DASH-IF Implementation Guidelines: DASH events and timed metadata tracks timing and processing model and client reference model

July 10, 2020

DASH Industry Forum

Version 1.0.2 (Final)

Scope

The scope of this document is the timing and processing of the DASH event and timed metadata track timing as well as the corresponding DASH client-Application API for these events and metadata.

The DASH standard includes MPD events to include in DASH manifest. It also enables including inband events to be carried with the media segments. Finally, it allows the streaming of metadata tracks. All three functionalities may be used to deliver application-related events to the Application. Since these data are timed sensitive, the DASH client must receive this data and pass it to Application such that the application can use them on time.

This document draws the timing model for DASH events and timed metadata tracks based on the MPEG DASH specification. Based on this timing model, this document also includes a client processing model for these data. Finally, this document also includes an API for dispatching events and timed metadata information to the Application as well as a WebIDL instantiation of this API.

This document is expected to be integrated into the DASH-IF IOP V5.0 guidelines.

Disclaimer

This is a document made available by DASH-IF. The technology embodied in this document may involve the use of intellectual property rights, including patents and patent applications owned or controlled by any of the authors or developers of this document. No patent license, either implied or express, is granted to you by this document. DASH-IF has made no search or investigation for such rights and DASH-IF disclaims any duty to do so. The rights and obligations which apply to DASH-IF documents, as such rights and obligations are set forth and defined in the DASH-IF Bylaws and IPR Policy including, but not limited to, patent and other intellectual property license rights and obligations. A copy of the DASH-IF Bylaws and IPR Policy can be obtained at http://dashif.org/.

The material contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, this material is provided AS IS, and the authors and developers of this material and DASH-IF hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of workmanlike effort, and of lack of negligence.

In addition, this document may include references to documents and/or technologies controlled by third parties. Those third-party documents and technologies may be subject to third party rules and licensing terms. No intellectual property license, either implied or express, to any third-party material is granted to you by this document or DASH-IF. DASH-IF makes no any warranty whatsoever for such third party material.

If you have comments on the document or identify and bugs or problems, please submit them by one of the following means:

- at the github repository https://github.com/Dash-IndustryForum/Events/issues, or
- at the public repository https://dashif+iop@groupspaces.com with a subject tag [Events]

Please add a detailed description of the problem and the comment.

Note that technologies included in this document and for which no test and conformance material is provided, are only published as a candidate technology, and may be removed if no test material is provided before releasing a new version of this guidelines document. The status of the test material can be verified on http://testassests.dashif.org.

Contents

List of Figures

List of Tables

List of Equations

1 Introduction

This section describes the DASH events and time metadata track timing and processing model. It describes the timing models of MPD and inband events as well as the timing of the timed metadata tracks. This section also outlines the DASH player's reference architecture for processing the DASH events as well as timed metadata tracks, the possible dispatch modes, and the information carried to the Application. Finally, it defines the reference API for the Application to subscribe to the events and/or metadata tracks as well as the API for dispatching event instances and metadata samples.

A server/application provider should consider the information provided in this section for building interactive application as the timing and processing model of the events and metadata impact the usability and capabilities of application build using these features.

2 DASH Player architecture for processing DASH events and timed metadata tracks

Figure 1 demonstrates a generic architecture of DASH Player including DASH Events and timed metadata tracks processing models.

