INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND ASSOCIATED AUDIO

ISO/IEC JTC1/SC29/WG11/ N15993
February 2016, San Diego, US

Title: **Committee Draft for 23009-6: DASH with Server Push and WebSockets**
Source: **Systems**
Status: **Approved**
Authors: **Viswanathan (Vishy) Swaminathan (Adobe)**
Kevin Streeter (Adobe)
Imed Bouazizi (Samsung)
Franck Denoual (Canon)
Frédéric Mazé (Canon)

ISO/IEC JTC 1/SC 29 N

Date: 2014-07-11

ISO/IEC WD 23009-5

ISO/IEC JTC 1/SC 29/WG 11

Secretariat:

Information Technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND)

Élément introductif — Élément central — Partie 5: Titre de la partie

Contents

Page

.

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 23009-6 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second/third/... edition cancels and replaces the first/second/... edition (), [clause(s) / subclause(s) / table(s) / figure(s) / annex(es)] of which [has / have] been technically revised.

ISO/IEC 23009 consists of the following parts, under the general title *Information Technology — Dynamic adaptive streaming over HTTP (DASH)*:

- *Part 1: Media presentation description and segment formats*
- *Part 2: Conformance and reference software*
- *Part 3: Implementation guidelines*
- *Part 4: Segment encryption and authentication*
- *Part 5: Server and network assisted DASH (SAND)*
- *Part 6: DASH with Server Push and WebSockets*

.

Introduction

Dynamic Adaptive Streaming over HTTP (DASH) is intended to support a media-streaming model for delivery of media content in which control lies exclusively with the client.

This part of ISO/IEC 23009 specifies carriage of MPEG DASH media presentations over full duplex HTTP-compatible protocols, particularly HTTP/2 and WebSockets.

Information Technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND)

1 Scope

This part of ISO/IEC 23009 specifies carriage of MPEG-DASH media presentations over full duplex HTTP-compatible protocols, particularly HTTP/2 and WebSockets. This carriage takes advantage of the features these protocols support over HTTP/1.1 to improve delivery performance, while still maintaining backwards compatibility, particularly for the delivery of low latency live video.

2 Normative References

IEEE 1003.1-2008, *IEEE Standard for Information Technology – Portable Operating System Interface (POSIX)*, Base Specifications, Issue 7

IETF RFC 2616*, Hypertext Transfer Protocol – HTTP/1.1,* June 1999

IETF RFC 5234, *Augmented BNFr Syntax Specifications: ABNF*, January 2008

IETF RFC 6455, *The WebSocket Protocol,* December 2011

IETF RFC 6570, *URI Template*, March 2012

IETF RFC 7540*, Hypertext Transfer Protocol Version 2 (HTTP/2),* May 2015

3 Terms, definitions, symbols and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
full duplex HTTP
any protocol that is designed to be backward compatible with standard HTTP/1.1 (for example through HTTP's protocol upgrade mechanism) and that supports bidirectional communication initiated either by the client or by the server

3.1.2
HTTP/2
version 2 of the HTTP protocol, as defined by the IETF in RFC 7540

3.1.3
push
see "server push"

3.1.4
Push Acknowledgement (also Push Ack)
a response modifier, sent from a server to a client, which enables a server to state the push strategy used when processing a request

3.1.5
Push Directive
a request modifier, sent from a client to a server, which enables a client to express its expectations regarding the server's push strategy for processing a request

3.1.6
push strategy
a segment transmission strategy, that defines the ways in which segments may be pushed from a server to a client

3.1.7
server push (also push)
transmission of a segment from server to client based on a push strategy, as opposed to directly in response to a client request

3.1.8
WebSocket
the WebSocket protocol, as defined by the IETF in RFC 6455

3.2 Conventions

In this document data formats are described using the ABNF method as described in RFC 5234. A number of basic rules are used throughout the document:

STRING = 1* VCHAR

INTEGER = 1* DIGIT

FLOAT = INTEGER "." INTEGER / INTEGER

4 Introduction

The basic mechanisms of MPEG-DASH over HTTP/1.1 can be augmented by utilizing the new features and capabilities that are provided by the more recent Internet protocols such as HTTP/2 and WebSockets. While in details HTTP/2 and WebSocket are quite different, they both allow server-initiated and client-initiated transactions, data request cancelation, and multiplexing of multiple data responses. These capabilities can be used to reduce the transmission delay (latency) and to improve the responsiveness to server-initiated events in media presentations delivery.

The overall workflow of MPEG-DASH over these protocols is shown in Figure 1. The client and server first initiate a media channel, where the server can actively push data to the other (enabled by HTTP/2 server push or WebSocket messaging). The media channel is established via the HTTP/1.1 protocol upgrade mechanism. After the upgrade, the DASH client requests the media or the MPD from the server, with a URI and a push strategy. This strategy informs the server about how the client would like media delivery to occur (initiated

by the server or initiated by the client). Once the server receives the request, it responds with the requested data and initializes the push cycle as defined in the push strategy.

Error! Reference source not found. shows an example DASH session wherein the client requests the MPD first and then the media segments, with a push strategy. After receiving the requested MPD, the client starts requesting video segments from the server with the respective DASH segment URL and a segment push strategy. Then, the server responds with the requested video segment, followed by the push cycles as indicated by the segment push strategy. Typically, the client starts playing back the video after a minimum amount of data is received and then the aforementioned process repeats until the end of the media streaming session.

