0" / "2"
TimeScaleAttribute = S TimeScaleAttributeName S Eq S
                     (DQ TimeScale DQ) / (SQ TimeScale SQ) S?
TimeScaleAttributeName = "TimeScale"
TimeScale = STRING_UINT64
DurationAttribute = S DurationAttributeName S Eq S
                    (DQ Duration DQ) / (SQ Duration SQ) S?
```

```
DurationAttributeName = "Duration"
Duration = STRING_UINT64
IsLiveAttribute = S IsLiveAttributeName S Eq S
                 (DQ IsLive DQ) / (SQ IsLive SQ) S?
IsLiveAttributeName = "IsLive"
IsLive = TRUE / FALSE
LookaheadCountAttribute = S LookaheadCountAttributeName S Eq S
                         (DQ LookaheadCount DQ) / (SQ LookaheadCount SQ) S?
LookaheadCountAttributeName = "LookaheadCount"
LookaheadCount = STRING_UINT32
DVRWindowLengthAttribute = S DVRWindowLengthAttributeName S Eq S
                          (DQ DVRWindowLength DQ) / (SQ DVRWindowLength SQ) S?
DVRWindowLengthAttributeName = "DVRWindowLength"
DVRWindowLength= STRING_UINT64
SmoothStreamingMediaContent = [ ProtectionElement S?] 1* StreamElement
```

2.2.2.2   ProtectionElement

The **ProtectionElement** and related fields encapsulate metadata required to play back Protected Content.

ProtectionElement (variable): An XML Element that encapsulates metadata required by the client to play back protected content.

ProtectionHeaderElement (variable): An XML Element that encapsulates content protection metadata for a specific content protection system.

SystemID (variable): A UUID that uniquely identifies the Content Protection System to which this **ProtectionElement** pertains.

ProtectionHeaderContent (variable): Opaque data that the Content Protection System identified in the **SystemID** field can use to enable playback for authorized users, **encoded** using Base-64 Encoding [RFC3548].

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
ProtectionElement = "<" ProtectionElementName S ">"
                    S 1*( ProtectionHeaderElement S?)
                    "</" ProtectionElementName ">"
ProtectionElementName = "Protection"
ProtectionHeaderElement = "<" ProtectionHeaderElementName S
                          ProtectionHeaderAttributes S ">"
                          S ProtectionHeaderContent S?
                          "</" ProtectionHeaderElementName ">"
ProtectionHeaderAttributes = SystemIDAttribute
SystemIDAttribute = S SystemIDAttributeName S Eq S
                    (DQ SystemID DQ) / (SQ SystemID SQ) S?
SystemIDAttributeName = "SystemID"
SystemID = "{"
           4*4 HEXCODED_BYTE "-"
           2*2 HEXCODED_BYTE "-"
           2*2 HEXCODED_BYTE "-"
           2*2 HEXCODED_BYTE "-"
           6*6 HEXCODED_BYTE "-"
           "}"
```

```
     ProtectionHeaderContent = STRING_BASE64
```

2.2.2.3 StreamElement

The **StreamElement** and related fields encapsulate metadata required to play a specific **stream** in the presentation.

StreamElement (variable): An XML Element that encapsulates all metadata required by the client to play back a stream.

StreamAttributes (variable): The collection of XML Attributes for the **SmoothStreamingMedia** Element. Attributes can appear in any order. However, the following field is required and MUST be present in **StreamAttributes**: **TypeAttribute**. The following additional fields are required and MUST be present in **StreamAttributes** unless an Embedded Track is used in the **StreamContent** field: **NumberOfFragmentsAttribute**, **NumberOfTracksAttribute**, and **UrlAttribute**.

StreamContent (variable): Metadata describing available **tracks** and fragments.

Type (variable): The type of the stream: video, audio, or text. If the type specified is text, the following field is required and MUST appear in **StreamAttributes**: **SubtypeAttribute**. Unless the type specified is video, the following fields MUST NOT appear in **StreamAttributes**: **StreamMaxWidthAttribute**, **StreamMaxHeightAttribute**, **DisplayWidthAttribute**, and **DisplayHeightAttribute**.

StreamTimeScale (variable): The time scale for duration and time values in this stream, specified as the number of increments in one second.

Name (variable): The name of the stream.

NumberOfFragments (variable): The number of fragments available for this stream.

NumberOfTracks (variable): The number of tracks available for this stream.

Subtype (variable): A four-character code that identifies the intended use category for each **sample** in a text track. However, the **FourCC** field, specified in section 2.2.2.5, is used to identify the media format for each sample. The following range of values is reserved, with the following semantic meanings:

- "SCMD": Triggers for actions by the higher-layer implementation on the client

- "CHAP": Chapter markers

- "SUBT": Subtitles used for foreign-language audio

- "CAPT": Closed captions for people who are deaf

- "DESC": Media descriptions for people who are deaf

- "CTRL": Events the control the application business logic

- "DATA": Application data that does not fall into any of the above categories

Url (variable): A pattern used by the client to generate Fragment Request messages.

SubtypeControlEvents (variable): Control events for applications on the client.

StreamMaxWidth (variable): The maximum width of a video sample, in pixels.

StreamMaxHeight (variable): The maximum height of a video sample, in pixels.

DisplayWidth (variable): The suggested display width of a video sample, in pixels.

DisplayHeight (variable): The suggested display height of a video sample, in pixels.

ParentStream (variable): Specifies the **non-sparse stream** that is used to transmit timing information for this stream. If the **ParentStream** field is present, it indicates that the stream described by the containing **StreamElement** field is a sparse stream. If present, the value of this field MUST match the value of the Name field for a non-sparse stream in the presentation.

ManifestOutput (variable): Specifies whether sample data for this stream appears directly in the manifest as part of the **ManifestOutputSample** field, specified in section 2.2.2.6.1, if this field contains a TRUE value. Otherwise, the **ManifestOutputSample** field for fragments that are part of this stream MUST be omitted.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
StreamElement = "<" StreamElementName S
                StreamAttributes S ">"
                S StreamContent S?
                "</" StreamElementName ">"
Name = "StreamIndex"
StreamAttributes = *(
                    TypeAttribute
                    / SubtypeAttribute
                    / StreamTimeScaleAttribute
                    / NameAttribute
                    / NumberOfFragmentsAttribute
                    / NumberOfTracksAttribute
                    / UrlAttribute
                    / StreamMaxWidthAttribute
                    / StreamMaxHeightAttribute
                    / DisplayWidthAttribute
                    / DisplayHeightAttribute
                    / ParentStreamAttribute
                    / ManifestOutputAttribute
                    / VendorExtensionAttribute
                   )
TypeAttribute = S TypeAttributeName S Eq S
              (DQ Type DQ) / (SQ Type SQ) S?
TypeAttributeName = "Type"
Type = "video" / "audio" / "text"
SubtypeAttribute = S SubtypeAttributeName S Eq S
                  (DQ Subtype DQ) / (SQ Subtype SQ) S?
SubtypeAttributeName = "Subtype"
Subtype = 4*4 ALPHA
StreamTimeScaleAttribute = S StreamTimeScaleAttributeName S Eq S
                          (DQ StreamTimeScale DQ) / (SQ StreamTimeScale SQ) S?
StreamTimeScaleAttributeName = "TimeScale"
StreamTimeScale = STRING_UINT64
NameAttribute = S NameAttributeName S Eq S
              (DQ Name DQ) / (SQ Name SQ) S?
NameAttributeName = "Name"
Name = ALPHA *( ALPHA / DIGIT / UNDERSCORE / DASH )
NumberOfFragmentsAttribute = S NumberOfFragmentsAttributeName S Eq S
                            (DQ NumberOfFragments DQ) / (SQ NumberOfFragments SQ)
```

```
                                     S?
NumberOfFragmentsAttributeName = "Chunks"
NumberOfFragments = STRING_UINT32
NumberOfTracksAttribute = S NumberOfTracksAttributeName S Eq S
                        (DQ NumberOfTracks DQ) / (SQ NumberOfTracks SQ) S?
NumberOfTracksAttributeName = "QualityLevels"
NumberOfTracks = STRING_UINT32
UrlAttribute = S UrlAttributeName S Eq S
               (DQ Url DQ) / (SQ Url SQ) S?
UrlAttributeName = "Url"
Url = UrlPattern
StreamMaxWidthAttribute = S StreamMaxWidthAttributeName S Eq S
                        (DQ StreamMaxWidth DQ) / (SQ StreamMaxWidth SQ) S?
StreamMaxWidthAttributeName = "MaxWidth"
StreamMaxWidth = STRING_UINT32
StreamMaxHeightAttribute = S StreamMaxHeightAttributeName S Eq S
                         (DQ StreamMaxHeight DQ) / (SQ StreamMaxHeight SQ) S?
StreamMaxHeightAttributeName = "MaxHeight"
StreamMaxHeight = STRING_UINT32
DisplayWidthAttribute = S DisplayWidthAttributeName S Eq S
                        (DQ DisplayWidth DQ) / (SQ DisplayWidth SQ) S?
DisplayWidthAttributeName = "DisplayWidth"
DisplayWidth = STRING_UINT32
DisplayHeightAttribute = S DisplayHeightAttributeName S Eq S
                        (DQ DisplayHeight DQ) / (SQ DisplayHeight SQ) S?
DisplayHeightAttributeName = "DisplayHeight"
DisplayHeight = STRING_UINT32
ParentStreamAttribute = S ParentStreamAttributeName S Eq S
                        (DQ ParentStream DQ) / (SQ ParentStream SQ) S?
ParentStreamAttributeName = "ParentStreamIndex"
ParentStream = ALPHA *( ALPHA / DIGIT / UNDERSCORE / DASH )
ManifestOutputAttribute = S ManifestOutputAttributeName S Eq S
                         (DQ ManifestOutput DQ) / (SQ ManifestOutput SQ) S?
ManifestOutputAttributeName = "ManifestOutput"
ManifestOutput = TRUE / FALSE
StreamContent = 1*(TrackElement S?) *(StreamFragment S?)
```

2.2.2.4 UrlPattern

The **UrlPattern** and related fields define a pattern that can be used by the client to make semantically valid Fragment Requests for the presentation.

UrlPattern (variable): Encapsulates a pattern for constructing Fragment Requests.

BitrateSubstitution (variable): A placeholder expression for the **bit rate** of a track.

CustomAttributesSubstitution (variable): A placeholder expression for the Attributes used to disambiguate a track from other tracks in the stream.

TrackName (variable): A unique identifier that applies to all tracks in a stream.

StartTimeSubstitution (variable): A placeholder expression for the time of a fragment.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
UrlPattern = QualityLevelsPattern "/" FragmentsPattern
```

```
QualityLevelsPattern = QualityLevelsNoun "(" QualityLevelsPredicatePattern ")"
QualityLevelsNoun = "QualityLevels"
QualityLevelsPredicate = BitrateSubstitution ["," CustomAttributesSubstitution ]
Bitrate = "{bitrate}" / "{Bitrate}"
CustomAttributesSubstitution = "{CustomAttributes}"
FragmentsPattern = FragmentsNoun "(" FragmentsPatternPredicate ")";
FragmentsNoun = "Fragments"
FragmentsPatternPredicate = TrackName "=" StartTimeSubstitution;
TrackName = URISAFE_IDENTIFIER_NONNUMERIC
StartTimeSubstitution = "{start time}" / "{start_time}"
```

2.2.2.5 TrackElement

The **TrackElement** and related fields encapsulate metadata required to play a specific track in the stream.

TrackElement (variable): An XML Element that encapsulates all metadata required by the client to play a track.

TrackAttributes (variable): The collection of XML Attributes for the TrackElement. Attributes can appear in any order. However, the following fields are required and MUST be present in **TrackAttributes**: **IndexAttribute**, **BitrateAttribute**. If the track is contained in a stream whose Type is video, the following additional fields are also required and MUST be present in **TrackAttributes**: **MaxWidthAttribute**, **MaxHeightAttribute**, and **CodecPrivateDataAttribute**. If the track is contained in a stream whose Type is audio, the following additional fields are also required and MUST be present in **TrackAttributes**: **MaxWidthAttribute**, **MaxHeightAttribute**, **CodecPrivateDataAttribute**, **SamplingRateAttribute**, **ChannelsAttribute**, **BitsPerSampleAttribute**, **PacketSizeAttribute**, **AudioTagAttribute**, and **FourCCAttribute**.

Index (variable): An ordinal that identifies the track and MUST be unique for each track in the stream. The Index SHOULD start at 0 and increment by 1 for each subsequent track in the stream.

Bitrate (variable): The average bandwidth consumed by the track, in bits per second (bps). The value 0 MAY be used for tracks whose bit rate is negligible relative to other tracks in the presentation.

MaxWidth (variable): The maximum width of a video sample, in pixels.

MaxHeight (variable): The maximum height of a video sample, in pixels.

SamplingRate (variable): The Sampling Rate of an audio track, as defined in [ISO/IEC-14496-12].

Channels (variable): The Channel Count of an audio track, as defined in [ISO/IEC-14496-12].