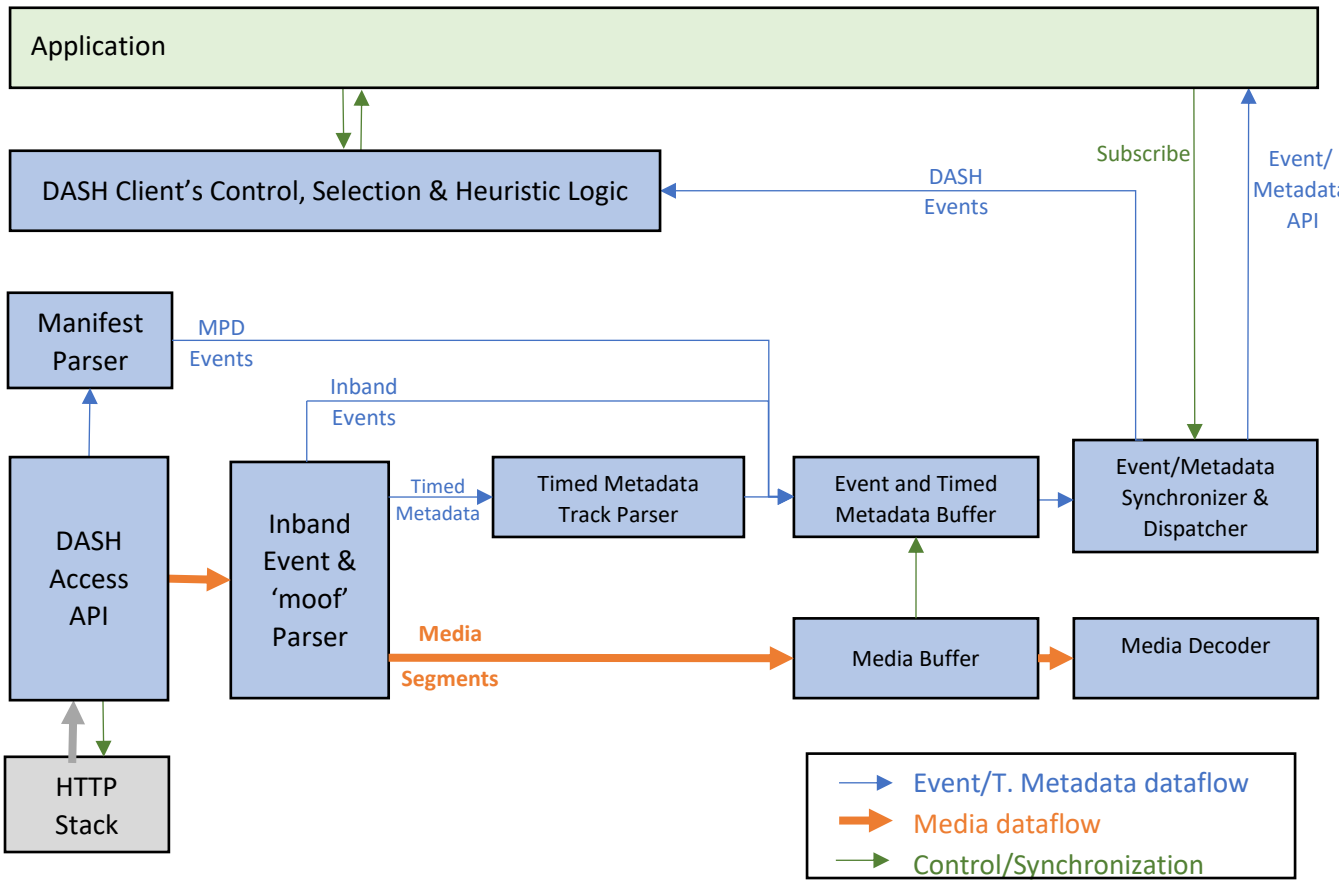

Figure 1 DASH Player architecture including the inband Event and Application-related timed metadata handling

In the above figure:

1. The DASH Player processes the received MPD. For every Period, the MPD may include one or multiple Event Streams (each scope by a scheme/value pair), and Adaptation Sets that carry Representation/tracks for timed metadata. Some or all of these event streams/timed metadata tracks may be suitable for consumption by the application. In this clause, we refer to any of these streams/tracks as application event or metadata streams (AEMS).

2. The Application subscribes to all AEMSs of interest and also specifies the desired dispatch mode for each AEMS.

3. If the MPD includes any MPD Event streams, the DASH Player parses each Event in the Event Stream accordingly and appends the relevant to the Event & Timed Metadata Buffer, based on their presentation time in MPD.

4. Based on the information in the MPD, the DASH Player selects and schedules the fetching of media Segments and appends them to the Media Buffer. This is typically done to maintain a stable playback buffer, but media segments are typically only accessed close to the time before their playback is scheduled. Parsing a Segment includes:

    a. Parsing high-level boxes such as Segment Index (sidx), Event Message (emsg), Producer Reference Time (prft), movie fragment header (moof) boxes, and interpreting the information in the DASH client. For inband event streams, the emsg and typically the moof need to be parsed. The DASH client then uses this information and appends the relevant data to the Event & Timed Metadata Buffer.

    b. For an Application-related timed metadata track, the entire Representation/track is parsed including the Initialization Segment (i.e. track header) as well as the Media Segments (i.e. the movie fragments and media data containers). The DASH client then uses this information and appends the relevant data to the Event & Timed Metadata Buffer.

5. Event & Metadata Buffer passes the events and timed metadata samples to Event & Metadata Synchronizer and Dispatcher function. An example of the Buffer's data object is defined in subclause 11 of this section.

6. The DASH Player-specific Events are dispatched to DASH Player's Control, Selection & Heuristic Logic, while the Application-related Events and timed metadata track samples are dispatched to the application as the following. If an Application is subscribed to a specific AEMS, dispatch the corresponding event instances or timed metadata samples, according to the dispatch mode:

    a. For on-receive dispatch mode, dispatch the entire event or timed metadata information as soon as they are appended to the Event and Timed Metadata Buffer.

    b. For on-start dispatch mode, dispatch the message data of the event their associated presentation time or latest before the event duration has ceased, or timed metadata samples at their presentation time using the synchronization signal from the media decoder.

Note: The metadata buffer maintains a sequence of events and timed metadata samples. The maintain and purging management of this buffer is synchronized with the Media buffer management.

3 Inband Event timing parameters

Figure 2 presents the timing of inband Events along the media timeline:

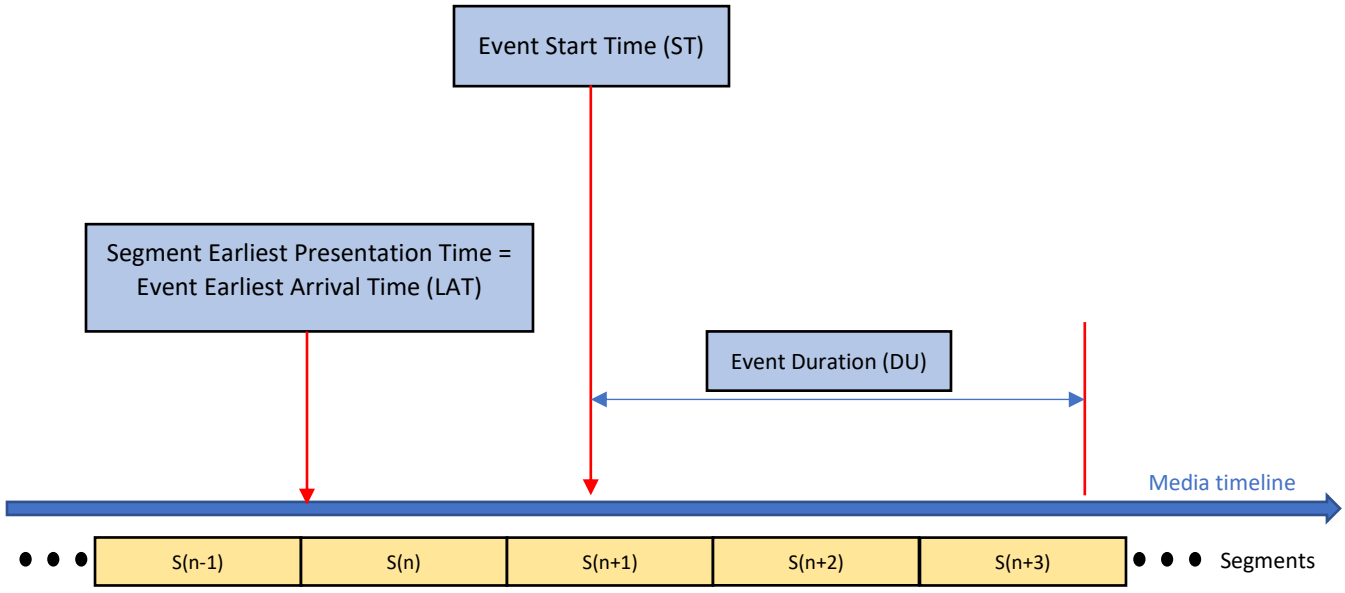

Figure 2 Inband event timing parameter on the media timeline

As shown in Figure 2, every inband Event can be described by three timing parameters on the media timeline:

1. Event Latest Arrival Time (*LAT*) which is the earliest presentation time of the Segment containing the Event Message box. An inband Event is inserted at the beginning of a Segment. Since each media segment has the earliest presentation time equal to (*LAT*), *LAT* of the Segment carrying the Event Message box can be considered as the time instance of that box on the media timeline. The DASH Player is expected to fetch and parse the Segment before or at its earliest presentation time. Therefore, an Event inserted in a Segment with EPT will be available in the client no later than EPT of the carrying Segment on the media timeline. Therefore, the Event inserted in a Segment will be ready to be processed and fetched no **later than** *LAT* on the media timeline. In the case in which the event is inserted at the beginning of a Chuck and the chucks are delivered in low latency mode, i.e. HTTP chunked encoding mode, LAT is the earliest presentation time of the corresponding Chunk.

2. Event Presentation/Start Time (*ST*) which is the moment in the media timeline that the Event becomes active. *ST* is the moment in the media timeline that the Event becomes active. This value can be calculated using the parameters included in the DashEventMessageBox.

3. Event duration (*DU*): the duration for which the Event is active. *DU* is signaled in the Event Message box using a specific value.

3.1 Timing parameters

The ST of an event can be calculated using the values in the corresponding emsg box of [MPEGDASH] subclause 5.10.3.3.1:

$$ST = \begin{cases} PeriodStart - \frac{SegmentBase@PresentationTimeOffset}{SegmentBase@timescale} + LAT + \frac{emsg@presentation\_time\_delta}{emsg@timescale} & version\ 0 \\ PeriodStart - \frac{InbandEventStream@presentationTimeOffset}{InbandEventStream@timescale} + \frac{emsg@presentationtime}{emsg@timescale} & version\ 1 \end{cases}$$

Equation 1 Event Start Time of inband event

Where:

- SegmentEPT is the earliest presentation time of the media Segment
- SegmentBase@PresentationTimeoffest is the presentation time offset of media
- emsg@x is the field x in emsg box, and
- LAT is the earliest presentation time of the Segment containing the Event Message box.

Note: ST is always equal to or larger than LAT in both versions of emsg.

Note: Since the media sample timescales might be different from the emsg's timescale, ST might not line up exactly with a media sample presentation time if different timescales are used.

In this section, we use the following common variable names instead of some of above variables to harmonize parameters between Inband events, MPD events, and timed metadata samples:

- *scheme_id* = scheme_id_uri
- *value* = value
- *presentation_time* = *ST*
- *duration* = event_duration/timescale
- *message_data* = message_data()

4 MPD Events timing model

MPD Events carry a similar data model to inband Events but are carried in the MPD, under the Period elements. Each Period event can have EventStream element(s), defining the EventStream@schemeIdUri, EventStream@value, EventStream@timescale, and contained sequences of Event elements. Each event may have Event@presentationTime, Event@duration, Event@id, and Event@messageData attributes as specified in [MPEGDASH] subclause 5.10.2. As is shown in Figure 3, each MPD Event has three timing parameters along the media timeline:

1. The PeriodStart Time (*LAT*) of the Period element containing the EventStream element.

2. Event Start Time (*ST*): the moment in the media timeline that a given MPD Event becomes active and can be calculated from the attributeEvent@presentationTime.

3. Event duration (*DU*): the duration for which the event is active that can be calculated from the attribute Event@duration.