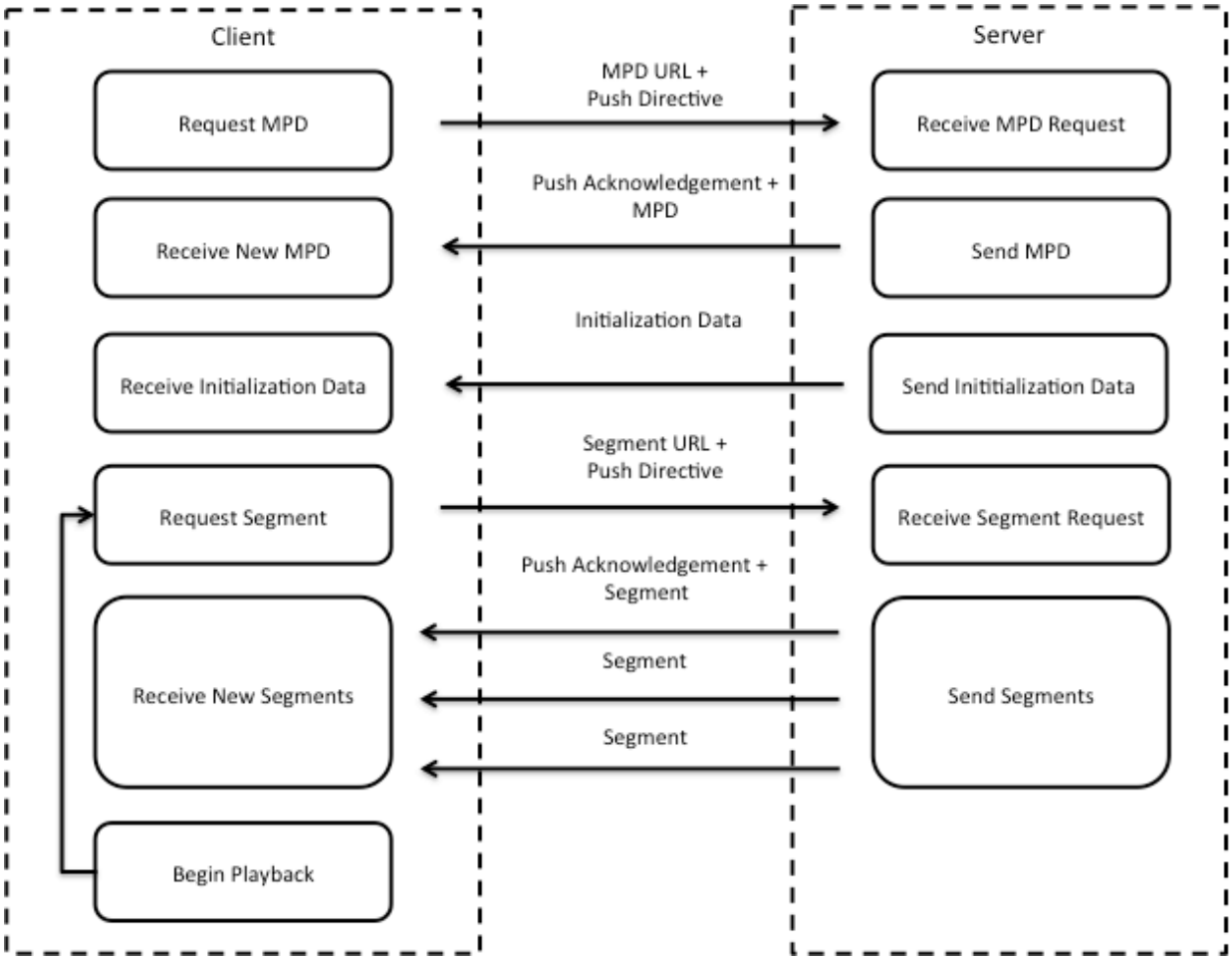

Figure 1 Overall flow of video streaming using server push

5 Specification Structure

This specification defines the signalling and message formats for driving the delivery of MPEG-DASH media presentations over full-duplex HTTP-compatible protocols. Details are provided for utilizing this signalling over the HTTP/2 (Section **Error! Reference source not found.**) and the WebSocket (Section 8) protocols.

A number of informational annexes are provided to demonstrate the use of the specified signalling and message formats to build streaming systems that take advantage of the full-duplex capabilities of the underlying transport protocol.

6 Definitions

6.1 Data Type Definitions

This section describes a number of primitive data types used to define the signalling over protocols addressed in this specification. Details for implementing these primitives for a given protocol may be found in the section of this specification defining that binding.

6.1.1 General

Table 1 — Definitions of primitive data types

Data Type	Base Type	Description
`BinaryObject`	N/A	An untyped binary object made up of 0 or more bytes
`Boolean`	N/A	A true or false value
`MPD`	MPD	An MPEG-DASH Media Presentation Description (MPD), as defined in ISO/IEC 23009-1
`Null`	N/A	An empty value
`PushDirective`	String	A directive describing the requested push strategy to be employed within the streaming session. See Table 2 for valid values for this type. See section 6.1.2.
`Segment`	Segment	An MPEG-DASH initialization or media segment, as defined in ISO/IEC 23009-1
`String`	N/A	A UTF-8 character string
`URI`	String	A Uniform Resource Identifier (URI), as defined in RFC 3986
`PushAck`	String	A response from the server acknowledging a push request. The PushAck contains the accepted values for the push strategy specified in the PushDirective. See Table 2 for valid values for this type. See section 6.1.3.
`URLTemplate`	String	A URL template and corresponding parameters that describes a set of URLs. See section 6.1.4.

6.1.2 PushDirective

A PushDirective signals the push strategy that a client would like the server to use for delivery of one or more future segments. A PushDirective has a type (described in Table 2) and, depending on the type, may have one or more additional parameters associated with it.

In general, a client may signal one or more PushDirectives for a single message. The server may select any one of the provided push strategies. This mechanism allows for clients to interoperate with servers that allow different push strategies, and for forward compatibility as the new types of push strategies are introduced.

The format of a `PushDirective` in the ABNF form is as follows:

PUSH_DIRECTIVE = PUSH_TYPE [";" PUSH_PARAMS]

PUSH_TYPE = DQUOTE STRING DQUOTE

PUSH_PARAMS = NUMBER / URL_TEMPLATE

where PUSH_PARAMS depends on the type of the Push Directive (See Table 2).

6.1.3 PushAck

A Push Acknowledgement (PushAck) is sent from the server to the client to indicate that the server intends to follow a given push strategy. Multiple Push Acknowledgments may be returned, indicating that multiple push strategies are in effect at once.

The format of the `PushAck` in the ABNF form is as follows:

PUSH_ACK = PUSH_TYPE [";" PUSH_PARAMS]

PUSH_TYPE = DQUOTE STRING DQUOTE

PUSH_PARAMS = NUMBER / URL_TEMPLATE

Where PUSH_PARAMS depends on the type of the Push Directive (See Table 2).