AudioTag (variable): A numeric code that identifies which media format and variant of the media format is used for each sample in an audio track. The following range of values is reserved with the following semantic meanings:

- "1": The sample media format is Linear 8 or 16-bit Pulse Code Modulation

- "353": Microsoft Windows Media Audio v7, v8 and v9.*x* Standard (WMA Standard)

- "353": Microsoft Windows Media Audio v9.*x* and v10 Professional (WMA Professional)

- "85": ISO MPEG-1 Layer III (MP3)

- "255": ISO Advanced Audio Coding (AAC)

- "65534": Vendor-extensible format. If specified, the **CodecPrivateData** field SHOULD contain a hex-encoded version of the WAVE_FORMAT_EXTENSIBLE structure [WFEX].

BitsPerSample (variable): The sample Size of an audio track, as defined in [ISO/IEC-14496-12].

PacketSize (variable): The size of each audio **packet**, in bytes.

FourCC (variable): A four-character code that identifies which media format is used for each sample. The following range of values is reserved with the following semantic meanings:

- "H264": Video samples for this track use Advanced Video Coding, as specified in [ISO/IEC-14496-15]

- "WVC1": Video samples for this track use VC-1, as specified in [VC-1].

- "AACL": Audio samples for this track use AAC (Low Complexity), as specified in [ISO/IEC-14496-3]

- "WMAP": Audio samples for this track use WMA Professional

- A vendor extension value containing a registered with MPEG4-RA, as specified in [ISO/IEC-14496-12].

CodecPrivateData (variable): Data that specifies parameters specific to the media format and common to all samples in the track, represented as a string of hex-coded bytes. The format and semantic meaning of byte sequence varies with the value of the **FourCC** field as follows:

- The **FourCC** field equals "H264": The **CodecPrivateData** field contains a hex-coded string representation of the following byte sequence, specified in ABNF [RFC5234]:

  - %x00 %x00 %x00 %x01 SPSField %x00 %x00 %x00 %x01 SPSField

  - SPSField contains the Sequence Parameter Set (SPS).

  - PPSField contains the Slice Parameter Set (PPS).

- The **FourCC** field equals "WVC1": The **CodecPrivateData** field contains a hex-coded string representation of the VIDEOINFOHEADER structure, specified in [MSDN-VIH].

- The **FourCC** field equals "AACL": The **CodecPrivateData** field SHOULD be empty.

- The **FourCC** field equals "WMAP": The **CodecPrivateData** field contains the WAVEFORMATEX structure, specified in [WFEX], if the **AudioTag** field equals "65534", and SHOULD be empty otherwise.

- The **FourCC** is a vendor extension value: The format of the **CodecPrivateData** field is also vendor-extensible. Registration of the **FourCC** field value with MPEG4-RA, as specified in [ISO/IEC-14496-12], can be used to avoid collision between extensions.

NALUnitLengthField (variable): The number of bytes that specify the length of each Network Abstraction Layer (NAL) unit. This field SHOULD be omitted unless the value of the **FourCC** field is "H264". The default value is 4.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
Track = TrackWithoutContent / TrackWithContent
TrackWithoutContent = "<" TrackElementName S TrackAttributes S "/>"
TrackWithContent = "<" TrackElementName S TrackAttributes S ">"
                   S TrackContent S "</" TrackElementName ">"
TrackElementName = "QualityLevel"
TrackAttributes = *(
                    IndexAttribute
                    / BitrateAttribute
                    / CodecPrivateDataAttribute
                    / MaxWidthAttribute
                    / MaxHeightAttribute
                    / SamplingRateAttribute
                    / ChannelsAttribute
                    / BitsPerSampleAttribute
                    / PacketSizeAttribute
                    / AudioTagAttribute
                    / FourCCAttribute
                    / NALUnitLengthFieldAttribute
                    / VendorExtensionAttribute
                   )
IndexAttribute = S IndexAttributeName S Eq S
                 (DQ Index DQ) / (SQ Index SQ) S
IndexAttributeName = "Index"
Index = STRING_UINT32
BitrateAttribute = S BitrateAttributeName S Eq S
                   (DQ Bitrate DQ) / (SQ Bitrate SQ) S
BitrateAttributeName = "Bitrate"
Index = STRING_UINT32
MaxWidthAttribute = S MaxWidthAttributeName S Eq S
                    (DQ MaxWidth DQ) / (SQ MaxWidth SQ) S
MaxWidthAttributeName = "MaxWidth"
MaxWidth = STRING_UINT32
MaxHeightAttribute = S MaxHeightAttributeName S Eq S
                     (DQ MaxHeight DQ) / (SQ MaxHeight SQ) S
MaxHeightAttributeName = "MaxHeight"
MaxHeight = STRING_UINT32
CodecPrivateDataAttribute = S CodecPrivateDataAttributeName S Eq S
                            (DQ CodecPrivateData DQ) / (SQ CodecPrivateData SQ) S
CodecPrivateDatatAttributeName = "CodecPrivateData"
CodecPrivateData = *HEXCODED_BYTE
SamplingRateAttribute = S SamplingRateAttributeName S Eq S
                        (DQ SamplingRate DQ) / (SQ SamplingRate SQ) S
SamplingRateAttributeName = "SamplingRate"
SamplingRate = STRING_UINT32
ChannelsAttribute = S ChannelsAttributeName S Eq S
                    (DQ Channels DQ) / (SQ Channels SQ) S
ChannelsAttributeName = "Channels"
Channels = STRING_UINT16
BitsPerSampleAttribute = S BitsPerSampleAttributeName S Eq S
                         (DQ BitsPerSample DQ) / (SQ BitsPerSample SQ) S
BitsPerSampleAttributeName = "BitsPerSample"
BitsPerSample = STRING_UINT16
PacketSizeAttribute = S PacketSizeAttributeName S Eq S
                      (DQ PacketSize DQ) / (SQ PacketSize SQ) S
PacketSizeAttributeName = "PacketSize"
PacketSize = STRING_UINT32
AudioTagAttribute = S AudioTagAttributeName S Eq S
                    (DQ AudioTag DQ) / (SQ AudioTag SQ) S
PacketSizeAttributeName = "AudioTag"
```

```
AudioTag = STRING_UINT32
FourCCAttribute = S FourCCAttributeName S Eq S
                    (DQ FourCC DQ) / (SQ FourCC SQ) S
FourCCAttributeName = "AudioTag"
FourCC = 4*4 ALPHA
NALUnitLengthFieldAttribute = S NALUnitLengthFieldAttributeName S Eq S
                                (DQ NALUnitLengthField DQ)
                                / (SQ NALUnitLengthField SQ) S
NALUnitLengthFieldAttributeName = "NALUnitLengthField"
NALUnitLengthField = STRING_UINT16
TrackContent = CustomAttributes?
```

2.2.2.5.1 CustomAttributesElement

The **CustomAttributesElement** and related fields are used to specify metadata that disambiguates tracks in a stream.

CustomAttributes (variable): Metadata expressed as key/value pairs that disambiguate tracks.

CustomAttributeName (variable): The name of a custom Attribute for a track.

CustomAttributeValue (variable): The value of a custom Attribute for a track.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
CustomAttributesElement = S "<" CustomAttributesElementName S ">"
                            S 1*(AttributeElement S?)
                            "</" CustomAttributesElementName ">"
AttributeElement = "<" AttributeElementName S AttributeAttributes S "/>"
AttributeAttributes = (AttributeNameAttribute S AttributeValueAttribute)
                    / (AttributeValueAttribute S AttributeNameAttribute)
AttributeNameAttribute = S AttributeNameAttributeName S Eq S
                        (DQ CustomAttributeName DQ) / (SQ CustomAttributeName SQ) S?
AttributeNameAttributeName = "Name"
CustomAttributeName = IDENTIFIER
AttributeValueAttribute = S AttributeValueAttributeName S Eq S
                        (DQ CustomAttributeValue DQ) / (SQ CustomAttributeValue SQ) S?
AttributeValueAttributeName = "Value"
CustomAttributeValue = IDENTIFIER
```

2.2.2.6 StreamFragmentElement

The **StreamFragmentElement** and related fields are used to specify metadata for one set of related fragments in a stream. The order of repeated **StreamFragmentElement** fields in a containing **StreamElement** is significant for the correct function of the IIS Smooth Streaming Transport Protocol. To this end, the following elements make use of the terms "preceding" and "subsequent" **StreamFragmentElement** in reference to the order of these fields.

StreamFragmentElement (variable): An XML Element that encapsulates metadata for a set of related fragments. Attributes can appear in any order. However, either one or both of the following fields is required and MUST be present in **StreamFragmentAttributes**: **FragmentDuration**, **FragmentTime**.

FragmentNumber (variable): The ordinal of the **StreamFragmentElement** in the stream. If **FragmentNumber** is specified, its value MUST monotonically increase with the value of the **FragmentTime** field.

FragmentDuration (variable): The duration of the fragment, specified as a number of increments defined by the implicit or explicit value of the containing **StreamElement's StreamTimeScale** field. If the **FragmentDuration** field is omitted, its implicit value MUST be computed by the client by subtracting the value of the preceding **StreamFragmentElement's FragmentTime** field from the value of this **StreamFragmentElement's FragmentTime** field. If no preceding **StreamFragmentElement** exists, the implicit value of the **FragmentDuration** field MUST be computed by the client by subtracting the value of this **StreamFragmentElement FragmentTime** field from the subsequent **StreamFragmentElement's FragmentTime** field.

If no preceding or subsequent **StreamFragmentElement** field exists, the implicit value of the **FragmentDuration** field is the value of the **SmoothStreamingMedia's Duration** field.

FragmentTime (variable): The time of the fragment, specified as a number of increments defined by the implicit or explicit value of the containing **StreamElement's StreamTimeScale** field. If the **FragmentTime** field is omitted, its implicit value MUST be computed by the client by adding the value of the preceding **StreamFragmentElement's FragmentTime** field to the value of the preceding **StreamFragmentElement's FragmentDuration** field. If no preceding **StreamFragmentElement** exists, the implicit value of the **FragmentTime** field is 0.

FragmentRepeat (variable): The repeat count of the fragment, specified as the number of contiguous fragments with the same duration defined by the **StreamFragmentElement's FragmentTime** field. This value is one-based (a value of two means two fragments in the contiguous series). The **SmoothStreamingMedia's MajorVersion** and **MinorVersion** fields MUST both be set to 2.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
StreamFragmentElement = "<" StreamFragmentElementName S
                        StreamFragmentAttributes S ">"
                        S StreamFragmentContent S?
                        "</" StreamFragmentElementName ">"
StreamFragmentElementname = "c"
StreamFragmentAttributes = *(
                            FragmentNumberAttribute
                            / FragmentDurationAttribute
                            / FragmentTimeAttribute
                          )
FragmentNumberAttribute = S FragmentNumberAttributeName S Eq S
                          (DQ FragmentNumber DQ) / (SQ FragmentNumber SQ) S?
FragmentNumberAttributeName = "n"
FragmentNumber = STRING_UINT32
FragmentDurationAttribute = S FragmentDurationAttributeName S Eq S
                          (DQ FragmentDuration DQ) / (SQ FragmentDuration SQ) S?
FragmentDurationAttributeName = "d"
FragmentDuration = STRING_UINT64
FragmentTimeAttribute = S FragmentTimeAttributeName S Eq S
                      (DQ FragmentTime DQ) / (SQ FragmentTime SQ) S?
FragmentTimeAttributeName = "t"
FragmentTime = STRING_UINT64
FragmentRepeatAttribute = S FragmentRepeatAttributeName S Eq S?
                          (DQ FragmentRepeat DQ) / (SQ FragmentRepeat SQ) S?
FragmentRepeatAttributeName = "r"
```

```
FragmentRepeat = STRING_UINT64
StreamFragmentContent = *( TrackFragment S )
TrackFragment = "<" TrackFragmentElementName S
                TrackFragmentAttributes S ">"
                S 1*(TrackFragmentContent S?)
                "</" TrackFragmentElementName ">"
TrackFragmentAttributes = *(
                              TrackFragmentIndexAttribute
                              / VendorExtensionAttribute
                          )
TrackFragmentIndexAttribute = S TrackFragmentIndexAttribute S Eq S
                              (DQ TrackFragmentIndex DQ)
                              / (SQ TrackFragmentIndex SQ) S?
TrackFragmentIndexAttribute = "i"
TrackFragmentIndex = STRING_UINT32
TrackFragmentContent = VendorExtensionTrackData
VendorExtensionTrackData = XML_CHARDATA
```

2.2.2.6.1 TrackFragmentElement

TrackFragmentElement and related fields are used to specify metadata pertaining to a fragment for a specific track, rather than all versions of a fragment for a stream.

TrackFragmentElement (variable): An XML Element that encapsulates informative track-specific metadata for a specific fragment. Attributes can appear in any order. However, the following field is required and MUST be present in **TrackFragmentAttributes**: **TrackFragmentIndexAttribute**.

TrackFragmentIndex (variable): An ordinal that MUST match the value of the Index field for the track to which this **TrackFragment** field pertains.

ManifestOutputSample (variable): A string that contains the Base64-encoded representation of the raw bytes of the sample data for this fragment. This field MUST be omitted unless the **ManifestOutput** field for the corresponding stream contains a TRUE value.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
TrackFragmentElement = "<" TrackFragmentElementName S
                       TrackFragmentAttributes S ">"
                       S TrackFragmentContent S
                       "</" TrackFragmentElementName ">"
TrackFragmentElementName = "f"
TrackFragmentAttributes = *(
                              TrackFragmentIndexAttribute
                              / VendorExtensionAttribute
                          )
TrackFragmentIndexAttribute = S TrackFragmentIndexAttribute S Eq S
                              (DQ TrackFragmentIndex DQ)
                              / (SQ TrackFragmentIndex SQ) S?
TrackFragmentIndexAttribute = "i"
TrackFragmentIndex = STRING_UINT32
TrackFragmentContent = ManifestOutputSample
ManifestOutputSample = BASE64_STRING
```

2.2.3 Fragment Request

The **FragmentRequest** and related fields contain data required to request a fragment from the server.