Note that the first parameter is inherited from the Period containing the Events and only the 2nd and 3rd parameters are explicitly included in the Event element. Each EventStream also has EventStream@timescale to scale the above parameters.

Figure 3 demonstrates these parameters in the media timeline.

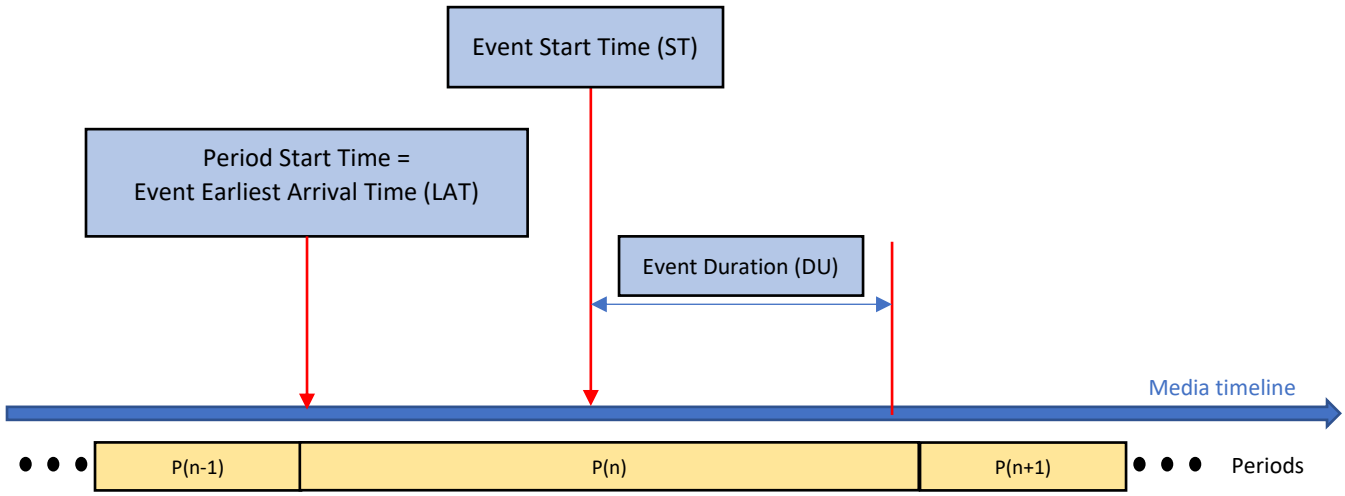

Figure 3 MPD events timing model

4.1 Timing parameters

The ST of an MPD event, relative to PeriodStart of Period containing the Event, can be calculated using values in its EventStream and Event elements:

$$ST = PeriodStart - \frac{EventStream@presentationTimeOffset}{EventStream@timescale} + \frac{Event@presentationTime}{EventStream@timescale}$$

Equation 2 Event Start Time of MPD event

In this section, we use the following common variable names instead of some of above variables to harmonize parameters between Inband events, MPD events, and timed metadata samples:

- *scheme_id* = EventStream@schemeIdUri
- *value* = EventStream@value
- *presentation_time* = *ST*
- *duration* = Event@duration/EventStream@timescale
- *id* = Event@id
- *message_data* = decode64(Event@messageData)

In which decode() function is:

$$decode(x) = \begin{cases} x & Event@contentEncoding \ \ not\ present \\ base64\ decoding\ of\ (x) & Event@contentEncoding = base64 \end{cases}$$

Equation 3 Decoding the message_data payload

Note that the DASH client shall Base64 decode the Event@messageData value if the received Event@contentEncoding value is base64.

5 Timed metadata sample timing model

An alternative way to convey information synchronized to a media is using timed metadata tracks. Timed metadata tracks are ISOBMFF formatted tracks that obey the following characteristics according to [ISOBMFF]:

1. The **sample description box** stsd in the MovieBox contains a sampleEntry that is a URIMetaSampleEntry, to signal that the media samples contain metadata based on a urn in a URIBox to signal that scheme.
2. The **Handler Box** hdlr has handler_type set to **meta** to signal the fact that the track contains metadata
3. The null media header **nmhd** is used in the minf box
4. Contain metadata (non-media data relating to presentation) embedded in ISOBMFF samples

Figure 4 shows the timing model for a simple ISOBMFF timed metadata sample.