6.1.4 URLTemplate

<Editors' Note: The requirement for a template mechanism supporting macro expansion for URLs is based on the concern that simple lists of URLs may be very long, exhausting the available buffer space that a typical web server allocates for HTTP headers. We invite National Body comments on the validity of this concern, and on whether we need to be more specific about the maximum size of a header that can be supported.>

A `URLTemplate` describes a specific set of URLs via a template and the corresponding parameters required to expand the template. A client may use a template to explicitly signal the segments to be pushed during a push transaction. The string is formed as a list of individual URL templates, each of which may be parameterized to signal one or more URL values. When fully evaluated, the complete list of URLs describes the sequence of segments to be pushed within this push transaction.

The `URLTemplate` format is inspired by the "level 1" URI template scheme defined in IETF RFC 6570.

The above template mechanism ay be used to describe URLs contained in the MPEG-DASH MPD, whether they are formed using a SegmentTemplate or SegmentList. It is not possible to use URLTemplate to describe URLs formed via SegmentTemplate when they

use `$Time$` variable, unless the time value of each segment can be predicted or is described via SegmentTimeline, typically when @r is present and is not negative.

In addition, each parameter may be suffixed with an additional format tag aligned with the printf format tag as defined in IEEE 1003.1-2008 following this prototype:

%0[width]d

The width parameter is an unsigned integer that provides the minimum number of characters to be printed. If the value to be printed is shorter than this number, the result shall be padded with zeros. The value is not truncated even if the result is larger.

The `URLTemplate` string format ABNF follows:

URL_TEMPLATE =

URL_TEMPLATE_LIST

URL_TEMPLATE_LIST =

URL_TEMPLATE_LIST “;” TEMPLATE_ITEM / TEMPLATE_ITEM

TEMPLATE_ITEM =

TEMPLATE_ELEMENT “:” “{“ PARAMS “}” /

TEMPLATE_ELEMENT

TEMPLATE_ELEMENT =

CLAUSE_LITERAL CLAUSE_VAR CLAUSE_LITERAL /

CLAUSE_LITERAL

CLAUSE_VAR =

“{%0“ INTEGER “d}” /

“{}”

CLAUSE_LITERAL = STRING

PARAMS =

VALUE_LIST /

VALUE_RANGE

VALUE_LIST = VALUE_LIST “,” INTEGER / INTEGER

VALUE_RANGE = INTEGER “-“ INTEGER

Each template element is formed as a URL containing up to one macro for parameterization. This URL is relative to the segment being requested.

The {} parameter is used to specify a specified list or range of URLs that differ by segment number or timestamp, and is expanded using the provided value specifier. If no parameter is provided, the value specifier is optional. This makes it possible to provide a simple list of URLs.

The URL list will be generated from each template item by evaluating the provided parameter. For number ranges, this means generating a URL for each segment number in the range provided (inclusive).

The complete URL list is formed by expanding each URL template in turn, creating an ordered list of URLs.

See Annex F for examples of the push template under various scenarios.

6.2 Push Strategy Definitions

The Table 2 below provides the PushDirectives defined in this specification with their type and parameters.

Table 2 — Valid values for PushDirective

PushType	PushParams	Description
`urn:mpeg:dash:fdh:2016:push-fast-start`	N/A	Indication that, along with an MPD, initialization data are considered for push A server receiving such push directive may push some or all available initialization segments related to the requested MPD. A client receiving such push directive is informed that a server intends to push some or all available initialization segments. <Editors' Note: Adding additional parameterizations is recognized as valuable to push the more adapted initialization segments and optionally few media segments. National Bodies are kindly invited to provide comments and contributions on the details of which parameters may be signaled and what information a client may need to be signaled back to take advantage of the fast start >

urn:mpeg:dash:fdh:2016:push-next	K:Number	Indication that the next K segments, using the requested segment as the initial index are considered for push. A server receiving such push directive may push consecutive segments to the requested one. A client receiving such push directive is informed that server intends to push the next segments consecutive to the requested one.
urn:mpeg:dash:fdh:2016:push-none	N/A	Indication that no push should occur. A server receiving such push directive should prevent from pushing. A client receiving such push directive is informed that server does not intend to push.
urn:mpeg:dash:fdh:2016:push-template	URLTemplate	Indication that some segments as described by the URL template are considered for push. A server receiving such push directive may use it to identify some segments to push. A client receiving such push directive can be informed on the segments the server intends to push.
urn:mpeg:dash:fdh:2016:push-time	T:Number	Indication that the next segments until the specified segment time (presentation time of the first frame) a segment exceeds time T, beginning with the requested segment are considered for push. A server receiving such push directive may push a given duration of media segments. A client receiving such push directive is informed that server intends to push a given duration of media segments.

<Editors' Note1: All of the push directives currently defined in this document assume that pushed segments all come from a single representation. There is no capability to push segments from different representations in a single transaction. We ask for comments as to whether this limitation is reasonable.>

7 Server Push over HTTP/2

7.1 PushDirective Binding

In HTTP/2, Push Directives may be signalled using an HTTP header in a request with the following form:

ACCEPT_PUSH_POLICY = “Accept-Push-Policy:” PUSH_DIRECTIVE “;” PARAMS

PUSH_DIRECTIVE = < a PushDirective as specified in section 6.1.2. >

PARAMS = “q=” FLOAT

Where “q” is a floating point value indicating the relative priority of each directive, when multiple directives are present.

7.2 PushAck Binding

In HTTP/2, Push Acknowledgments may be signalled using an HTTP header provided in a response with the following form:

PUSH_POLICY = “Push-Policy:” PUSH_ACK

PUSH_ACK = < a PushAck as is specified in section 6.1.3 >

7.3 Cancelling a push request

It is possible for a client to cancel a push sequence by sending RST_STREAM frames each referencing the promised stream identifiers as specified in HTTP/2.

In the case where the cancel is to take effect immediately the client will issue a RST_STREAM on all pushed segments that have been promised by the server via a PUSH_PROMISE frame. In the case where the cancel is not immediate the client should continue to receive the next pushed segment, and cancel all other promised segments using RST_STREAM.

8 Server Push over WebSockets

8.1 Message Flow over WebSockets

Error! Reference source not found. Figure 2 shows the message flow for carrying an MPEG-DASH media presentation over a full duplex WebSocket session. Messages are defined to allow for MPD and segment objects to be delivered over a WebSocket sub-protocol. These messages may carry Push Directives that signal additional segment objects to be delivered over the WebSocket channel. Note this flow is identical to the general message flow described in Section 4, using WebSocket-specific message bindings.