FragmentRequest (variable): The URI [RFC2616] of the fragment resource.

BitratePredicate (variable): The bit rate of the requested fragment.

CustomAttributesPredicate (variable): An Attribute of the requested fragment used to disambiguate tracks.

CustomAttributesKey (variable): The name of the Attribute specified in the **CustomAttributesPredicate** field.

CustomAttributesValue (variable): The value of the Attribute specified in the **CustomAttributesPredicate**.

FragmentsNoun (variable): The type of response expected by the client.

StreamName (variable): The name of the stream that contains the requested fragment.

Time (variable): The time of the requested fragment.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
FragmentRequest = PresentationURI "/" QualityLevelsSegment "/" FragmentsSegment
                ; PresentationURI is specified in section 2.2.1
QualityLevelsSegment = QualityLevelsNoun "(" QualityLevelsPredicate ")"
QualityLevelsNoun = "QualityLevels"
QualityLevelsPredicate = BitratePredicate *( "," CustomAttributesPredicate )
BitratePredicate = STRING_UINT32
CustomAttributesPredicate = CustomAttributesKey "=" CustomAttributesValue
CustomAttributesKey = URISAFE_IDENTIFIER_NONNUMERIC
CustomAttributesValue = URISAFE_IDENTIFIER
FragmentsSegment = FragmentsNoun "(" FragmentsPredicate ")"
FragmentsNoun = FragmentsNounFullResponse
               / FragmentsNounMetadataOnly
               / FragmentsNounDataOnly
               / FragmentsNounIndependentOnly
FragmentsNounFullResponse = "Fragments"
FragmentsNounMetadataOnly = "FragmentInfo"
FragmentsNounDataOnly = "RawFragments"
FragmentsNounIndependentOnly = "KeyFrames"
FragmentsPredicate = StreamName "=" Time
StreamName = URISAFE_IDENTIFIER_NONNUMERIC
Time = STRING_UINT64
```

2.2.4 Fragment Response

The **FragmentResponse** and/or related fields encapsulate media and metadata specific to the requested fragment.

FragmentResponse (variable): The media and/or related metadata for a fragment.

FragmentFullResponse (variable): A Fragment Response that contains data and metadata.

FragmentMetadataResponse (variable): A Fragment Response that only contains metadata.

FragmentDataResponse (variable): A Fragment Response that contains only data.

FragmentMetadata (variable): Metadata for the fragment.

FragmentData (variable): Media data for the fragment.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
FragmentResponse = FragmentFullResponse
                 / FragmentMetadataResponse
                 / FragmentDataResponse
FragmentFullResponse = FragmentMetadata FragmentData
FragmentMetadataResponse = FragmentMetadata
FragmentDataResponse = SampleData
FragmentMetadata = MoofBox
FragmentData = MdatBox
```

SampleData, in the preceding ABNF syntax, is specified in section 2.2.4.8.

2.2.4.1   MoofBox

The **MoofBox** and related fields encapsulate metadata specific to the requested fragment. The syntax of **MoofBox** is a strict subset of the syntax of the Movie Fragment Box specified in [ISO/IEC-14496-12].

MoofBox (variable): Top-level metadata container for the requested fragment. The following fields are required and MUST be present in **MoofBoxChildren**: **MfhdBox**, **TrafBox**.

MoofBoxLength (4 bytes): The length of the **MoofBox** field, in bytes, including the **MoofBoxLength** field. If the value of the **MoofBoxLength** field is %00.00.00.01, the **MoofBoxLongLength** field MUST be present.

MoofBoxLongLength (8 bytes): The length of the **MoofBox** field, in bytes, including the **MoofBoxLongLength** field.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
MoofBox = MoofBoxLength MoofBoxType [MoofBoxLongLength]
          MoofBoxChildren
MoofBoxType = %d109 %d111 %d111 %d102
MoofBoxLength = BoxLength
MoofBoxLongLength = BoxLongLength
MoofBoxChildren = 2 *( MfhdBox / TrafBox / VendorExtensionUUIDBox )
```

2.2.4.2   MfhdBox

The **MfhdBox** and related fields specify the fragment's position in the sequence for the track. The syntax of the **MfhdBox** field is a strict subset of the syntax of the Movie Fragment Header Box defined in [ISO/IEC-14496-12].

MfhdBox (variable): Metadata container for the sequence information for the track.

MfhdBoxLength (4 bytes): The length of the **MfhdBox** field, in bytes, including the **MfhdBoxLength** field. If the value of the **MfhdBoxLength** field is %00.00.00.01, the **MfhdBoxLongLength** field MUST be present.

MfhdBoxLongLength (8 bytes): The length of the **MfhdBox** field, in bytes, including the **MfhdBoxLongLength** field.

SequenceNumber (4 bytes): An ordinal for the fragment in the track timeline. The **SequenceNumber** value for a fragment later in the timeline MUST be greater than for a fragment earlier in the timeline, but **SequenceNumber** values for consecutive Fragments are not required to be consecutive.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
MfhdBox = MfhdBoxLength MfhdBoxType [MfhdBoxLongLength] MfhdBoxFields
          MfhdBoxChildren
MfhdBoxType = %d109 %d102 %d104 %d100
MfhdBoxLength = BoxLength
MfhdBoxLongLength = LongBoxLength
MfhdBoxFields = SequenceNumber
SequenceNumber = UNSIGNED_INT32
MfhdBoxChildren = *( VendorExtensionUUIDBox )
```

2.2.4.3 TrafBox

The **TrafBox** and related fields encapsulate metadata specific to the requested fragment and track. The syntax of the **TrafBox** field is a strict subset of the syntax of the Track Fragment Box defined in [ISO/IEC-14496-12].

TrafBox (variable): Top-level metadata container for track-specific metadata for the fragment. The following fields are required and MUST be present in **TrafBoxChildren**: **TfhdBox**, **TrunBox**.

TrafBoxLength (4 bytes): The length of the **TrafBox** field, in bytes, including the **TrafBoxLength** field. If the value of the **TrafBoxLength** field is %00.00.00.01, the **TrafBoxLongLength** field MUST be present.

TrafBoxLongLength (8 bytes): The length of the **TrafBox** field, in bytes, including the **TrafBoxLongLength** field.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
TrafBox = TrafBoxLength TrafBoxType [TrafBoxLongLength]
          TrafBoxChildren
TrafBoxType = %d116 %d114 %d97 %d102
TrafBoxLength = BoxLength
TrafBoxLongLength = LongBoxLength
TrafBoxChildren = 2 *( TfhdBox / TrunBox
                     / VendorExtensionUUIDBox )
```

2.2.4.4 TfxdBox

The **TfxdBox** and related fields encapsulate the absolute timestamp and duration of a fragment in a live presentation. This field SHOULD be ignored if it appears in an on-demand presentation.

TfxdBox (variable): Metadata container for per sample defaults.

TfxdBoxLength (4 bytes): The length of the **TfxdBox** field, in bytes, including the **TfxdBoxLength** field. If the value of the **TfxdBoxLength** field is %00.00.00.01, the **TfxdBoxLongLength** field MUST be present.

TfxdBoxLongLength (8 bytes): The length of the **TfxdBox** field, in bytes, including the **TfxdBoxLongLength** field.

TfxdBoxVersion (1 byte): The box version. If this field contains the value %x01, the **TfxdBoxDataFields64** field MUST be present inside the **TfxdBoxFields** field. Otherwise, the **TfxdBoxDataFields32** field MUST be present inside the **TfxdBoxFields** field.

FragmentAbsoluteTime32 (4 bytes): The absolute timestamp of the first sample of the fragment, in time scale increments for the track.

FragmentDuration32 (4 bytes): The total duration of all samples in the fragment, in time scale increments for the track.

FragmentAbsoluteTime64 (8 bytes): The absolute timestamp of the first sample of the fragment, in time scale increments for the track.

FragmentDuration64 (8 bytes): The total duration of all samples in the fragment, in time scale increments for the track.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
TfxdBox = TfxdBoxLength TfxdBoxType [TfxdBoxLongLength] TfxdBoxUUID TfxdBoxFields
          TfxdBoxChildren
TfxdBoxType = %d117 %d117 %d105 %d100
TfxdBoxLength = BoxLength
TfxdBoxLongLength = LongBoxLength
TfxdBoxUUID = %x6D %x1D %x9B %x05 %x42 %xD5 %x44 %xE6
              %x80 %xE2 %x14 %x1D %xAF %xF7 %x57 %xB2
TfxdBoxFields = TfxdBoxVersion
                TfxdBoxFlags
                TfxdBoxDataFields32 / TfxdBoxDataFields64
TfxdBoxVersion = %x00 / %x01
TfxdBoxFlags = 24*24 RESERVED_BIT
TfxdBoxDataFields32 = FragmentAbsoluteTime32
                      FragmentDuration32
TfxdBoxDataFields64 = FragmentAbsoluteTime64
                      FragmentDuration64
FragmentAbsoluteTime64 = UNSIGNED_INT32
FragmentDuration64 = UNSIGNED_INT32
FragmentAbsoluteTime64 = UNSIGNED_INT64
FragmentDuration64 = UNSIGNED_INT64
TfxdBoxChildren = *( VendorExtensionUUIDBox )
```

2.2.4.5   TfrfBox

The **TfrfBox** and related fields encapsulate the absolute timestamp and duration for one or more subsequent fragments of the same track in a live presentation. This field SHOULD be ignored if it appears in an on-demand presentation. For a live presentation, this field MUST be present unless one of the following conditions is true:

- The containing track for this fragment is a **sparse track**.

- The number of subsequent fragments in the track is less than the value of the **LookaheadCount** field, specified in section 2.2.2.1, for the presentation.

- The **LookaheadCount** field is set to 0.

TfrfBox (variable): Metadata container for per sample defaults.

TfrfBoxLength (4 bytes): The length of the **TfrfBox** field, in bytes, including the **TfrfBoxLength** field. If the value of the **TfrfBoxLength** field is %x00.00.00.01, the **TfrfBoxLongLength** field MUST be present.

TfrfBoxLongLength (8 bytes): The length of the **TfrfBox** field, in bytes, including the **TfrfBoxLongLength** field.

TfrfBoxVersion (1 byte): The box version. If this field contains the value %x01, the **TfrfBoxDataFields64** field MUST be present inside the **TfrfBoxFields** field. Otherwise, the **TfrfBoxDataFields32** field MUST be present inside the **TfrfBoxFields** field.

FragmentCount (4 byte): The number of fragments for which the **TfrfBox** field contains information.

TfrfBoxDataFields32 (variable): The absolute timestamps and durations for a set of subsequent fragments, represented as 32-bit values. If the value of the **TfrfBoxVersion** field is %x00, there MUST be exactly **FragmentCount** instances of this field.

TfrfBoxDataFields64 (variable): The absolute timestamps and durations for a set of subsequent fragments, represented as 64-bit values. If the value of the **TfrfBoxVersion** field is %x00, there MUST be exactly **FragmentCount** instances of this field.

FragmentAbsoluteTime32 (4 bytes): The absolute timestamp of the first sample of a subsequent fragment, in time scale increments for the track.

FragmentDuration32 (4 bytes): The total duration of all samples in a subsequent fragment, in time scale increments for the track.

FragmentAbsoluteTime64 (8 bytes): The absolute timestamp of the first sample of a subsequent fragment, in time scale increments for the track.