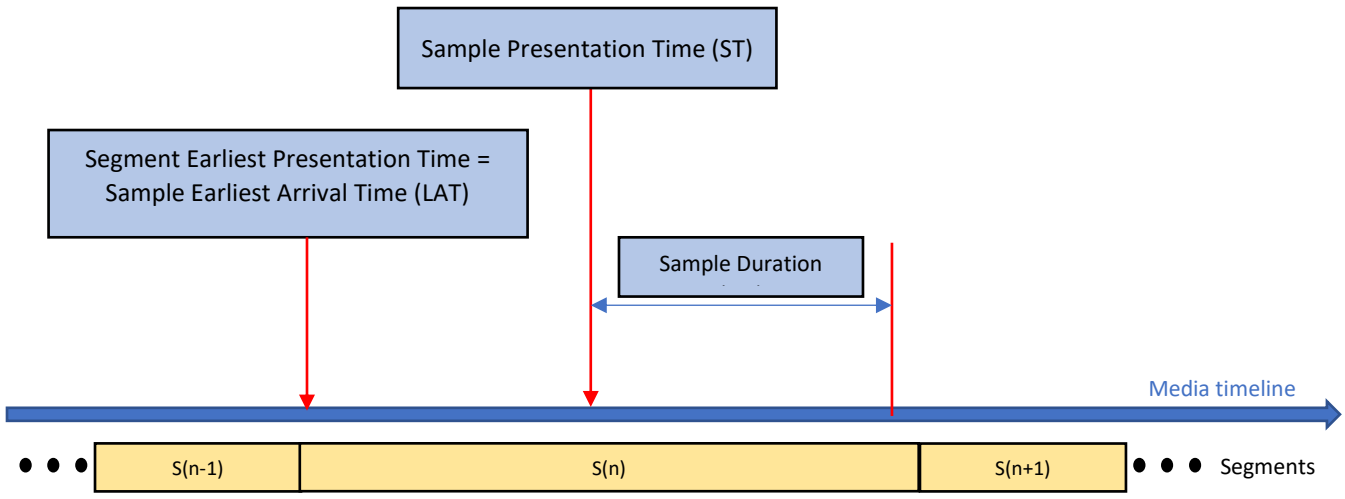

Figure 4 Timing parameters of a timed metadata sample on the media timeline

As shown in this figure, the metadata sample timing includes metadata sample presentation time (*ST*) and metadata sample duration (*DU*). Also, one or more metadata samples are included in a segment with Segment earliest presentation time (*LAT*).

Note that the metadata sample duration cannot go beyond DASH Segments/ISOBMFF fragment duration for fragmented metadata tracks, i.e. to the next fragment.

5.1 Timing parameters

In this document, we use the following variable names instead of some of the above variables to harmonize parameters between Inband events, MPD events, and timed metadata samples used in dispatch process:

- *scheme_id = track URI*, signalled in URIBox in URIMetaSampleEntry
- *timescale* = track timescale in mdhd box.
- *presentation_time = sample presentation time/timescale*
- *duration = sample duration/timescale*
- *message_data = sample data (exteracted from mdat)*

6 Events and timed metadata samples dispatch timing modes

Figure 5 shows two possible dispatch timing models for DASH events and timed metadata samples.

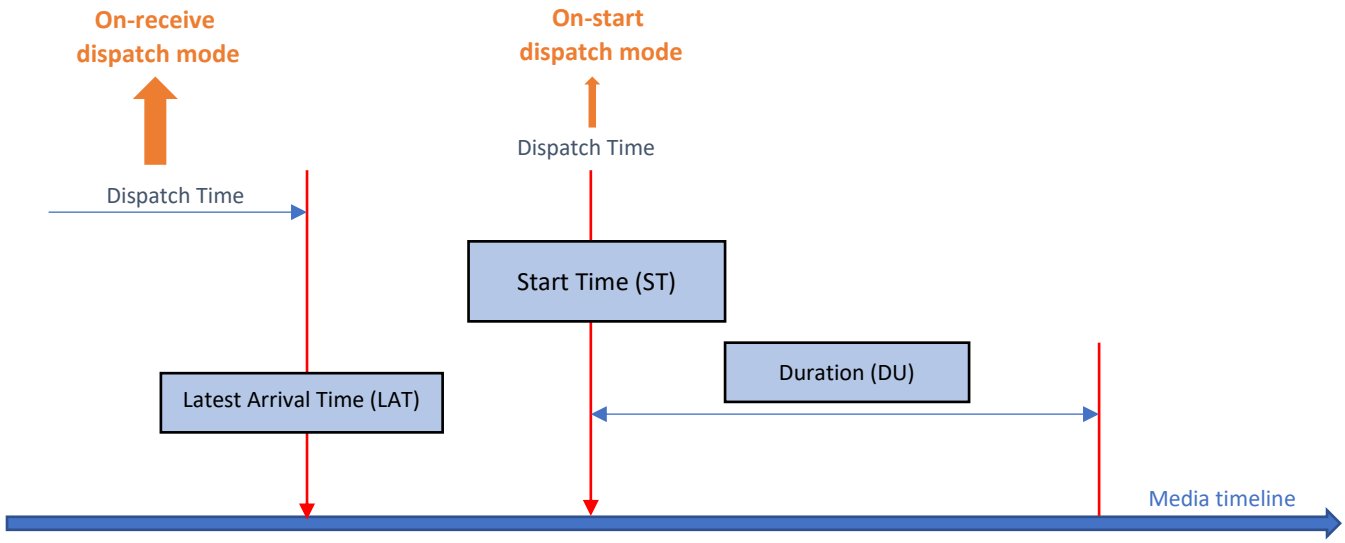

Figure 5 The Application events and timed metadata dispatch modes

In this figure, two modes are shown:

1. ***on-receive*** Dispatch Mode: Dispatching at *LAT* or earlier. Since the segment carrying an emsg/metadata sample has to be parsed at or before *LAT* on the media timeline, the event/metadata sample shall be dispatched at or before this time to the Application in this mode. The Application has a duration of *ST-LAT* for preparing for the event. In this mode, the DASH Player doesn't need to maintain the state of Application events or metadata samples. Applications must maintain the state for any event/metadata sample, its *ST* and *DU*, and monitor its activation duration if they need these. Applications may also need to schedule each event/sample at its *ST*.