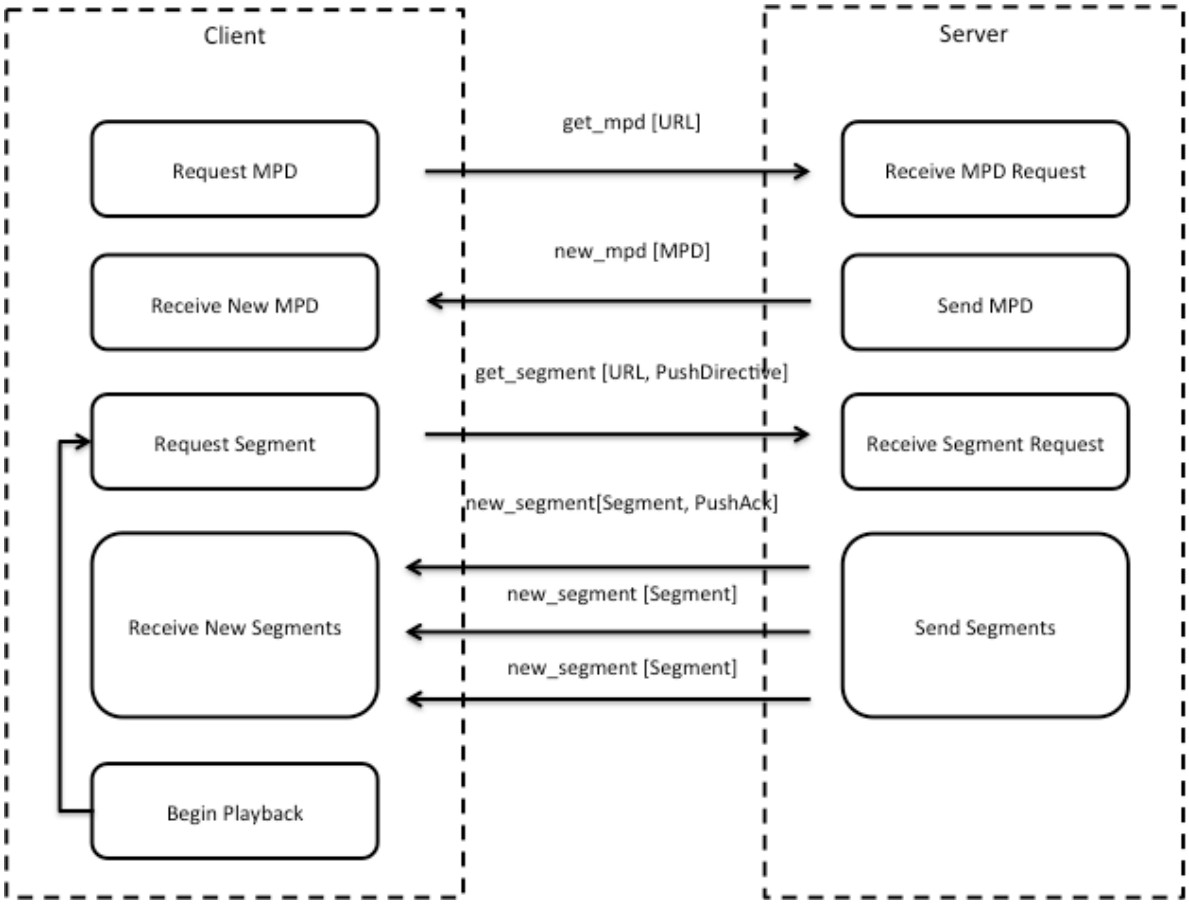

Figure 2 — Message flow over WebSocket

8.2 WebSocket sub-protocol for MPEG-DASH

8.2.1 MPEG-DASH WebSocket Frame Format and Semantics

The DASH sub-protocol uses the ‘binary’ format (opcode ‘binary’ or any ‘continuation’ frames thereof) for all messages exchanged over the WebSocket connection, as described in RFC 6455.

The MPEG-DASH sub-protocol frame consists of a frame header and frame payload. The frame header shall be formed as WebSocket frame Extension Data, which shall be present and of which the size can be determined as 4+4*EXT_LENGTH bytes as given by the DASH sub-protocol frame header.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   STREAM_ID   |   MSG_CODE    |  F |       EXT_LENGTH          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Extension (JSON encoded parameters)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

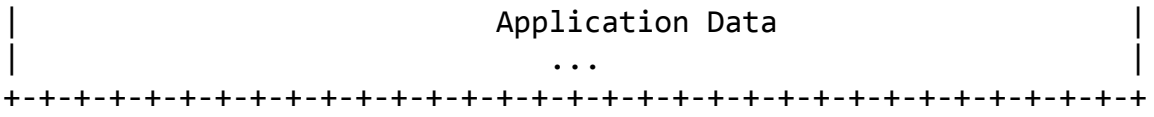

Figure 3 — DASH sub-protocol frame header for WebSocket

The DASH sub-protocol frame header is defined as follows:

STREAM_ID: 8 bits
Is an identifier of the current stream, which allows multiplexing of multiple requests/responses over the same WebSocket connection. The responses to a particular request shall use the same STREAM_ID as that request. The appearance of a new STREAM_ID indicates that a new stream is connected. A cancel request, an end of stream, or an error message close the stream identified by the carried STREAM_ID.

MSG_CODE: 8 bits
Indicates the MPEG-DASH message represented by this frame. Available message codes are defined in Section 8.3.

F: 3 bits
This field provides a set of flags that are to be set and interpreted based on the command.

EXT_LENGTH: 13 bits
Provides the length in 4 bytes of the extension data that precedes the application data. The extension header must be a JSON encoding of additional information fields that apply to the request/response. To align with 4 byte boundaries, padding 0 bytes may be added after the extension header.

Extension: 4*EXT_LENGTH
The extension header must be a JSON encoding of additional information fields that apply to the request/response. To align with 4 byte boundaries, padding 0 bytes may be added after the extension header. The content shall be encoded in UTF-8 format. All NVP have to be at the root level.