FragmentDuration64 (8 bytes): The total duration of all samples in a subsequent fragment, in time scale increments for the track.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
TfrfBox = TfrfBoxLength TfrfBoxType [TfrfBoxLongLength] TfrfBoxUUID TfrfBoxFields
          TfrfBoxChildren
TfrfBoxType = %d117 %d117 %d105 %d100
TfrfBoxLength = BoxLength
TfrfBoxLongLength = LongBoxLength
TfrfBoxUUID = %xD4 %x80 %x7E %xF2 %xCA %x39 %x46 %x95
              %x8E %x54 %x26 %xCB %x9E %x46 %xA7 %x9F
TfrfBoxFields = TfrfBoxVersion
                TfrfBoxFlags
                FragmentCount
                (1* TfrfBoxDataFields32) / (1* TfrfBoxDataFields64)
TfrfBoxVersion = %x00 / %x01
```

```
TfrfBoxFlags = 24*24 RESERVED_BIT
FragmentCount = UINT8
TfrfBoxDataFields32 = FragmentAbsoluteTime32
                      FragmentDuration32
TfrfBoxDataFields64 = FragmentAbsoluteTime64
                      FragmentDuration64
FragmentAbsoluteTime64 = UNSIGNED_INT32
FragmentDuration64 = UNSIGNED_INT32
FragmentAbsoluteTime64 = UNSIGNED_INT64
FragmentDuration64 = UNSIGNED_INT64
TfrfBoxChildren = *( VendorExtensionUUIDBox )
```

2.2.4.6 TfhdBox

The **TfhdBox** and related fields encapsulate defaults for per sample metadata in the fragment. The syntax of the **TfhdBox** field is a strict subset of the syntax of the Track Fragment Header Box defined in [ISO/IEC-14496-12].

TfhdBox (variable): Metadata container for per sample defaults.

TfhdBoxLength (4 bytes): The length of the **TfhdBox** field, in bytes, including the **TfhdBoxLength** field. If the value of the **TfhdBoxLength** field is %00.00.00.01, the **TfhdBoxLongLength** field MUST be present.

TfhdBoxLongLength (8 bytes): The length of the **TfhdBox** field, in bytes, including the **TfhdBoxLongLength** field.

BaseDataOffset (8 bytes): The offset, in bytes, from the beginning of the **MdatBox** field to the sample field in the **MdatBox** field.

SampleDescriptionIndex (4 bytes): The ordinal of the sample description for the track that is applicable to this fragment. This field SHOULD be omitted.

DefaultSampleDuration (4 bytes): The default duration of each sample, in increments defined by the **TimeScale** for the track.

DefaultSampleSize (4 bytes): The default size of each sample, in bytes.

DefaultSampleFlags (4 bytes): The default value of the **SampleFlags** field for each sample.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
TfhdBox = TfhdBoxLength TfhdBoxType [TfhdBoxLongLength] TfhdBoxFields
          TfhdBoxChildren
TfhdBoxType = %d116 %d102 %d104 %d100
TfhdBoxLength = BoxLength
TfhdBoxLongLength = LongBoxLength
TfhdBoxFields = TfhdBoxVersion
                TfhdBoxFlags
                [ BaseDataOffset ]
                [ SampleDescriptionIndex ]
                [ DefaultSampleDuration ]
                [ DefaultSampleSize ]
                [ DefaultSampleFlags ]
TfhdBoxVersion = %x00
TfhdBoxFlags = 18*18 RESERVED_BIT
```

```
                DefaultSampleFlagsPresent
                DefaultSampleSizePresent
                DefaultSampleDurationPresent
                RESERVED_BIT
                SampleDescriptionIndexPresent
                BaseDataOffsetPresent
    BaseDataOffset = UNSIGNED_INT64
    SampleDescriptionIndex = UNSIGNED_INT32
    DefaultSampleDuration = UNSIGNED_INT32
    DefaultSampleSize = UNSIGNED_INT32
    DefaultSampleFlags = SampleFlags
    TfhdBoxChildren = *( VendorExtensionUUIDBox )
```

2.2.4.7 TrunBox

The **TrunBox** and related fields encapsulate per sample metadata for the requested fragment. The syntax of **TrunBox** is a strict subset of the syntax of the Track Fragment Run Box defined in [ISO/IEC-14496-12].

TrunBox (variable): Container for per sample metadata.

TrunBoxLength (4 bytes): The length of the **TrunBox** field, in bytes, including **TrunBoxLength** field. If the value of the **TrunBoxLength** field is %00.00.00.01, the **TrunBoxLongLength** field MUST be present.

TrunBoxLongLength (8 bytes): The length of the **TrunBox** field, in bytes, including the **TrunBoxLongLength** field.

SampleCount (4 bytes): The number of samples in the fragment.

FirstSampleFlagsPresent (1 bit): Indicates that the default flags for the first sample are replaced if this field takes the value %b1.

SampleSizePresent (1 bit): Indicates that each sample has its own size if this field takes the value %b1. If this field is not present, then the default value specified by the **DefaultSampleSize** field is used.

SampleDurationPresent (1 bit): Indicates that each sample has its own duration if this field takes the value %b1. If this field is not present, then the default value specified by the **DefaultSampleDuration** field is used.

SampleFlagsPresent (1 bit): Indicates that each sample has its own flags if this field takes the value %b1. If this field is not present, then the default value specified by the **DefaultSampleFlags** field is used.

SampleCompositionTimeOffsetPresent (1 bit): Indicates that each sample has a composition time offset if this field takes the value %b1.

FirstSampleFlags (4 bytes): The value of the **SampleFlags** field for the first sample. This field MUST be present if and only if the **FirstSampleFlagsPresent** takes the value %b1.

SampleSize (4 bytes): The size of each sample, in bytes. This field MUST be present if and only if the **SampleSizePresent** field takes the value %b1. If this field is not present, its implicit value is the value of the **DefaultSampleSize** field.

SampleDuration (4 bytes): The duration of each sample, in increments defined by the **TimeScale** for the track. This field MUST be present if and only if SampleDurationPresent takes the value %b1. If this field is not present, its implicit value is the value of the **DefaultSampleDuration** field.

TrunBoxSampleFlags (4 bytes): The Sample flags of each sample. This field MUST be present if and only if the **SampleFlagsPresent** field takes the value %b1. If this field is not present, its implicit value is the value of the **DefaultSampleFlags** field. If the **FirstSampleFlags** field is present and this field is omitted, this field's implicit value for the first sample in the fragment MUST be the value of the **FirstSampleFlags** field.

SampleCompositionTimeOffset (4 bytes): The Sample Composition Time offset of each sample, as defined in [ISO/IEC-14496-12]. This field MUST be present if and only if the **SampleCompositionTimeOffsetPresent** field takes the value %b1.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
TrunBox = TrunBoxLength TrunBoxType [TrunBoxLongLength] TrunBoxFields
          TrunBoxChildren
TrunBoxType = %d116 %d114 %d117 %d110
TrunBoxLength = BoxLength
TrunBoxLongLength = LongBoxLength
TrunBoxFields = TrunBoxVersion
                TrunBoxFlags
                SampleCount
                [ FirstSampleFlags ]
                *( TrunBoxPerSampleFields )
                ; TrunBoxPerSampleFields MUST be repeated exactly SampleCount times
TrunBoxFlags = 12*12 RESERVED_BIT
                SampleCompositionTimeOffsetPresent
                SampleFlagsPresent
                SampleSizePresent
                SampleDurationPresent
                RESERVED_BIT
                RESERVED_BIT
                RESERVED_BIT
                RESERVED_BIT
                RESERVED_BIT
                FirstSampleFlagsPresent
                RESERVED_BIT
                RESERVED_BIT
SampleCompositionTimeOffsetPresent = BIT
SampleFlagsPresent = BIT
SampleSizePresent = BIT
SampleDurationPresent = BIT
FirstSampleFlagsPresent = BIT
FirstSampleFlags = SampleFlags
TrunBoxPerSampleFields = [ SampleDuration ]
                        [ SampleSize ]
                        [ TrunBoxSampleFlags ]
                        [ SampleCompositionTimeOffset ]
SampleDuration = UNSIGNED_INT32
SampleSize = UNSIGNED_INT32
TrunBoxSampleFlags = SampleFlags
SampleCompositionTimeOffset = UNSIGNED_INT32
TrunBoxChildren = *( VendorExtensionUUIDBox )
```

2.2.4.8 MdatBox

The **MdatBox** and related fields encapsulate media data for the requested fragment. The syntax of the **MdatBox** field is a strict subset of the syntax of the Media Data Container Box defined in [ISO/IEC-14496-12].

MdatBox (variable): Media data container.

MdatBoxLength (4 bytes): The length of the **MdatBox** field, in bytes, including the **MdatBoxLength** field. If the value of the **MdatBoxLength** field is %00.00.00.01, the **MdatBoxLongLength** field MUST be present.

MdatBoxLongLength (8 bytes): The length of the **MdatBox** field, in bytes, including the **MdatBoxLongLength** field.

SampleData (variable): A single sample of media. Sample boundaries in the **MdatBox** field are defined by the values of the **DefaultSampleSize** and **SampleSize** fields in the **TrunBox** field.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
MdatBox = MdatBoxLength MdatBoxType [MdatBoxLongLength]
          MdatBoxFields
MoofBoxType = %d109 %d100 %d97 %d116
MoofBoxLength = BoxLength
MoofBoxLongLength = LongBoxLength
MdatBoxFields = *( SampleData )
SampleData = *BYTE
```

2.2.4.9 Fragment Response Common Fields

This section defines the common fields used in the Fragment Response message for the following fields: MoofBox, MfhdBox, TrafBox, TfxdBox, TfxfBox, TfhdBox, and TrunBox.

SampleFlags (4 bytes): A comprehensive Sample flags field.

SampleDependsOn (2 bits): Specifies whether this sample depends on another sample.

SampleDependsOnUnknown (2 bits): Unknown whether this sample depends on other samples.

SampleDependsOnOthers (2 bits): This sample depends on other samples.

SampleDoesNotDependOnOthers (2 bits): This sample does not depend on other samples.

SampleIsDependedOn (2 bits): Specifies whether other samples depend on this sample.

SampleIsDependedOnUnknown (2 bits): Unknown whether other samples depend on this sample.

SampleIsNotDisposable (2 bits): Other samples depend on this sample.

SampleIsDisposable (2 bits): No other samples depend on this sample.

SampleHasRedundancy (2 bits): Specifies whether this sample uses redundant coding.

RedundantCodingUnknown (2 bits): Unknown whether this sample uses redundant coding.

RedundantCoding (2 bits): This sample uses redundant coding.

NoRedundantCoding (2 bits): This sample does not use redundant coding.

SampleIsDifferenceValue (1 bit): A value of %b1 specifies that the sample is not a random access point in the stream.

SamplePaddingValue (3 bits): The sample padding value, as specified in [ISO/IEC-14496-12].

SampleDegradationPriority (2 bytes): The sample degradation priority, as specified in [ISO/IEC-14496-12].

VendorExtensionUUIDBox (Variable): A user extension box, as specified in [ISO/IEC-14496-12].

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
SampleFlags = 6*6 RESERVED_BIT
                     SampleDependsOn
                     SampleIsDependedOn
                     SampleHasRedundancy
                     SamplePaddingValue
                     SampleIsDifferenceValue
                     SampleDegradationPriority
SampleDependsOn = SampleDependsOnUnknown
                / SampleDependsOnOthers
                / SampleDoesNotDependsOnOthers
SampleDependsOnUnknown = %b0 %b0
SampleDependsOnOthers = %b0 %b1
SampleDoesNotDependOnOthers = %b1 %b0
SampleIsDependedOn = SampleIsDependedOnUnknown
                   / SampleIsNotDisposable
                   / SampleIsDisposable
SampleIsDependedOnUnknown = %b0 %b0
SampleIsNotDisposable = %b0 %b1
SampleIsDisposable = %b1 %b0
SampleHasRedundancy = RedundantCodingUnknown
                    / RedundantCoding
                    / NoRedundantCoding
RedundantCodingUnknown = %b0 %b0
RedundantCoding = %b0 %b1
NoRedundantCoding = %b1 %b0
SamplePaddingValue = 3*3 BIT
SampleIsDifferenceValue = BIT
SampleDegradationPriority = UNSIGNED_INT16
VendorExtensionUUIDBox = UUIDBoxLength UUIDBoxType [UUIDBoxLongLength] UUIDBoxUUID
                         UUIDBoxData
UUIDBoxType = %d117 %d117 %d105 %d100
UUIDBoxLength = BoxLength
UUIDBoxLongLength = LongBoxLength
UUIDBoxUUID = UUID
UUIDBoxData = *BYTE
BoxLength = UNSIGNED_INT32
LongBoxLength = UNSIGNED_INT64
RESERVED_UNSIGNED_INT64 = %x00 %x00 %x00 %x00 %x00 %x00 %x00 %x00
UNSIGNED_INT64 = 8*8 BYTE
RESERVED_UNSIGNED_INT32 = %x00 %x00 %x00 %x00
UNSIGNED_INT32 = 4*4 BYTE
```

```
RESERVED_UNSIGNED_INT16 = %x00 %x00
UNSIGNED_INT16 = 2*2 BYTE
RESERVED_BYTE = %x00
BYTE = 8*8 BIT
RESERVED_BIT = %b0
BIT = %b0 / %b1
```

2.2.5 Sparse Stream Pointer

The **SparseStreamPointer** and related fields contain data required to locate the latest fragment of a sparse stream. This message is used in conjunction with a Fragment Response message.

SparseStreamPointer (variable): A set of data that indicates the latest fragment for all related sparse streams.

SparseStreamSet (variable): The set of latest fragment pointer for all sparse streams related to a single requested fragment.

SparseStreamFragment (variable): The latest fragment pointer for a single related sparse stream.

SparseStreamName (variable): The stream Name of the related Sparse Name. The value of this field MUST match the **Name** field of the **StreamElement** field that describes the stream, specified in section 2.2.2.3, in the Manifest Response.

SparseStreamTimeStamp (variable): The timestamp of the latest timestamp for a fragment for the SparseStream that occurs at the same point in time or earlier than the presentation than the requested fragment.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
SparseStreamPointer = (HeaderData DELIMETER)? "ChildTrack" "="
                      DQ SparseStreamSet *( DELIMETER  SparseStreamSet ) DQ
HeaderData = 1*CHAR
DELIMETER = ";"
SparseStreamSet = SparseStreamFragment *( "," SparseStreamFragment )
SparseStreamFragment = SparseStreamName "=" SparseStreamTimeStamp
SparseStreamTimeStamp = STRING_UINT64
```

2.2.6 Fragment Not Yet Available

The Fragment Not Yet Available message is an HTTP Response with an empty message body field and the HTTP Status Code 412 Precondition Failed, as specified in [RFC2616].