2. ***on-start*** Dispatch Mode: Dispatching exactly at *ST*, which is the start/presentation time of the event/metadata sample. The DASH player shall dispatch the event to the application at the presentation time of the corresponding media sample, or in the case of the start of playback after that moment and during the event duration, at the earliest time within the event duration. In this mode, since the Applications receive the event/sample at its start/presentation time, it may need to act on the received data immediately.

Note: According to ISO/IEC 23009-1, the parameter duration has a different meaning in each dispatch mode. In the case of on-start, duration defines the duration starting from ST in which DASH Player shall dispatch the event exactly once. In the normal playback, the player dispatches the event at ST. However if DASH Player for instance seeks to a moment after ST and during the above duration, then it must dispatch the event immediately. In the case of on-receive, duration is a property of event instance and is defined by the scheme_id owner.

7 The Dispatch Processing Model

7.1 Prerequisite

Application is subscribed to a specific event stream identified by a (scheme/value) pair with a specific dispatch_mode, either on start or on_receive, as described in subclause 9.1.

The processing model varies depending on *dispatch_mode*.

7.2 Common process

The DASH Player implements the following process:

1. Parse the emsg/timed metadata sample and retrieve *scheme_uri*/(*value*).

2. If Application is not subscribed to the *scheme_uri*/(*value*) pair, end the processing of this emsg.

7.3 on-receive processing

The DASH Player implements the following process when *dispatch_mode* = *on_receive*:

- Dispatch the event/timed metadata, including *ST*, *id*, *DU*, *timescale,* and *message_data* as described in subclause 9.2.

7.4 on-start processing

The DASH Player sets up an Active Event Table for each subscribed *scheme_uri*/(*value*) in the case of *dispatch_mode* = *on_start*. The *Active Event Table* maintains a single list of emsg's *id* that have been

dispatched.

The DASH Player implements the following process when *dispatch_mode* = *on_start*:

1. Derive the event instance/metadata sample's *ST*
2. If the current media presentation time value is smaller than *ST*, then go to Step 5.
3. Derive the ending time *ET*= *ST* + *DU*.
4. If the current presentation time value is greater than *ET*, then end processing.
5. In the case of event: Compare the event's *id* with the entries of the Active Event Table of the same *scheme_uri*/(*value)* pair:
    - If an entry with the identical *id* value exists, end processing;
    - If not, add emsg's *id* to the corresponding the Active Event Table.
6. Dispatch the event/metadata *message_data* at time *ST*, or immediately if the current presentation time is larger than *ST*, as described in subclause 9.2.

8 Event/metadata buffer model

Along with the media samples, the event instances and timed metadata samples are buffered. The event/metadata buffer should be managed with the same scheme as the media buffer, i.e. as long as a media sample exists in the media buffer, the corresponding events and/or metadata samples should be maintained in the event/metadata buffer.

9 Prose description of APIs

The event/timed metadata API is an interface defined between a "DASH client" as defined in 3GPP TS 26.247 or ISO/IEC 23009-1 and a device application in the exchange of subscription data and dispatch/transfer of matching DASH Event or timed metadata information between these entities. The Event/timed metadata API is shown in Figure 1.

Note: In this section, the term "DASH Player" is used.

The description of the API below is strictly functional, i.e. implementation-agnostic, is intended to be employed for the specification of the API in Javascript for the dash.js open source DASH Player, and in IDL such as the OMG IDL or WebIDL. For example, the subscribeEvent() method as defined below may be mapped to the existing **on(type, listener, scope)** method as defined for the dash.js under **MediaPlayerEvents**.

As part of this API and prior to any operations, the DASH Player provides a list of *scheme_id*/(*value*) listed in the MPD when it receives it. This list includes all MPD and inband events as well as *scheme_id* of all timed metadata tracks. At this point, the Application is aware of the possible events and metadata deliverable by the DASH Player.

9.1 Event and metadata track subscription

The subscription state diagram of DASH Player associated with the API is shown below in Figure 6 :

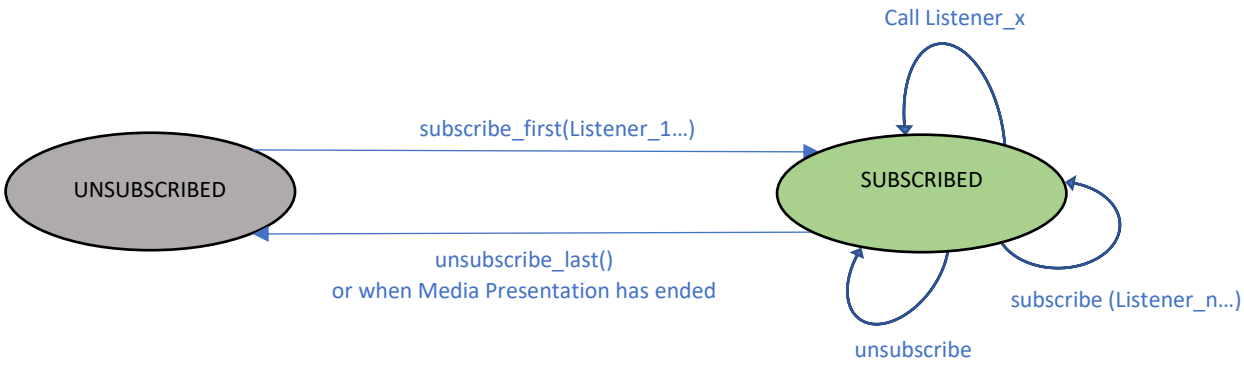

Figure 6 State Diagram of DASH Player for the event/timed metadata API

The scope of the above state diagram is the entire set of applicable events/timed metadata streams being subscribed/unsubscribed, i.e. it is not indicating the state model of DASH Player in the context of a single Event/timed metadata stream subscription/un-subscription.