8.2.2 Definition of WebSocket Streams

The DASH sub-protocol for WebSocket defines the concept of streams that allows for an independent, bi-directional, sequence of frames to be exchanged between client and server. Multiple streams may be created on top of the same WebSocket connection. The server shall send responses to client's requests on the same stream that was used to submit the request. The streams are identified by their STREAM_ID as defined in section 8.2.2.

Each stream shall only carry at most one push directive and its responses. New Push Directives shall be started in a new stream.

8.3 WebSocket Message Codes

Table 3 —List of available DASH sub-protocol message codes

Message Code	Message	Definition
1	get_mpd	8.4.1
2	get_segment	8.4.2
3	new_mpd	8.4.3
4	new_segment	8.4.4
255	cancel	**Error! Reference source not found.**

8.4 WebSocket Message Definitions

8.4.1 MPD request (Client → Server)

The MPD request message initiates the request for a DASH MPD file. A Push Directive may be provided with the MPD request.

- Message Name: get_mpd

- Supplied Arguments

Parameter Name	Type	Cardinality	Description
mpd_uri	URI	1	the full URI for the MPD being requested
push_directive	PushDirective	0..N	A push strategy to be applied to this MPD request. This is typically used to signal the server to send initialization data to the client along with the MPD (“fast start”).

- Preconditions

  - None

- Postconditions

- The MPD request is initiated and pending all requested *new_mpd* messages are sent from the server to the client. The *new_mpd* message indicates that the server has responded with a requested MPD.

- A Push Acknowledgment in the *new_mpd* message may indicate that server understood and applied the Push Directive indicated by the client.

- Errors/Exceptions

- None

8.4.2 Segment request (Client → Server)

The segment request message initiates the request for a DASH segment. The segment requests may include a Push Directive to inform the server to actively push one or more future segments.

-Message Name: get_segment

- Supplied Arguments

Parameter Name	Type	Cardinality	Description
`segment_uri`	URI	1	the full URI for the video segment being requested
`push_directive`	PushDirective	0..N	the desired push strategy for requesting the segment.

- Preconditions

- The client has a valid MPD.

- Postconditions

- The segment request is initiated and pending until all requested segments (including any segments to be pushed) are received by the client.

- Depending on the provided Push Directive, the client may receive one or more server pushed segments following the requested segment.

- Errors/Exceptions

- Server Push not available. Triggered when a Push Directive is specified but the server detects that a full duplex channel does not exist or does not function normally at the time of request.

8.4.3 MPD received (Server → Client)

This message represents the server's response from a previous *get_mpd* message sent by the client.

- Message Name: new_mpd

- Supplied Arguments

Parameter Name	Type	Cardinality	Description
`mpd`	MPD	1	The MPD returned by the server
`push_acknowledge`	PushAck	0..N	The push strategy that the server will follow

- Preconditions

- The client requested an MPD by sending the *get_mpd* message, optionally with one or more Push Directives

- Postconditions

- The client is ready to parse the received MPD.
- The client is informed on the push strategy to be taken by the server, including possibly that no push strategy will be in effect.

- Errors/Exceptions

- None

8.4.4 Segment received (Server → Client)

This message represents the server's response from a previous *get_segment* message sent by the client. A server may issue multiple responses for a single request, as appropriate for the push strategy in the corresponding *get_segment* message.

- Message Name: new_segment

- Supplied Arguments

Parameter Name	Type	Cardinality	Description
`segment`	Segment	1	The segment returned by the server
`push_acknowledge`	PushAck	0..N	The push strategy that the server will follow

- Preconditions

  - The client requests a segment by sending the *get_segment* message.

- Postconditions

  - The client is ready to parse the received segment and process the media.
  - The client is informed on the push strategy to be taken by the server, including possibly that no push strategy will be in effect.

- Errors/Exceptions

  - None

8.4.5 Segment cancel (Client → Server)

This message represents a client request for the server to cancel the outstanding push transaction over a given WebSocket stream. If no outstanding push transaction is in effect this message will have no effect. In the case where the cancel is to take effect immediately (signalled by the “immediate” parameter in the description of this message) the server should cancel all pushed segments that have been scheduled by the server. In the case where the cancel is not immediate the server should continue to send the next pushed segment, and cancel all other scheduled segments.

- Message Name: segment_cancel

- Supplied Arguments

Parameter Name	Type	Cardinality	Description
`immediate`	Boolean	1	If true, the client indicates that it would like the server to stop transmission immediately. If false, the client indicates it would like the server to complete transmission of the currently pushed segment (if any) before cancelling the transaction.

- Preconditions

  - The client has initiated a push transaction via an earlier call to get_segment.
  - The server has not completed the requested push transaction.

- Postconditions

  - The push transaction is no longer maintained at the server, and no future segments will be pushed.

- Errors/Exceptions

- None

8.5 MPEG-DASH Sub-protocol Registration

RFC 6455 [1] requires that sub-protocols be registered with the IANA [2]. The registry requires the following information:

Subprotocol-Identifier: “mpeg-dash”

Subprotocol Common Name: “MPEG-DASH”

Subprotocol Definition: refers to this specification.

Annex A
(informative)

Considered Use Cases

A.1 Use Case 1: Basic Streaming for VOD

A viewer begins a playback session for a DASH stream. The DASH client begins the playback session in the usual way, by requesting or otherwise acquiring the DASH MPD. Through some means of protocol negotiation, the client establishes a push session with a push-enabled media server. Using its knowledge of the content and network conditions, the server transmits DASH content segments and/or MPDs to the client, which plays them back just as it would had the client requested those segments over HTTP. As this is VOD content, initial playback usually occurs at the beginning of the presentation and ends when the entirety of the presentation has completed.

A.2 Use Case 2: Basic Streaming for Live

A viewer begins playback as described in Use Case 1, but as this is a live stream playback begins at the “live” end of the presentation, and continues indefinitely until the live stream ends.

A.3 Use Case 3: Seeking

A viewer begins playback as described in Use Case 1 or 2 (using time-shifting). At some point during playback of the presentation, the viewer seeks to a particular point in the VOD presentation, or within the time shift buffer of the live stream. Playback begins at the new location.