2.2.7 Live Ingest

The **LiveIngest** and related fields contain data required to request the start of a live broadcast.

LiveIngestRequest (variable): The URI [RFC2396] to which the **LiveIngestRequest** is sent.

Identifier (variable): A unique URISAFE_IDENTIFIER that enables the server to differentiate between different streams. Each identifier can have at most one active connection.

EventID (variable): An optional identifier that enables the reuse of URLs without collision due to downstream cache pollution. Publishing streams with different event names to the same publish URL simultaneously is an error. All encoders MUST use the same **EventID** identifier, either blank or a string. The default value is the empty string.

StreamID (variable): A unique identifier used to collate fragments in the case of encoder failover. Allows separate encoder nodes to POST to separate URLs, but multiple active connection URLs with the same **StreamID** identifier can be used for redundancy, in which case the server will filter out duplicated or out-of-order fragments. Commonly used to distinguish between video quality (for example "Streams(1080p)", "Streams(720p)", "Streams(480p)").

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
LiveIngestRequest = Protocol "://" BroadcastURL Identifier
Protocol = "http" / "https"
BroadcastURL = ServerAddress "/" PresentationPath
ServerAddress = URISAFE_IDENTIFIER
PresentationPath = URISAFE_IDENTIFIER ".isml"
Identifier = [EventID ]StreamID
EventID = "/Events(" URISAFE_IDENTIFIER ")"
StreamID = "/Streams(" URISAFE_IDENTIFIER ")"
```

LiveIngestMessage (variable): The structure of the long-running POST operation requests sent from the encoder to the **LiveIngestRequest**.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
LiveIngestMessage = FileType [StreamManifest] LiveServerManifest MoovBox *1Fragment
```

2.2.7.1 FileType

FileType (variable): specifies the sub-type and intended use of the MPEG-4 ([MPEG4-RA]) file, and high-level attributes.

MajorBrand (variable): The major brand of the media file. MUST be set to "isml".

MinorVersion (variable): The minor version of the media file. MUST be set to 1.

CompatibleBrands (variable): Specifies the supported brands of MPEG-4. MUST include "piff" and "iso2".

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
FileType = MajorBrand MinorVersion CompatibleBrands
MajorBrand = STRING_UINT32
MinorVersion = STRING_UINT32
CompatibleBrands = "piff" "iso2" 0*(STRING_UINT32)
```

2.2.7.2 StreamManifestBox

The **StreamManifestBox** and related fields contain metadata required to inform the client of all comprising streams in a broadcast. If the **StreamManifestBox** is present in a POST request, the server sends a response, but does not initialize the broadcast until all of the streams enumerated in

the **StreamManifest** have sent an initial POST request. If the desired functionality is for the server broadcast to begin as soon as the first encoder connects, the **StreamManifestBox** MUST be omitted.

StreamManifestBox (variable): Contains the **StreamManifest** and associated metadata.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
StreamManifestBox = SMBoxType SMBoxLength SMBoxUUID SMVersion SMFlagsStreamManifest
SMBoxType = %d117 %d117 %d105 %d100
SMBoxLength = BoxLength
SMBoxUUID = %x3C %x2F %xE5 %x1B %xEF %xEE %x40 %xA3
            %xAE %x81 %x53 %x00 %x19 %x9D %xC3 %x48
SMVersion = STRING_UINT8
SMFlags = 24*24 RESERVED_BIT
```

StreamManifest (variable): A SMIL 2.0-compliant document [SMIL2.1] that informs the server of all streams to allow broadcast delay until all are acquired. This field MUST be a Well-Formed XML Document [XML] subject to the following constraints:

- The Document's root Element is a SMIL element.

- The Document's XML Declaration's major version is 1.

- The Document's XML Declaration's minor version is 0.

- The Document does not use a Document Type Definition (DTD).

- The Document uses an encoding that is supported by the client implementation.

- The XML Elements specified in this document MUST use
  "http://www.w3.org/2001/SMIL20/Language" for a namespace. Instead of the default
  namespace, a named namespace MAY be used, in which case all the tags described below MUST
  have the namespace prefix that maps to this XML namespace.

Prolog (variable): The **Prolog** field, as specified in [XML].

StreamSMIL (variable): The body of the document field, as specified in 2.2.7.2.1.

Misc (variable): The **Misc** field, as specified in [XML].

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
StreamManifest = prolog StreamSMIL Misc
```

2.2.7.2.1   StreamSMIL

The **StreamSMIL** and related fields encapsulate the data that is required for the client to identify all the streams in a presentation.

SMIL (variable): an XML element that encapsulates all the metadata required for the client to identify all the streams in a presentation.

SMILReference (variable): Specifies a single stream. The server MUST wait for this stream before starting the broadcast. The **src** attribute is required and specifies the stream's relative URL.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
SMIL = "<" SMILMediaElementName SMILMediaNamespace ">" S?
        SMILStreamBody S?
       "</" SMILMediaElementName ">"
SMILMediaElementName = "smil"
SMILMediaNamespace = "xmlns" Eq DQ "http://www.w3.org/2001/SMIL20/Language" DQ
SMILStreamBody = "<body>" S "<par>" S *1(SMILReference) S "</par>" S "</body>"
SMILReference = "<ref" S "src" Eq DQ "Streams(" IDENTIFIER ")" DQ S "/>"
```

2.2.7.3   LiveServerManifestBox

The **LiveServerManifestBox** and related fields comprise the data provided to the server by the encoder. The data enables the server to interpret the incoming live stream and assign semantic meaning to the stream's tracks.

LiveServerManifestBox (variable): Contains the **LiveServerManifestBox** and associated metadata.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
LiveServerManifestBox = LSBoxType LSBoxLength LSBoxUUID LSVersion LSFlags LSManifest
LSBoxType = %d117 %d117 %d105 %d100
LSBoxLength = BoxLength
LSUUID = %xA5 %xD4 %x0B %x30 %xE8 %x14 %x11 %xDD
           %xBA %x2F %x08 %x00 %x20 %x0C %x9A %x66
LSVersion= IDENTIFIER
LSFlags= 24*24 RESERVED_BIT
```

LiveServerManifest (variable): SMIL 2.0-compliant document [SMIL2.1] that specifies the metadata for all the tracks that appear in a live presentation. This field MUST be a Well-Formed XML Document [XML] subject to the following constraints:

- The Document's root Element is a SMIL field.

- The Document's XML Declaration's major version is 1.

- The Document's XML Declaration's minor version is 0.

- The Document does not use a Document Type Definition (DTD).

- The Document uses an encoding that is supported by the client implementation.

- The XML Elements specified in this document MUST use "http://www.w3.org/2001/SMIL20/Language" for a namespace. Instead of the default namespace, a named namespace MAY be used, in which case all the tags described below MUST have the namespace prefix that maps to this XML namespace.

- The XML Elements specified in this document do not use XML Namespaces.

Prolog (variable): The **Prolog** field, as specified in [XML].

LiveSMIL (variable): The body of the document field, as specified in section 2.2.7.3.1.

Misc (variable): The **Misc** field, as specified in [XML].

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
LiveServerManifest = prolog LiveSMIL Misc
```

2.2.7.3.1   LiveSMIL

The **LiveSMIL** and related fields comprise the data required for the client to identify all the streams in a presentation.

SMIL (variable): The root container used by SMIL 2.0 [SMIL2.1]. The **xmlns** field MUST be set to http://www.w3.org/2001/SMIL20/Language.

SMILLiveHead (variable): The head element that contains the presentation-level metadata.

SMILLiveMeta (variable): The metadata of the presentation. The following attributes are supported:

- name: Specifies the semantic meaning of metadata; this attribute MUST be present.

- content: Specifies the value of metadata; this attribute MUST be present.

SMILLiveBody (variable): The body element contains track information and references to other media.

SMILTrack (variable): One of the comprising tracks in the presentation. Can be video, audio, or text. The following attributes are supported:

- src: Specifies the file in which the track resides. This attribute is optional. For all Live Smooth Streaming manifests, the **src** attribute SHOULD be set to "Streams".

- systemBitrate: Specifies the bit rate of the track; this attribute MUST be present.

SMILParam (variable): Specifies a single parameter for a given **SMILTrack**. The following attributes are supported:

- name: Specifies the name of the parameter; this attribute MUST be present.

- value: Specifies the parameter's value; this attribute MUST be present.

- valuetype: Specifies "type" of value attribute; this attribute MUST be present. The **valuetype** attribute MUST be set to "data".

The valid range and semantic meaning of the name and value attributes varies as described below, depending on the value of the name attribute:

- <param name = "trackID">: The value attribute specifies the ID of the track containing the video, audio, or textstream element described. This attribute MUST be present. This value correlates to the **track_ID** field in the tfhd and trak boxes for the track.

- <param name = "trackName">: Specifies the client-facing name of the track, and appears in the Fragments() noun as part of the URL. This attribute is optional. All the tracks that have the same value for systemBitrate and attributes MUST have distinct track names. If this parameter is omitted, the following default track names are assigned by IIS:

  - For video tracks: value="video"

  - For audio tracks: value="audio"

- For textstream tracks: value="textstream"

- <param name = "manifestOutput">: Specifies that all sample data for this track is accumulated into the manifest, indicating that any sample data will be made available immediately to the client. This attribute is optional. The valid value for this parameter is TRUE or FALSE. Omitting this parameter indicates that sample data for the track is not to be accumulated in the manifest.

- <param name = "parentTrackName">: Identifies the track as being a sparse track and also specifies the name of its **parent track**. This attribute is optional. If this track is a control track, its data is downloaded transparently with the data from the parent track. Valid values for this parameter are trackName parameters for other tracks in the presentation. The tracks referenced by parentTrackName MUST NOT be control tracks, and MUST NOT be sparse.

- <param name = "{namepace}_{custom attribute}">: Specifies additional extended attributes for the track that can be used to discriminate between tracks having the same track name and bit rate. This attribute is optional.

- <param name = "timescale">: Specifies the timescale for this track, as the number of units that pass in one second. This attribute is optional. If this parameter is not present, the default value used is 10,000,000, which maps to increments of 100ns.

- <param name = "trackType">: Specifies the type of the track when a ref tag is used to describe it. This attribute is optional. This attribute is required when the track is specified using a ref element. Supported values are:

  - binary: The track contains binary data that is not audio, video, or text (for example, images).

- <param name = "Subtype">: Specific information that can be used by the client to identify characteristics of the track. This attribute is optional.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
SMIL = "<" SMILMediaElementName SMILMediaNamespace ">" S?
        SMILLiveHead S SMILLiveBody S?
       "</" SMILMediaElementName ">"
SMILMediaElementName = "smil"
SMILMediaNamespace = "xmlns" Eq DQ "http://www.w3.org/2001/SMIL20/Language" DQ
SMILLiveHead = "<head>" S SMILLiveMeta S "</head>"
SMILLiveMeta = "<meta " S "name" Eq DQ IDENTIFIER DQ S "content" Eq DQ IDENTIFIER DQ S?
               "/>"
SMILLiveBody = "<body>" S "<switch>" S SMILTracks S "</switch>" S "</body>"
SMILTracks = 1*(SMILVideoTrack / SMILAudioTrack / SMILTexttrack)
SMILVideoTrack = ="<video S SMILTrackAttributes S ">" S 1*SMILParam S "</video>"
SMILAudioTrack ="<audio S SMILTrackAttributes S ">" S 1*SMILParam S "</audio>"
SMILTextTrack = ="<textstream S SMILTrackAttributes S ">" S 1*SMILParam S "</textstream>"
SMILTrackAttributes = "src" S Eq S DQ "Streams" DQ S "systemBitrate" S Eq S DQ
                       1*DIGIT DQ S ">"
SMILParam = "<param" S "name" Eq DQ IDENTIFIER DQ S "value" Eq DQ IDENTIFIER DQ S?
            "valuetype" Eq DQ "data" DQ S "/>"
```

2.2.7.4 MoovBox

The **MoovBox** field is as described in [ISO/IEC-14496-12].

2.2.7.5 Fragment

The fragment is comprised of the MoofBox (section 2.2.4.1) and the MdatBox (section 2.2.4.8).

In order to handle live streams, the server requires a **TrackFragmentExtendedHeader** field inside the TrafBox (section 2.2.4.3).

2.2.7.5.1 Track Fragment Extended Header

The **TrackFragmentExtendedHeader** field and related fields specify the fragment's duration and absolute starting offset in timescale increments for the track. These fields MUST be present for every fragment in a live stream originating from an encoder and SHOULD be omitted otherwise.

Version (variable): Limited to 1 or 0 signifying 64-bit or 32-bit times respectively.

FragTime (variable): Specifies the absolute time of the fragment's first sample in timescale increments. Must be a 32-bit integer if **Version** = 0 and a 64-bit integer otherwise.

FragDuration (variable): Specifies the duration of the entire fragment in timescale increments. Must be a 32-bit integer if **Version** = 0 and a 64-bit integer otherwise.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
TrackFragmentExtendedHeader = Version FragTime FragDuration TFEHFlags
Version = STRING_UINT8
FragTime = STRING_UINT32 / STRING_UINT64
FragDuration = STRING_UINT32 / STRING_UINT64
TFEHFlags = 24*24 RESERVED_BIT
```

2.2.8 Server-to-Server Ingest

The **ServerIngest** and related fields contain data required for a server when requesting a broadcast and its related streams from another server.