The application subscribes to the reception of the desired event/timed metadata and associated information by the **subscribeEvent()** method. The parameters to be passed in this method are:

- *scheme_uri* – The scheme identifier for the event stream being subscribed to. This must be one returned by the list of events the DASH player supplied. By setting this value to urn:mpeg:dash:event:catchall:2020, the Application may subscribe to all existing events and metadata schemes described in the MPD. In this case, the value of *value* is irrelevant.

- *value* – A value of the event or timed metadata stream within the scope of the above *scheme_uri*, optional to include. When not present, no default value is defined – i.e., no filtering criterion is associated with the Event scheme identification.

- *dispatch_mode* – Indicates when the event handler function identified in the *callback_function* argument should be called:

    - *dispatch_mode* = *on_receive* – provide the event/timed metadata sample data to the Application as soon as it is detected by DASH Player;

    - *dispatch_mode* = *on_start* – provide the event/timed metadata sample data to the App at the start time of Event message or at the presentation time of timed metadata sample.

  The default mode for *dispatch_mode* should be set to *on_receive*, i.e. if the dispatch_mode is not passed during the subscribe_first operation, DASH Player should assume *dispatch_mode* = *on_receive* for that specific subscription.

- *callback_handler* – the name of the function to be (asynchronously) called for an event corresponding to the specified *scheme_uri*/(*value*). The callback function is invoked with the arguments described below.

Note: ISO/IEC 23009-1 does not include any explicit signaling for the desired dispatch mode in MPD or timed metadata track. In the current design, an Application relays its desired dispatch mode to DASH Player when it subscribes to an event stream or timed metadata track. In this approach, the scheme owner should consider the dispatch mode as part of the scheme design and define whether any specific dispatch mode should be selected during the design of the scheme.

9.2 Invoking the call back function

Upon successful execution of the event/timed metadata subscription call, the DASH Player shall monitor the source of potential Event stream information, i.e., the MPD or incoming DASH Segments, for matching values of the subscribed *scheme_uri*/(*value*). The parentheses around value are because this parameter may be absent in the event/timed metadata subscription call. When a matching event/metadata sample is detected, DASH Player invokes the function specified in the callback Function argument with the following parameters. It should additionally provide to the Application the current presentation time at DASH Player when performing the dispatch action. The parameters to be passed in this method are shown in Table 1 below:

Table 1 Event/timed metadata API parameters and datatypes

API Parameter	MPD event	Inband emsg	Metadata	Data Type	'on-receive'	'on-start'
scheme_id	EventStream@schemeIdUri	scheme_id_uri	timed metadata track URI	string	Y	Y
value	EventStream@value	value		string	Y	Y
presentation_time	Event@presentationTime	presentation_time	timed metadata sample presentation time	unsigned int(64) in milliseconds	Y	N
duration	Event@duration	event_duration	timed metadata sample duration	unsigned int(32) in milliseconds	Y	N
id	Event@id	id		unsigned int(32)	Y	N
message_data	Event@messageData	message_data()	timed metadata sample data in mdat	unsigned int(8) x messageSize	Y	Y
Y= Yes, N= NO, O= Optional						

When the duration of the event is unknown, the variable *duration* shall be set to its maximum value

(xFFFFFFFF = 4,294,967,295).

Note: In the case of 'emsg' version 0, DASH Player is expected to calculate presentation_time from presentation_time_delta.

In order to remove a listener the **unsubscribeEvent()** function is called with the following arguments:

- *scheme_uri* - A unique identifier scheme for the associated DASH Event stream of interest to the Application.
- *value*
- *callback_handler*

If a specific listener is given in the *callback_function* argument, then only that listener is removed for the specified *scheme_uri*/(*value*). Omitting or passing null to the *callback_function* argument would remove all event listeners for the specified *scheme_uri*/(*value*).

9.3 Detailed processing of events and metadata samples

As shown in Figure 1, the event/metadata buffer holds the events or metadata samples to be processed. We assume that this buffer has the same data structure to hold events or metadata. We use Table 1 to define this Event/Metadata Internal Object (EMIO), shown in Table 2:

Table 2 Event/Metadata Internal Object (EMIO)

```
event-metadata-internal-object {
        string              scheme_id_uri;
        string              value;
        unsigned int(32)    presentation_time;
        unsigned int(32)    duration;
        unsigned int(32)    id;
    unsigned int(8)         message_data();
}
```

The process for converting the received event/metadata sample to EMIO is as following:

1. For MPD event
    a. For each period
        i. Parse each EventStream
        ii. Get Eventstream common parameters
        iii. For each Event Stream:
            1. Parse each event
            2. For each event

a. Calculate presentation time and event duration
b. Add it to EMIO

2. For inband event
    a. For each Segment
        i. Parse event boxes as well as moof
        ii. Calculate EPT of the segment
        iii. For each event:
            1. Map emsg box parameters to EMIO
3. For simple metadata samples
    a. For each Segment
        i. Parse moof
        ii. For each sample:
            1. Parse the format
            2. map the data to EMIO

10 DASH Event Metadata WebIDL API

10.1 Abstract

This section specifies a WebIDL API that a user agent or DASH client can expose for application access to DASH events. This builds upon Media Source Extensions.