A.4 Use Case 4: Trick Play

A viewer begins playback as described in Use Case 1 or 2 (using time-shifting). At some point during playback of the presentation, the viewer begins to fast-forward to a new point in the VOD presentation, or within the time shift buffer of the live stream. Playback begins at the new location.

A.5 Use Case 5: HTTP-compatible full duplex protocol not supported by Client

A viewer begins playback as described in Use Case 1 or 2. The DASH client does not support a push-based protocol, although in this case the server does. The playback session is initiated and operates smoothly, using HTTP as a transport.

A.6 Use Case 6: HTTP-compatible full duplex protocol not supported by Server

A viewer begins playback as described in Use Case 1 or 2. The server does not support a push-based protocol, although in this case the DASH client does. The playback session is initiated and operates smoothly, using HTTP as a transport.

Annex B
(informative)

System Architecture for HTTP/2

The architecture of an end-to-end video streaming system over HTTP/2 is shown in Figure 4. There are three major system components: (1) the origin server to host the video assets for streaming, which is an HTTP/2 enabled web server deployed with one or more video push strategies (2) the DASH client to receive and play back the video stream, which consists of a HTTP/2 enabled web browser and a video player; and (3) a content distribution network (CDN) in between the client and origin, which consists of HTTP/2 enabled web cache servers, deployed with one or more push strategies.

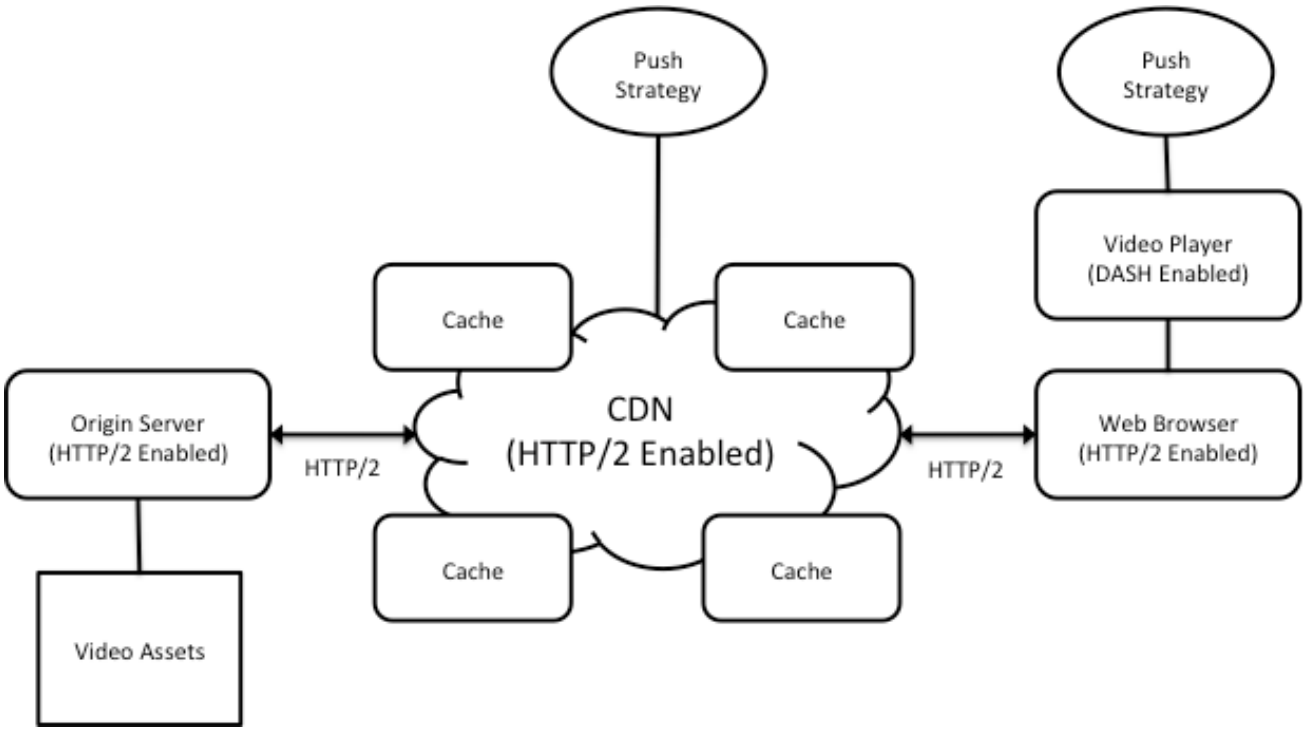

Figure 4 — System Architecture of HTTP/2 DASH Streaming

In the above system, there are two HTTP/2 persistent connections, one between the client and the CDN, and one between the CDN and the origin server. In addition, a tunnelled HTTP/2 connection may also be established between the client and origin, for live streaming that requires low latencies. Unlike HTTP 1.0/1.1 streaming, in HTTP/2 the server (origin or cache) can actively push segments (or MPDs) to the client (or the CDN) as soon as they are generated, in addition to the resources that have been explicitly requested by the client (or the CDN).