ServerIngestRequest (variable): The URI [RFC2396] to which the **ServerIngest** request is sent.

EventID (variable): An optional identifier that enables the reuse of URLs, as specified in section 2.2.7.

The syntax of the fields defined in this section, specified in ABNF [RFC5234], is as follows:

```
ServerToServerRequest= Protocol "://" BroadcastURL Identifier
Protocol = "http" / "https"
BroadcastURL = ServerAddress "/" PresentationPath "/streammanifest"
ServerAddress = URISAFE_IDENTIFIER
PresentationPath = URISAFE_IDENTIFIER ".isml"
```

The response sent by a server receiving a **ServerIngestRequest** resembles a **LiveIngestMessage**, as specified in sections 2.2.7.1 through 2.2.7.5.

3   Protocol Details

3.1   Client Details

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 document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The client acts in accordance with the following model:
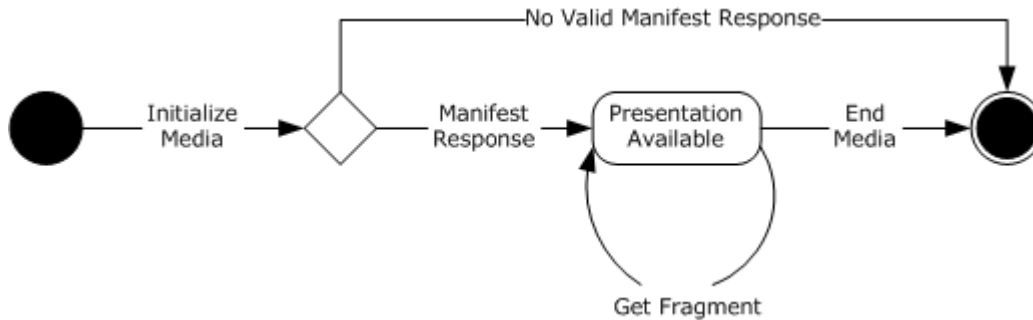
Figure 3: Client state machine diagram

The main data elements that are required by any implementation are:

- Presentation Description: A hierarchical data element that encapsulates metadata from the presentation, as specified in section 3.1.1.1.

- Fragment Description: Metadata and samples for a single fragment, as specified in section 3.1.1.2.

- Active Presentation: An instance of the presentation Description data element. This data element is maintained as state by the client.

- Presentation Available: A flag that indicates whether the Active Presentation has been successfully initialized. This data element is maintained as state by the client.

- Sparse Stream Pointer Header: A string that contains the name of the HTTP header used to carry the sparse stream Pointer message, specified in section 2.2.5.

3.1.1.1   Presentation Description

The Presentation Description data element encapsulates all metadata for the presentation.

Presentation Metadata: A set of metadata that is common to all streams in the presentation. Presentation Metadata comprises the following fields, specified in section 2.2.2.1:

- **MajorVersion**

- **MinorVersion**

- **TimeScale**

- **Duration**

- **IsLive**

- **LookaheadCount**

- **DVRWindowLength**

Stream Collection: A collection of Stream Description data elements, as specified in section 3.1.1.1.2.

Protection Description: A collection of Protection System Metadata Description data elements, as specified in section 3.1.1.1.1.

3.1.1.1.1   Protection System Metadata Description

The Protection System Metadata Description data element encapsulates metadata specific to a single Content Protection System.

Protection Header Description: Content protection metadata that pertains to a single Content Protection System. Protection Header Description comprises the following fields, specified in section 2.2.2.2:

- **SystemID**

- **ProtectionHeaderContent**

3.1.1.1.2   Stream Description

The Stream Description data element encapsulates all metadata specific to a single stream.

Stream Metadata: A set of metadata that is common to all tracks for the stream. Stream Metadata comprises the following fields, specified in section 2.2.2.3:

- **StreamTimeScale**

- **Type**

- **Name**

- **NumberOfFragments**

- **NumberOfTracks**

- **Subtype**

- **Url**

- **StreamMaxWidth**

- **StreamMaxHeight**

- **DisplayWidth**

- **DisplayHeight**

Track Collection: A collection of Track Description data elements, as specified in section 3.1.1.1.2.1.

Fragment Reference Collection: An ordered collection of Fragment Reference Description data elements, as specified in section 3.1.1.1.3.

Sparse Stream Flag: A Boolean flag that specifies the stream is a sparse stream.

Parent Stream: A reference to the Stream Description data element for this stream's Parent Stream. This data element SHOULD NOT be set unless the stream is a sparse stream.

Last Downloaded Fragment: A 64-bit unsigned integer that holds a reference to the Last Known Fragment for a sparse stream. This data element SHOULD NOT be set unless the stream is a sparse stream.

3.1.1.1.2.1 Track Description

The Track Description data element encapsulates all metadata specific to a single track.

Track Metadata: A set of metadata that is common to all fragments for the track. Track Metadata comprises the following fields, specified in section 2.2.2.5:

- **Index**

- **Bitrate**

- **MaxWidth**

- **MaxHeight**

- **SamplingRate**

- **AudioTag**

- **BitsPerSample**

- **PacketSize**

- **CodecPrivateData**

- **NALUnitLengthField**

Custom Attributes Collection: A collection of Custom Attribute Description data elements, as specified in section 3.1.1.1.2.1.1.

3.1.1.1.2.1.1 Custom Attribute Description

The Custom Attribute Description data element encapsulates a key/value pair that disambiguates tracks.

Key Value Pair: A single key/value pair. Key Value Pair comprises the following fields, specified in section 2.2.2.5.1:

- **CustomAttributeName**

- **CustomAttributeValue**

3.1.1.1.3 Fragment Reference Description

The Fragment Reference Description data element encapsulates metadata needed to identify a fragment in the stream, and create a corresponding Fragment Request message.

Fragment Reference Metadata: A set of metadata that describes a set of related fragments in all tracks for the stream. Fragment Reference Metadata comprises a collection of Track-Specific Fragment Reference Description data elements, specified in section 3.1.1.1.3.1, and the following fields, specified in section 2.2.2.6:

- **FragmentNumber**

- **FragmentDuration**

- **FragmentTime**

Track-Specific Fragment Reference Collection: A collection of Track-Specific Fragment Reference Description data elements, as specified in section 3.1.1.1.3.1.

3.1.1.1.3.1 Track-Specific Fragment Reference Description

The Fragment Reference Description data element encapsulates metadata needed to identify a fragment in the stream, and create a corresponding Fragment Request message.

Track-Specific Fragment Reference Metadata: A set of metadata that describes a set of related fragments in all tracks for the stream. Fragment Reference Metadata comprises the following fields, specified in section 2.2.2.6:

- **TrackFragmentIndexAttribute**

- **ManifestOutputSample**

3.1.1.2 Fragment Description

The Fragment Description data element encapsulates metadata and sample data for a single fragment.

Fragment Metadata: A set of metadata that is common to all samples in the fragment. Fragment Metadata comprises the following fields:

- **SequenceNumber**, specified in section 2.2.4.2

- **DefaultSampleDuration**, specified in section 2.2.4.4

- **DefaultSampleSize**, specified in section 2.2.4.4

- **DefaultSampleFlags**, specified in section 2.2.4.4

- **FirstSampleFlags**, specified in section 2.2.4.5

- **VendorExtensionUUIDBox**, specified in section 2.2.4.9

Sample Collection: A collection of Sample Description data elements, as specified in section 3.1.1.2.1.

3.1.1.2.1 Sample Description

The Sample Description data element encapsulates the metadata and data for a single sample.

Sample Metadata: A set of Attributes that pertain to the Sample. Sample Metadata comprises the following fields:

- **SampleDuration**, specified in section 2.2.4.5

- **TrunBoxSampleFlags**, specified in section 2.2.4.5

- **SampleSize**, specified in section 2.2.4.5

- **SampleCompositionTimeOffset**, specified in section 2.2.4.5

- **SampleData**, specified in section 2.2.4.6

3.1.2 Timers

None.

3.1.3 Initialization

Initialization of the client is triggered by Open Presentation event, specified in section 3.1.4.1. At Initialization, the Presentation Available flag is set to false.

The Sparse Stream Pointer Header is initialized from configuration above the IIS Smooth Streaming Transport Protocol layer. The configured value on the client MUST match the configured value on the server for interoperability. <1>

3.1.4 Higher-Layer Triggered Events

The client is driven by a higher-layer implementation that **decodes** samples for playback to the end user and uses the state of playback and end user interaction to drive Fragment Requests by the client. The following events trigger specific behavior on the client.

- Open Presentation, specified in section 3.1.4.1

- Get Fragment, specified in section 3.1.4.2

- Close Presentation, specified in section 3.1.4.3

3.1.4.1 Open Presentation

The Open Presentation event is used at the start of a viewing session. This event has no effect if the value of the Presentation Available flag is true.

The higher-layer implementation provides the following data element:

- Presentation URI: A string whose syntax matches the syntax of the **PresentationURI** field, specified in section 2.2.1.

When the Open Presentation event is triggered, the client sends a Manifest Request message to the server. Creation of the Manifest Request message is subject to the following rules:

- The value of the **PresentationURI** field in the Fragment Request is set to the value of the presentation URI data element.

If the processing of the Manifest Request, as specified in section 3.1.5.1, yields a presentation Description data element, the client MUST perform the following operations:

- Set the Presentation Available flag to true.

- Return the Presentation Description data element to the higher-layer implementation.

3.1.4.2 Get Fragment

The Get Fragment event is used during the course of the viewing session. This event has no effect when the value of the Presentation Available flag is false.

The higher-layer implementation provides the following data elements:

- Presentation URI: A string whose syntax matches the syntax of the **PresentationURI** field, specified in section 2.2.1.

- Request Stream: A Stream Description data element for the fragment to Request.

- Request Track: A Track Description data element for the fragment to Request.

- Request Fragment: A Fragment Reference Description data element for the fragment to Request.

When the Get Fragment event is triggered, the client sends a Fragment Request message to the server. Creation of the Fragment Request message is subject to the following rules:

- The value of the **PresentationURI** field in the Fragment Request is set to the value of the Presentation URI data element.

- The value of the **BitratePredicate** field in the Fragment Request is set to the value of the Bitrate field in the Request Track data element.

- One instance of the **CustomAttributesPredicate** field is created per instance of the Custom Attribute Description data element in the Request Track data element.

- The value of the **CustomAttributeKey** field of each **CustomAttributesPredicate** field is set to the value of the **CustomAttributeName** field in the corresponding CustomAttributesElement.

- The value of the **CustomAttributeValue** field of each **CustomAttributesPredicate** field is set to the value of the **CustomAttributeValue** field in the corresponding CustomAttributesElement.

- The value of the **StreamName** field in the Fragment Request is set to the value of the Name field in the Stream Description data element.

- The value of the Time field in the Fragment Request is set to the value of the **FragmentTime** field in the Request Fragment data element.

If the processing of the Fragment Response, as specified in section 3.1.5.2, yields a Fragment Description data element, the client MUST return the data element to the higher-layer implementation.

No state change is effected when the Get Fragment event is triggered.

3.1.4.3 Close Presentation

The Close Presentation event is used at the end of a viewing session. This event has no effect if the value of the Presentation Available flag is false.

When the Close Presentation event is triggered, the client sets the Presentation Available flag to false and enters the Final state.

3.1.5 Processing Events and Sequencing Rules

The following event processing and sequencing rules apply:

- Manifest Request and Manifest Response, as specified in section 3.1.5.1

- Fragment Request and Fragment Response, as specified in section 3.1.5.2

- The expected Response from the server to a Fragment Request message is a Fragment Response message. If the Response received to a Fragment Request message contains a message-body [RFC2616] but is not a valid Fragment Response, the client SHOULD return the error to the higher layer.

- The expected Response from the server to a Manifest Request Message is a Manifest Response message. If the Response received is not a valid Manifest Response message, the client MUST enter the Final state.

3.1.5.1 Manifest Request and Manifest Response

When a Manifest Request is sent to the server, the client MUST wait for a Manifest Response message to arrive. If the underlying transport returns an error, the client MUST enter the Final state.

If the underlying transport returns a Response that adheres to the syntax of the Fragment Response message, the message is processed to yield a Presentation Description, subject to the following processing rules:

- The Presentation Metadata data element is populated using data in the **SmoothStreamingMedia** field, as specified in section 2.2.2.1, subject to the field mapping rules specified in section 3.1.1.1.

- The Protection Description data element is populated using data in the **ProtectionElement** field, as specified in section 2.2.2.2, subject to the field mapping rules specified in section 3.1.1.1.1.