10.2 DASHEvent Interface

```
[Constructor(SourceBuffer source)]
interface DASHEvent : EventTarget {
    readonly        attribute EventData             eventData;
                    attribute EventHandler          ondashevent;
    Promise         setEvents(Eventlist eventList);
};
```

10.2.1 Attributes

`eventData` of type `EventData`, readonly

When an event is encountered, the DASH client *MUST* extract the event data, and *MUST* initialize the object's `eventData` attribute to a string representation of the event data.

`ondashevent` of type EventHandler

This event handler is invoked when a new DASH event arrives.

`setEvents()` of type Promise

This promise must include an `eventList` argument that enumerates all events in which the application is interested.

10.3 EventData Interface

```
interface EventData {
    readonly        attribute DOMString schemeIdURI;
    readonly        attribute DOMString value;
    readonly        attribute DOMTimeStamp? presentationTime;
    readonly        attribute unsigned long? duration;
    readonly        attribute unsigned long? id;
    readonly        attribute ByteString messageData;
};
```

10.3.1 Attributes

`schemeIdURI` of type DOMString, readonly

The `schemeIDURI` attribute *MUST* return a URI that identifies the DASH event scheme.

`value` of type DOMString, readonly

The `value` attribute *MUST* return the value for the event stream element. The value semantics are defined by the owners of the scheme identified in the `schemeIdUri` attribute.

`presentationTime` of type DOMTimeStamp, readonly

The `presentationTime` attribute *MUST* return a value corresponding to the exact moment in the media presentation timeline that the event becomes active. If this attribute is not present then its value shall be set to NULL and the event is assumed to be active immediately.

`duration` of type int, readonly

The `duration` attribute *MUST* return the time for which the event is in effect starting from presentationTime. The value of the duration is in milliseconds. If this attribute is not present then its value must be set to the maximum value (4294967295) and the event *MUST* be persisted until another DASH event is received.

`id` of type int, readonly

The `id` attribute *MUST* return an identifying value for this event. If this value is not present then its value must be set to NULL.

`messageData` of type ByteString, readonly

The `messageData` attribute *MUST* return the event message data payload.

10.4 EventList Interface

```
dictionary EventList {
    DOMString[]             desiredSchemeIdURI;
```

```
    optional DOMString[]    value;
    optional boolean[]      dispatchMode;
};
```

`EventList` contains one or more valid event scheme URI's along with associated parameters.

10.4.1 **Members**

`desiredSchemeIdURI` of type DOMString[]

> `desiredSchemeIdURI` is an array of valid DASH event scheme URI's. If the `desiredSchemeURI` array is set to NULL, then all events will be sent to the handler.

`value` of type DOMString[]

> `value` is an array of valid values for events. If the `value` array has only one member, then that value will be applied to all scheme URI's. If more than one member is present in the array then those values will be matched to each member in `desiredSchemeIdURI` in order. A `value` array with more members than the `desiredSchemeIdURI` array should be rejected if the `desiredSchemeIdURI` array is non-NULL. In the case of a non-NULL `desiredSchemeIdURI` array, members of the `value` array can be set to NULL if all values are acceptable for the associated Scheme URI. If the `desiredSchemeIdURI` array is NULL, then the `value` array can have only one member.

`dispatchMode` of type boolean[]

> `dispathMode` is an array of dispatch mode settings for events (true indicating a dispatch on receipt of the event, false representing a dispatch on the start of the event) . If the `dispatchMode` array has only one member, then that dispatch mode will be applied to all scheme URI's. If more than one member is present in the array then those values will be matched to each member in `desiredSchemeIdURI` in order. A `dispatchMode` array with more members than the `desiredSchemeIdURI` array should be rejected if the `desiredSchemeIdURI` array is non-NULL. In the case of a non-NULL `desiredSchemeIdURI` array, members of the `dispatchMode` array can be set to NULL if both dispatch modes are acceptable for the associated Scheme URI. If the `desiredSchemeIdURI` array is NULL, then the `dispatchMode` array can have only one member.

10.5 **Example**

```
<html>
<body>
<script>
  function onSourceOpen(videoTag, e) {
    var mediaSource = e.target;
    if (mediaSource.sourceBuffers.length > 0)
        return;
    try {
        dashevent = new DashEvent(mediaSource);
        dashevent.setEvents(["schemeURI1","schemeURI2"]).then(
            {
```

```
            console.log('Desired event list set');
            }
        }
        catche (e)
            {
            console.error('Failed to create Dash event handler due to:  ' +
e);
            return;
            }
    dashevent.ondashevent = dashEventHandler;
    function dashEventHandler(event){
        }
    }
</script>
<video id="v" autoplay> </video>
<script>
  var video = document.getElementById('v');
  var mediaSource = new MediaSource();
  mediaSource.addEventListener('sourceopen', onSourceOpen.bind(this,
video));
  video.src = window.URL.createObjectURL(mediaSource);
</script>
</body>
</html>
```