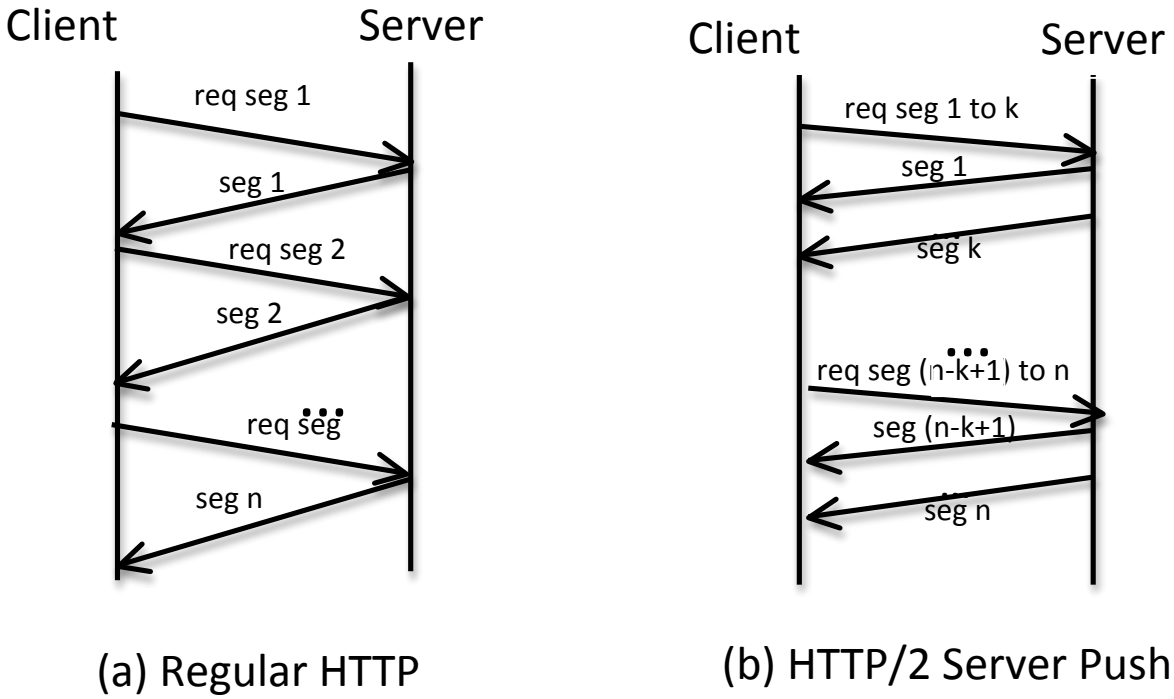

Figure 5 — Pushing Segments using HTTP/2 Server Push

Annex C
(informative)

Examples of HTTP/2 Client/Server Behaviour

<Editors' Note: additional examples may be required in this section. We invite comments on additional examples that would be useful. >

C.1 Example of segment push using "push-next"

In this example, a client requests that the server pushes the next two segments after the one initially requested.

Request [Stream ID = 1]:

```
HEADERS
   + END_STREAM
   + END_HEADERS
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment1
   accept-push-policy = urn:mpeg:dash:fdh:2016:push-next";2;q=1.0
```

Response [Stream ID = 1]:

```
PUSH_PROMISE
   Stream ID = 2
   + END_HEADERS
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment2
PUSH_PROMISE
   Stream ID = 4
   + END_HEADERS
```

```
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment3
HEADERS
   + END_STREAM
   + END_HEADERS
   :status = 200
   push-policy = “urn:mpeg:dash:fdh:2016:push-next”;2
DATA
    + END_STREAM
{binary data for segment 1}
```

Response [Stream ID = 2]:

```
HEADERS
   + END_STREAM
   + END_HEADERS
   :status = 200
DATA
    + END_STREAM
{binary data for segment 2}
```

Response [Stream ID = 4:

```
HEADERS
   + END_STREAM
   + END_HEADERS
   :status = 200
```

```
DATA
    + END_STREAM
{binary data for segment 3}
```

C.2 Example of segment push using “push-template”

In this example, a client requests that the server pushes a set of segments based on a provided push template.

Request [Stream ID = 1]:

```
HEADERS
   + END_STREAM
   + END_HEADERS
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment1
   accept-push-policy = “urn:mpeg:dash:fdh:2016:push-
template”;”../rendition1/segment{}”:{2,3};q=1.0
```

Response [Stream ID = 1]:

Response [Stream ID = 1]:

```
PUSH_PROMISE
   Stream ID = 2
   + END_HEADERS
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment2
PUSH_PROMISE
   Stream ID = 4
   + END_HEADERS
   :method = GET
```

```
   :scheme = http

   :path = /example/rendition1/segment3

HEADERS

   + END_STREAM

   + END_HEADERS

   :status = 200

   push-policy = “urn:mpeg:dash:fdh:2016:push-
template”;”../rendition1/segment{}”:{2,3}

DATA

    + END_STREAM

{binary data for segment 1}
```

Response [Stream ID = 2]:

```
HEADERS

   + END_STREAM

   + END_HEADERS

   :status = 200

DATA

    + END_STREAM

{binary data for segment 2}
```

Response [Stream ID = 4:

```
HEADERS

   + END_STREAM

   + END_HEADERS

   :status = 200

DATA
```

```
    + END_STREAM

{binary data for segment 3}
```

C.3 Example of initiating a push request with a server that does not support push

In this example, a client requests that the server pushes the next two segments after the one initially requested. The server is an older server that does not understand Push Directives. The server does not return a Push Acknowledgement or promise any additional segments.

Request [Stream ID = 1]:

```
HEADERS

   + END_STREAM

   + END_HEADERS

   :method = GET

   :scheme = http

   :path = /example/rendition1/segment1

   accept-push-policy = "urn:mpeg:dash:fdh:2016:push-next";2;q=1.0
```

Response [Stream ID = 1]:

```
HEADERS

   + END_STREAM

   + END_HEADERS

   :status = 200

DATA

    + END_STREAM

{binary data for segment 1}
```

In this alternative example, the server *does* understand the Push Directive, but is not configured to deliver pushed segments or has otherwise elected not to honor the push request. The server explicitly signals this with a Push Aknowledgment of "urn:mpeg:dash:fdh:2016:push-none".

Request [Stream ID = 1]:

```
HEADERS

   + END_STREAM

   + END_HEADERS

   :method = GET

   :scheme = http

   :path = /example/rendition1/segment1

   accept-push-policy = "urn:mpeg:dash:fdh:2016:push-next";2;q=1.0
```

Response [Stream ID = 1]:

```
HEADERS

   + END_STREAM

   + END_HEADERS

   :status = 200

   push-policy = "urn:mpeg:dash:fdh:2016:push-none"

DATA

    + END_STREAM

{binary data for segment 1}
```

C.4 Example of cancelling a push request

In this example, a client requests that the server pushes the next two segments after the one initially requested. The client receives the initial segment, as well as the next one. The client cancels the stream associated with the third segment, ending the push transaction. This example is representative of what may occur if the client decides to switch representations (i.e. an adaptive bitrate switch) after issuing a push request, or if an MPD update makes the previously requested segments unnecessary.