- The Stream Collection data element is populated by creating one Stream Description data element per instance of the **StreamElement** field, specified in section 2.2.2.3.

- Each Stream Description data element is populated using data in the corresponding **StreamElement** field, subject to the field mapping rules specified in section 3.1.1.1.2.

- If the Stream Description data element's Parent Track field is set, its Sparse Stream Flag is set to true. Otherwise the Sparse Stream Flag is set to false.

- The Track Collection data element of each Stream Description data element is populated by creating one Track Description data element per instance of the **TrackElement** field, specified in section 2.2.2.5, in the corresponding **StreamElement** field.

- Each Track Description data element is populated using data in the corresponding **TrackElement** field, subject to the field mapping rules specified in section 3.1.1.1.2.1.

- The Custom Attributes Collection data element of each Track Description data element is populated by creating one Custom Attribute Description data element per instance of the **CustomAttributesElement** field, specified in section 2.2.2.5.1, in the corresponding **TrackElement** field.

- Each Custom Attribute Description data element is populated using data in the corresponding **CustomAttributesElement** field, subject to the field mapping rules specified in section 2.2.2.5.1.

- The Fragment Reference Collection data element of each Stream Description data element is populated by creating one Fragment Reference Description data element per instance of the **StreamFragmentElement** field, specified in section 2.2.2.6, in the corresponding **StreamElement** field.

- Each Fragment Reference Description data element is populated using data in the corresponding **StreamFragmentElement** field, subject to the field mapping rules specified in section 3.1.1.1.3.

- The Fragment Reference Collection data element of each Stream Description data element is populated by creating one Fragment Reference Description data element per instance of the **StreamFragmentElement** field, specified in section 2.2.2.6, in the corresponding **StreamElement** field.

- Each Fragment Reference Description data element is populated using data in the corresponding **StreamFragmentElement** field, subject to the field mapping rules specified in section 3.1.1.1.3.

- The Track-Specific Fragment Reference Collection data element of each Fragment Reference Description data element is populated by creating one Track-Specific Fragment Reference Description data element per instance of the **TrackFragmentElement** field, specified in section 2.2.2.6.1, in the corresponding **StreamFragmentElement** field.

- Each Track-Specific Fragment Reference Description data element is populated using data in the corresponding **StreamFragmentElement** field, subject to the field mapping rules specified in section 3.1.1.1.3.

After the population of the Presentation Description, the following rules MUST be applied:

- If the **StreamTimeScale** field for any given Stream Description data element is not set, the **StreamTimeScale** field for that Stream Description data element is set to the value of the Duration field of the Presentation Description data element.

- If the Duration field of the Presentation Metadata data element is equal to 0, the Duration field is set to the result of the following computation:

  - For each Stream Description data element, compute a Stream Duration value by summing the Fragment Duration fields of each Fragment Reference Description data element, and dividing by the value of the **StreamTimeScale** field for that Stream Description data element

  - Set the Duration field by multiplying the maximum of the set of computed Stream Duration values by the value of the **TimeScale** field in the Presentation Metadata data element

- If the Name field of the Stream Description data element is not set, the Name element is set to the value of the Type field. If, after this operation, the Name fields of all Stream Description data elements are not unique with respect to each other, the data is considered invalid, and the client SHOULD enter the Final state without yielding a Presentation Description data element.

- If, for any given Stream Description data element, the **StreamMaxWidth** field is not set, and the **Type** field is "video", the **StreamMaxWidth** field is set to the maximum of all **MaxWidth** fields in all Track Description data elements contained in the Stream Description data element.

- If, for any given Stream Description data element, the **StreamMaxHight** field is not set, and the **Type** field is "video", the **StreamMaxHeight** field is set to the maximum of all **MaxHeight** fields in all Track Description data elements contained in the Stream Description data element.

- In each Stream Description data element, the client MUST iterate through the Fragment Reference Collection in order and apply the following rules for each Fragment Reference Description data element:

  - If the current Fragment Reference Description data element is the last in the collection and the value of the **FragmentDuration** field is not set, the data is considered invalid, and the client MUST enter the Final state without yielding a Presentation Description data element.

  - If neither of the values of the **FragmentTime** and **FragmentDuration** fields for a single Fragment Reference Description is set, the data is considered invalid, and the client MUST enter the Final state without yielding a Presentation Description data element.

  - If the current Fragment Reference Description data element is the first in the collection and the value of the **FragmentTime** field in the collection is not set, the value of the **FragmentTime** field is set to 0.

  - If the value of the **FragmentTime** field in the current Fragment Reference Description data element is not set, the value of the **FragmentTime** field is set to the sum of the values of the **FragmentTime** and **FragmentDuration** fields of the preceding Fragment Reference Description data element.

  - If the value of the **FragmentDuration** field in the current Fragment Reference Description data element is not set, the value of the **FragmentDuration** field is set to the value obtained by subtracting the value of the **FragmentTime** field from the value of the **FragmentTime** field in the following Fragment Reference Description data element in the collection.

  - If the value of the **FragmentTime** field is greater than the value of the **FragmentTime** field in the following Fragment Reference Description data element in the collection, and the client MUST enter the Final state without yielding a Presentation Description data element.

If the underlying transport returns a Response that does not adhere to the syntax of the Manifest Response message, the client MUST enter the Final state without yielding a Presentation Description data element.

3.1.5.2   Fragment Request and Fragment Response

When a Fragment Request is sent to the server, the client MUST wait for a Fragment Response message to arrive. If the underlying transport returns an error, the client MUST enter the Final state. Processing makes use of the following variable:

Current Sample Start: The offset of the beginning of the current sample, in bytes. This value of Current Sample Start is initialized to 0.

Sparse Stream Notifications: A collection in which each entry contains two data points:

- Stream Description Reference: A reference to the Stream Description data element for which a new fragment is available.

- Timestamp: A 64-bit unsigned integer that represents the timestamp of the new fragment.

If the underlying transport returns a Response that adheres to the syntax of the fragment Response message, the message is processed to yield a Presentation Description, subject to the following processing rules:

- The Fragment Description data element using data in the **FragmentMetadata** field, as specified in section 2.2.4, subject to the field mapping rules specified in section 3.1.1.2.

- The Sample Collection data element is populated by creating Sample Description data element per instance of the TrunBoxPerSampleFields field, specified in section 2.2.4.5.

- Each Sample Description data element is populated using data in the **FragmentMetadata** field, as specified in section 2.2.4.5, subject to the field mapping rules specified in section 3.1.1.1.2.

After the population of the Fragment Description, the following rules MUST be applied:

- If the **FirstSampleFlags** field of the Fragment Description is not set, the value of this field is set to the value of the DefaultSampleFlags field.

- In each Fragment Description data element, the client MUST iterate through the Fragment Collection in order and apply the following rules for each Fragment Description data element:

  - If the current Sample Description data element is the first in the collection and the value of the **SampleFlags** field is not set, the value of the **SampleFlags** is set to the value of the **FirstSampleFlags** field in the Fragment Description data element.

  - If the value of the **SampleDuration** field of the current Sample Description data element is not set, the value is set to the value of the **DefaultSampleDuration** field in the Fragment Description data element.

  - If the value of the **SampleSize** field of the current Sample Description data element is not set, the value is set to the value of the **DefaultSampleSize** field in the Fragment Description data element.

If the underlying transport returns a Response that does not adhere to the syntax of the Fragment Response message, the client MUST enter the Final state without yielding a Fragment Description data element and stop further processing of these rules.

If the Sparse Track Flag in the Stream Description data element used to generate the Fragment Request is true, client MUST update the Last Downloaded Fragment field of the Stream Description by setting the Last Downloaded Timestamp value to match the Timestamp of the Fragment Request.

The client MUST attempt to match the HTTP Header [RFC2616] whose name matches the value of the Sparse Stream Pointer Header field, specified in section 3.1.1, to the syntax of the Sparse Stream Pointer message, specified in section 2.2.5. If a match is found, the following additional processing is performed:

- For each **SparseStreamFragment** field in the Sparse Stream Pointer message, the client MUST perform the following operations:

  - Search the Stream Collection data element in the Active Presentation data element to locate the Stream Description whose Name field matches the **SparseStreamName** field.

    - If no match can be found or if the Sparse Stream Flag of the matching Stream Description data element is false, the client MUST enter the Final state without yielding a Fragment Description data element.

    - Compare the value of the **SparseStreamTimeStamp** field to the Last Downloaded Timestamp field of the Stream Description data element.

  - If the Last Downloaded Timestamp field is not set, or if the value of the **SparseStreamTimeStamp** is greater than the Last Downloaded Timestamp field, add an entry to the Sparse Stream Notifications collection for which the Stream Description Reference field is set to the matching Stream Description, and the Timestamp field is set to the value of the **SparseStreamTimeStamp** field.

If the Sparse Stream Notifications collection is not empty, the client yields this collection to the higher-layer in addition to the Fragment Description.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

The server does not maintain state and treats all arriving messages independently.

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 document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The server uses the same conceptual model as the client, specified in section 3.1.1.

3.2.2 Timers

None.

3.2.3 Initialization

There is no initialization required for the IIS Smooth Streaming Transport Protocol layer. Successful initialization of the underlying transport (HTTP) is a prerequisite for successful operation of the server.

The Sparse Stream Pointer Header is initialized from configuration above the IIS Smooth Streaming Transport Protocol layer. The configured value on the client MUST match configured value on the server for interoperability. <2>

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Processing Events and Sequencing Rules

The following event processing and sequencing rules apply:

- When a valid Manifest Request message arrives, the server MUST respond with a Manifest Response message.

- When a valid Fragment Request message arrives, the server MUST respond with a Fragment Response message. Depending on the value of the **FragmentsNoun** field in the Fragment Request (see section 2.2.3), the form of the **FragmentResponse** field, specified in section 2.2.4, varies according to the following rules:

- If the **FragmentsNoun** field is a FragmentsNounFullResponse, the **FragmentResponse** field MUST be a FragmentFullResponse.

- If the **FragmentsNoun** field is a FragmentsNounMetadataOnly, the **FragmentResponse** field MUST be a FragmentMetadataResponse.

- If the **FragmentsNoun** field is a FragmentsNounDataOnly, the **FragmentResponse** field MUST be a FragmentDataResponse.

- If the **FragmentsNoun** field is a FragmentsNounIndependentOnly, the **FragmentResponse** field MUST be a FragmentFullResponse, and all Samples in the **FragmentResponse** MUST be independently decodable, as defined in [ISO/IEC-14496-12].

The following special processing rules apply when generating a Fragment Response in a live presentation:

- Requested Streams Collection: The server computes the set of all Stream Descriptions which pertain to the fragments referenced in the incoming Fragment Request message.

- For each entry in the Requested Streams Collection, perform the following processing:

  - If the selected item is not a child stream of another stream, do the following:

    - Child Streams Collection: The server computes the set of all Stream Descriptions for which the Parent Streams field references the selected item in the Requested Streams Collection.

    - From each item in the Child Streams Collection, generate a **SparseStreamFragment** field, specified in section 2.2.5 by setting the **SparseStreamName** field to the Name field of the Stream Description data element, and setting the **SparseStreamTimestamp** field to the greatest timestamp of any fragment for the corresponding stream that is available from the server, but that is not later than the timestamp of the corresponding requested fragment.

    - Generate a **SparseStreamSet** field from the **SparseStreamFragment** fields generated in the previous step.

  - If the selected item is a child stream of another stream, and the requested fragment is not the first fragment in the track, do the following:

    - Generate a **SparseStreamFragment** field, specified in section 2.2.5, by setting **SparseStreamName** field to the **Name** field of the selected item. Set the **SparseStreamTimestamp** field to the timestamp of the preceding fragment in the track.

    - Generate a **SparseStreamSet** field containing a single **SparseStreamFragment** field, as specified in the preceding step.

  - If any **SparseStreamSet** fields are generated as a result of the preceding steps, generate a **SparseStreamPointer** field according to the following rules:

    - If the processing rules specified by HTTP [RFC2616] result in an HTTP Header whose name matches the value of the Sparse Stream Pointer Header field, the data becomes the **HeaderData** field. Otherwise, the **HeaderData** and DELIMITER fields are omitted.

    - The remainder of the **SparseStreamPointer** field is generate from the **SparseStreamSet** fields.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

3.3 Live Encoder Details

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that a live encoding implementation maintains in order to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The live encoder acts in accordance with the following model.
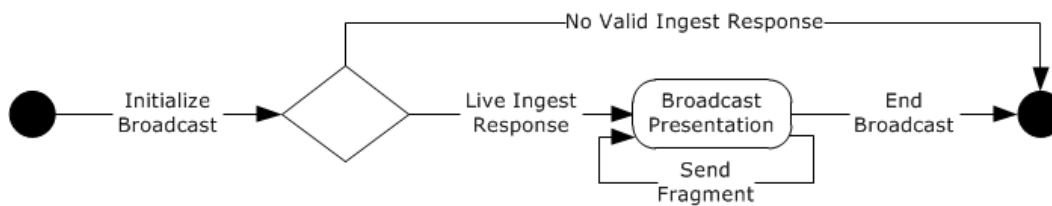
Figure 4: Live Encoder state machine diagram

The main data elements that are required by any implementation are:

Live Server Manifest Box: Includes a live server manifest that describes tracks and metadata in an MPEG-4 ([MPEG4-RA]) container, as specified in [ISO/IEC-14496-12].

Stream Manifest Box (optional): Responsible for enumerating all streams, ensuring that clients wait for all the streams before beginning a broadcast, as specified in section 2.2.7.2.

File Type Box: Specifies the sub-type and intended use of the MPEG-4 file, along with high-level attributes as specified in section 2.2.7.1.

Movie: This is the media file in fragmented-MPEG-4 format, as specified in [ISO/IEC-14496-12].

Fragment: This is the media fragment, as specified in section 2.2.7.5.

3.3.2 Timers

None.

3.3.3 Initialization

An HTTP POST request from an encoder with an empty body (zero content length) using the URL as specified in the **LiveIngestRequest** field (2.2.7).

The server does not send back a response until the entire POST is received. This allows for error detection before all the data is ready, which is necessary in long live streams.

3.3.4 Higher-Layer Triggered Events

The **LiveIngest** point (section 2.2.7) is driven by a higher-layer implementation that ingests streams for broadcast to the end user. The following events trigger specific behavior on the **LiveIngest** point:

- Open Presentation, specified by section 3.3.4.1.

- End Ingest, specified by section 3.3.4.2.

3.3.4.1 Start Stream

After a 200 HTTP response is received from the server following initialization, the encoder SHOULD initiate a new, long-running HTTP POST request. The payload of the request MUST be the fragment MPEG-4 ([MPEG4-RA]) stream, starting from the header boxes and followed by the fragments. The request MUST follow the specifications in Transport (section 2.1).

If a **StreamManifest** (section 2.2.7.2) is provided by the encoder, the server waits until all the enumerated streams are received before starting the broadcast. If all the streams are not received within the defined duration time, some of the earlier streams' data MAY be discarded and lost before initialization.

3.3.4.2 Stop Stream

A started stream can only be stopped by an End-Of-Stream (EOS) signal from the encoder, a manual shutdown command, or an internal error. To properly signify the end of a live broadcast, the encoder SHOULD send an empty MfraBox as specified by [ISO/IEC-14496-12] with no embedded sample entries in the Tfra box and no MfroBox following, as specified by [ISO/IEC-14496-12].

Lastly, the long-running POST request SHOULD be properly terminated by closing the HTTP connection as specified in the HTTP protocol [RFC2616].

3.3.5 Processing Events and Sequencing Rules

None.

3.3.6 Timer Events

None.

3.3.7 Other Local Events

None.

4 Protocol Examples

4.1 Manifest Response

The following is an example of a Manifest Response (section 2.2.2) message:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SmoothStreamingMedia MajorVersion="2" MinorVersion="0"
      Duration="2300000000" TimeScale="10000000">
   <Protection>
     <ProtectionHeader SystemID="{9A04F079-9840-4286-AB92E65BE0885F95}">
        <!-- Base 64-Encoded data omitted for clarity -->
     </ProtectionHeader>
   </Protection>

   <StreamIndex
      Type = "video"
      Chunks = "115"
      QualityLevels = "2"
      MaxWidth = "720"
      MaxHeight = "480"
      TimeScale="10000000"
      Url =
       "QualityLevels({bitrate},{CustomAttributes})/Fragments(video={start_time})"
      Name = "video"
   >
      <QualityLevel Index="0" Bitrate="1536000" FourCC="WVC1"
         MaxWidth="720" MaxHeight="480"
         CodecPrivateData = "270000010FCBEE1670EF8A16783BF180C9089CC4AFA11C0000010E1207F840"
         >
         <CustomAttributes>
            <Attribute Name = "Compatibility" Value = "Desktop" />
         </CustomAttributes>
      </QualityLevel>


      <QualityLevel Index="5" Bitrate="307200" FourCC="WVC1"
         MaxWidth="720" MaxHeight="480"
         CodecPrivateData = "270000010FCBEE1670EF8A16783BF180C9089CC4AFA11C0000010E1207F840">
         <CustomAttributes>
            <Attribute Name = "Compatibility" Value = "Handheld" />
         </CustomAttributes>
      </QualityLevel>


      <c t = "0" d = "19680000" />
      <c n = "1" t = "19680000" d="8980000" />

   </StreamIndex>
</SmoothStreamingMedia>
```

4.2 Fragment Request

The following is an example of a Fragment Request (section 2.2.3) message. It follows the Manifest Response (section 4.1) message example, in compliance with the sequencing rules specified in section 3.1.5:

```
/PubPoint.ism/QualityLevels(307200,Compatibility=Handheld)/Fragments(video=1968000)
```

4.3 Live Ingest Request

The following is an example of a **LiveIngestRequest** as specified in section 2.2.7.

```
http://Server/mybroadcast.isml/streams(720p)
```

4.4 Stream Manifest

The following is an example of a **StreamManifest** as specified in section 2.2.7.2.1.

```
<?xml version="1.0" encoding="utf-16"?>
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
   <body>
      <par>
         <ref src = "Events(exampleEvent)/Streams(1080p)" />
         <ref src = "Events(exampleEvent)/Streams(720p)" />
         <ref src = "Events(exampleEvent)/Streams(480p)" />
      </par>
   </body>
</smil>
```

4.5 Live Server Manifest

The following is an example of a **LiveServerManifest** as specified in section 2.2.7.3.

```
<?xml version="1.0" encoding="utf-16"?>
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
   <head>
     <meta name="Meta Data" content="Common meta data" />
   </head>
   <body>
      <switch>
         <video src = "Stream" systemBitrate="1450000">
            <param name="trackID" value="2" valuetype="data" />
            <param name="FourCC" value="WVC1" valuetype="data" />
            <param name="MaxWidth" value="640" valuetype="data" />
   <param name="MaxHeight" value="480" valuetype="data" />
            <param name="CodecPrivateData"
      value="250000010FD37E27F1678A27F (no line break here)
            859E80490824C4ADF5DC00000010E5A67F840"
      valuetype="data" />
         </video>
         <video src = "Stream" systemBitrate="1050000">
            <param name="FourCC" value="WVC1" valuetype="data" />
   <param name="trackID" value="2" valuetype="data" />
            <param name="MaxWidth" value="640" valuetype="data" />
```

```
              <param name="MaxHeight" value="480" valuetype="data" />
              <param name="CodecPrivateData"
    value="250000010FD37E27F1678A27F (no line break here)
                          859E80490824C4ADF5DC00000010E5A67F840"
    valuetype="data" />
          </video>
          <audio src = "Stream" systemBitrate="94208">
              <param name="trackID" value="1" valuetype="data" />
      <param name="Subtype" value="WMAPRO" valuetype="data" />
              <param name="CodecPrivateData"
                value="6101020044AC0000853E00009 (no line break here)
                      D0B10000A00008800000F0000000000"
      valuetype="data" />
      <param name="SamplingRate" value = "48000" valuetype="data" />
      <param name="BitsPerSample" value = "16" valuetype="data" />
      <param name="PacketSize" value = "1115" valuetype="data" />
          </audio>
      </switch>
    </body>
  </smil>
```

4.6 Server Ingest Request

The following is an example of a **ServerIngestRequest** as specified in section 2.2.8.

Note In this case it is assumed that the original broadcast included the **EventID** param.

```
http://Server/mybroadcast.isml/Events(myEvent)/streams(720p)
```

5   Security

5.1   Security Considerations for Implementers

If the content transported using this protocol has high commercial value, a Content Protection System should be used to prevent unauthorized use of the content. The **ProtectionElement** can be used to carry metadata related to the use of a Content Protection System.

5.2   Index of Security Parameters

Security parameter	Section
ProtectionElement	2.2.2.2

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows Server 2008 operating system

- Windows Server 2008 R2 operating system

- Windows Server 2012 operating system

- Windows Server 2012 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 3.1.3: The Windows implementation, "IIS Media Services 3.0", uses "Content-Type" as the default value of this field to maximize compatibility with existing web browser-based HTTP APIs.

<2> Section 3.2.3: The Windows implementation, "IIS Media Services 3.0", uses "Content-Type" as the default value of this field to maximize compatibility with existing web browser-based HTTP APIs.

7   Change Tracking

This section identifies changes that were made to the [MS-SSTR] protocol document between the August 2013 and November 2013 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.

- An extensive rewrite, addition, or deletion of major portions of content.

- The removal of a document from the documentation set.

- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.

- Content updated.

- Content removed.

- New product behavior note added.

- Product behavior note updated.

- Product behavior note removed.

- New protocol syntax added.

- Protocol syntax updated.

- Protocol syntax removed.

- New content added due to protocol revision.

- Content updated due to protocol revision.

- Content removed due to protocol revision.

- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.

- Protocol syntax removed due to protocol revision.

- New content added for template compliance.

- Content updated for template compliance.

- Content removed for template compliance.

- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.

- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1 Introduction	Updated the introduction summary.	Y	Content updated.
1.2.1 Normative References	Added the references [MPEG4-RA] and [SMIL2.1].	Y	Content updated.
2.1 Transport	Added the constraints rules for the Live Ingest Request.	Y	Content updated.
2.2 Message Syntax	Added the Live Ingest Request message and the STRING_UINT8 field definition.	Y	Content updated.
2.2 Message Syntax	Updated the fields TRUE, FALSE, HEXCODED_BYTE, and URL_ENCODED_CHAR and removed the field HEXDIGIT.	Y	Content updated.
2.2.2.1 SmoothStreamingMedia	69438 Updated that the MinorVersion field MUST be set to 0 or 2.	Y	Content updated.
2.2.2.1 SmoothStreamingMedia	Updated the ABNF syntax.	Y	Content updated.
2.2.2.1 SmoothStreamingMedia	Updated the IsLive field and the ABNF syntax.	Y	Content updated.
2.2.2.2 ProtectionElement	Updated the ABNF syntax.	Y	Content updated.

Release: Friday, October 25, 2013

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.2.3 StreamElement	Updated the ABNF syntax.	Y	Content updated.
2.2.2.3 StreamElement	Updated the ManifestOutput field and the ABNF syntax.	Y	Content updated.
2.2.2.5 TrackElement	Updated the ABNF syntax.	Y	Content updated.
2.2.2.5.1 CustomAttributesElement	Updated the ABNF syntax.	Y	Content updated.
2.2.2.6 StreamFragmentElement	69438 Added the FragmentRepeat field definition and syntax.	Y	Content updated.
2.2.2.6 StreamFragmentElement	Updated the ABNF syntax.	Y	Content updated.
2.2.2.6.1 TrackFragmentElement	Updated the ABNF syntax.	Y	Content updated.
2.2.2.6.1 TrackFragmentElement	Updated the ManifestOutputSample field.	Y	Content updated.
2.2.4.1 MoofBox	Updated the MoofBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.2 MfhdBox	Updated the MfhdBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.3 TrafBox	Updated the TrafBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.4 TfxdBox	Updated the TfxdBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.5 TfrfBox	Updated the TfrfBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.6 TfhdBox	Updated the TfhdBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.7 TrunBox	Updated the TrunBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.8 MdatBox	Updated the MoofBoxType field in the ABNF syntax.	Y	Content updated.
2.2.4.9 Fragment Response Common Fields	Updated the UUIDBoxType field in the ABNF syntax.	Y	Content updated.
2.2.7 Live Ingest	Added section.	Y	New content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
			added.
2.2.7.1 FileType	Added section.	Y	New content added.
2.2.7.2 StreamManifestBox	Added section.	Y	New content added.
2.2.7.2.1 StreamSMIL	Added section.	Y	New content added.
2.2.7.3 LiveServerManifestBox	Added section.	Y	New content added.
2.2.7.3.1 LiveSMIL	Added section.	Y	New content added.
2.2.7.4 MoovBox	Added section.	Y	New content added.
2.2.7.5 Fragment	Added section.	Y	New content added.
2.2.7.5.1 Track Fragment Extended Header	Added section.	Y	New content added.
2.2.8 Server-to-Server Ingest	Added section.	Y	New content added.
3.3.1 Abstract Data Model	Added section.	Y	New content added.
3.3.2 Timers	Added section.	Y	New content added.
3.3.3 Initialization	Added section.	Y	New content added.
3.3.4 Higher-Layer Triggered Events	Added section.	Y	New content added.
3.3.4.1	Added section.	Y	New

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
Start Stream			content added.
3.3.4.2 Stop Stream	Added section.	Y	New content added.
3.3.5 Processing Events and Sequencing Rules	Added section.	Y	New content added.
3.3.6 Timer Events	Added section.	Y	New content added.
3.3.7 Other Local Events	Added section.	Y	New content added.
4 Protocol Examples	Moved content to sections Manifest Response and Fragment Request.	Y	Content updated.
4.1 Manifest Response	Added section.	Y	New content added.
4.2 Fragment Request	Added section.	Y	New content added.
4.3 Live Ingest Request	Added section.	Y	New content added.
4.4 Stream Manifest	Added section.	Y	New content added.
4.5 Live Server Manifest	Added section.	Y	New content added.
4.6 Server Ingest Request	Added section.	Y	New content added.

8 Index

N

O

P

R

S

T

U

V

Release: Friday, October 25, 2013