Request [Stream ID = 1]:

```
HEADERS

   + END_STREAM

   + END_HEADERS
```

```
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment1
   accept-push-policy = “urn:mpeg:dash:fdh:2016:push-next”;2;q=1.0
```

Response [Stream ID = 1]:

```
PUSH_PROMISE
   Stream ID = 2
   + END_HEADERS
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment2
PUSH_PROMISE
   Stream ID = 4
   + END_HEADERS
   :method = GET
   :scheme = http
   :path = /example/rendition1/segment3
HEADERS
   + END_STREAM
   + END_HEADERS
   :status = 200
   push-policy = “urn:mpeg:dash:fdh:2016:push-next”;2
DATA
    + END_STREAM
{binary data for segment 1}
```

Response [Stream ID = 2]:

```
HEADERS
   + END_STREAM
   + END_HEADERS
   :status = 200
DATA
    + END_STREAM
{binary data for segment 2}
```

Request [Stream ID = 4:

```
RST_STREAM
   Error Code = CANCEL
```

Annex D
(informative)

Examples of WebSocket Client/Server Behaviour

<Editor's Note: additional examples may be required in this section. We invite comments on additional examples that would be useful. >

D.1 Example of client requesting an MPD

In this example, a client requests that the server sends the specified MPD.

Client Request:

```
STREAM_ID : 1

MSG_CODE: 1

EXT_LENGTH: 27

EXT: {"mpd_uri":"./example.mpd"}
```

Server Response:

```
STREAM_ID : 1

MSG_CODE: 3

EXT_LENGTH: 0

{binary data with example.mpd}
```

D.2 Example of client requesting a segment, using a push directive

In this example, the client requests a segment, indicating that the server should push the next two segments after the one initially requested.

Client Request:

```
STREAM_ID : 1

MSG_CODE: 2

EXT_LENGTH: 98

EXT:
{"segment_uri":"./rep1/segment1.mp4","push_directive":"urn:mpeg:dash:fdh:
2016:push-next;2"}
```

Server Response:

```
STREAM_ID : 1

MSG_CODE: 4

EXT_LENGTH: 59

EXT: {”push_policy”:”urn:mpeg:dash:fdh:2016:push-next;2”}

{binary data with segment1.mp4}


STREAM_ID : 1

MSG_CODE: 4

EXT_LENGTH: 0

{binary data with segment2.mp4}


STREAM_ID : 1

MSG_CODE: 4

EXT_LENGTH: 0

{binary data with segment3.mp4}
```

D.3 Example of cancelling a push request

In this example, the client asks the server to cancel any outstanding requests on stream ID 1. The immediate flag is signalled, meaning that the client wishes the server to stop sending data immediately.

Client Request:

```
STREAM_ID : 1

MSG_CODE: 255

EXT_LENGTH: 15

EXT: {“immediate”:1}
```

Annex E
(informative)

Protocol Upgrade and Fallback Procedure for WebSocket

E.1 Upgrade to DASH Sub-protocol over WebSocket

The DASH sub-protocol is identified by the name “dash” in the handshake request. A client wishing to use WebSocket for DASH streaming shall include the keyword “dash” as part of the *Sec-WebSocket-Protocol* header field together with the protocol upgrade request.

The following is an example of a WebSocket handshake in which the client requests the WebSocket DASH sub-protocol from the server:

```
GET / HTTP/1.1
Host: dash.mpeg.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dFhmILNhbYCsXSBub25jZQ==
Origin: http://www.example.com
Sec-WebSocket-Protocol: dash
Sec-WebSocket-Version: 13
```

The response from the server for a successful upgrade may look like this:

```
HTTP/1.1 101

Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: p3sPMLciToaR9kXGzhzYRbL+xOo=

Sec-WebSocket-Protocol: dash
```

E.2 Fallback to HTTP/1.1

If the server does not support WebSocket or the WebSocket DASH sub-protocol, an error event is emitted to inform the client that the upgrade to websocket failed. Upon reception of the error event, the client shall revert back to the usage of regular HTTP/1.1 (e.g. through XmlHttpRequest) and perform the resource requesting as usual.

When implemented in a web browser, the event handler may look like this:

```
window.ws.onerror = function(event) {
    // fallback to XHR
}
```

Annex F
(informative)

Examples of Push Template

<Editors' Note: additional examples may be required in this section. We invite comments on additional examples that would be useful. >

F.1 Example of push template with a list of segment numbers

This example shows a push template which lists segment numbers to be pushed. This would be appropriate to use with (for example) a SegmentTemplate using the $Number$ macro:

"../rep1/segment{%02d}.mp4" : {2, 3, 4}

This would expand to the following list of URLs:

../rep1/segment02.mp4

../rep1/segment03.mp4

../rep1/segment04.mp4

F.2 Example of push template with a range of segment numbers

This example shows a push template which lists segment numbers to be pushed. This would be appropriate to use with (for example) a SegmentTemplate using the $Number$ macro:

"../rep1/segment{%02d}.mp4" : {2-4}

This would expand to the following list of URLs:

../rep1/segment02.mp4

../rep1/segment03.mp4

../rep1/segment04.mp4

F.3 Example of push template with list of segment times

This example shows a push template which lists segment to be pushed which are based on the segment time. This would be appropriate to use with (for example) a SegmentTemplate using the $Time$ macro with a SegmentTimeline indicating a repetition of segments having a constant duration of 6006 (in timescale units).

"../rep1/segment{%06d}.mp4" : {6006, 12012, 18018}

This would expand to the following list of URLs:

../rep1/segment006006.mp4

../rep1/segment012012.mp4

../rep1/segment018018.mp4

F.4 Example of push template with multiple URL templates

This example shows multiple URLs templates in a single push template string.

"../rep1/segment{%02d}.mp4" : {2-4}, "../rep2/segment{%02d}.mp4" : {5-7}

This would expand to the following list of URLs:

../rep1/segment02.mp4

../rep1/segment03.mp4

../rep1/segment04.mp4

../rep2/segment05.mp4

../rep2/segment06.mp4

../rep2/segment07.mp4

F.5 Example of push template with no macro expansion (simple list)

This example shows multiple URLs templates in a single push template string, none of which include an expansion macro. In this case, the value specifier is optionally, making the string a simple list of URLs. This may be useful with segment names with no fixed pattern, as may be described using SegmentList

"../rep1/segment1650.mp4", "../rep1/seg1900.mp4", "../rep1/segment3500.mp4"

This would expand to the following list of URLs:

../rep1/segment1650.mp4

../rep1/segment1900.mp4

../rep1/segment3500.mp4