DASH Player’s Application Events and Timed Metadata Processing Models and APIs

Living Document, 31 May 2019

This version:
https://dashif.org/guidelines/name-of-doc

Issue Tracking:
GitHub

Editors:
DASH Industry Forum

Table of Contents

#External Defintion - (to be removed)

`SegmentBase`

`PresentationTimeOffset` `timescale`

Presentation time offset time scale cmaf

1. DASH Player architecture for processing DASH events and timed metadata tracks

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

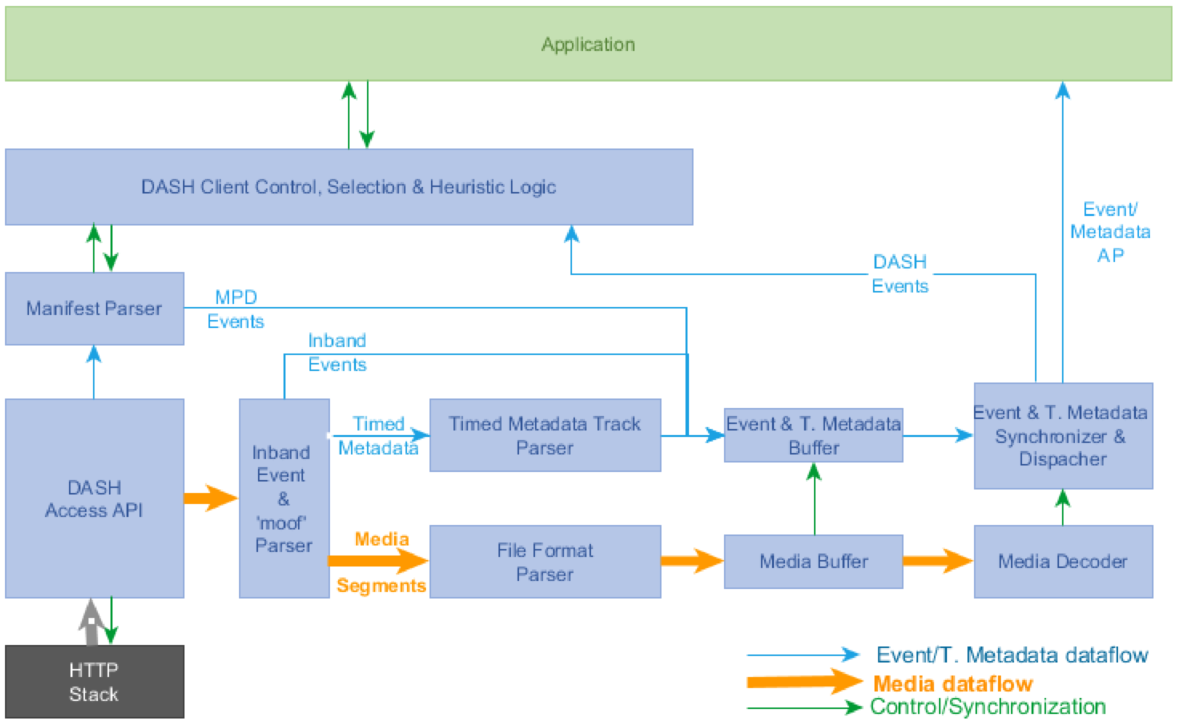

***Figure 1** DASH Player architecture including the inband Event and Application-related timed metadata handling*

In the above figure:

1. DASH Player processes the MPD. If the manifest includes any MPD Events, it parses them and appends them to the Event & Timed Metadata Buffer.
2. Based on the MPD, DASH Player manages fetching and parsing the Segments before appending them into the media decoder input buffer (named 'Media Buffer' in the Figure 1).
3. Parsing a Segment includes:
    1. parse the high-level boxes such as Segment Index (sidx) and Event Message boxes, and append Event Message boxes to the Event & Metadata Buffer.
    2. For an Application-related timed metadata track, extracting the data samples, and appending them to the Event & Metadata Buffer.
    3. For media segments, parse the segments and append them to Media Buffer.
4. The DASH Player-specific Events are passed to DASH Player control function (named 'DASH Client Control, Selection & Heuristic Logic' in Figure 1), while the Application-related Events and timed metadata track samples are passed to the Event & Metadata Synchronizer and Dispatcher function.
5. If an Application is subscribed to a specific Event or timed metadata stream, dispatch the corresponding event instances or timed metadata samples, according to the dispatch mode:
    1. For on-receive dispatch mode, dispatch the Event information or timed metadata samples as soon as they are received(or no later than *AT*).
    2. For on-start dispatch mode, dispatch the Event information or timed metadata samples at their associated presentation time, using the synchronization signal from the media decoder.

2. Event and Timed metadata sample timing models§

2.1. Inband Event timing parameters§

Figure 2 presents the timing of an inband Event along the media timeline:

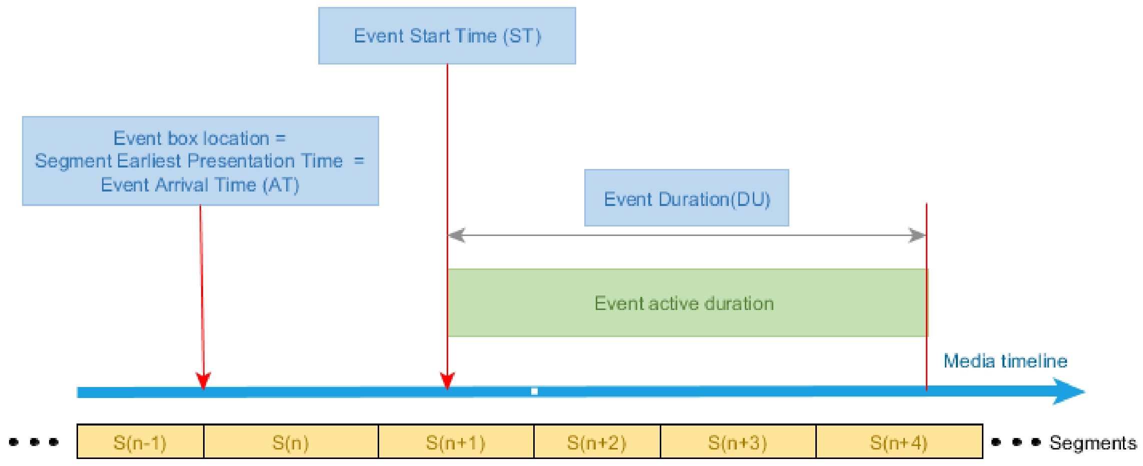

Figure 2 *The inband event timing parameter on the media timeline*

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

1. Event Arrival Time (*AT*) which is the earliest presentation time of the Segment containing the Event Message box.
2. Event Presentation/Start Time (*ST*) which is the moment in the media timeline that the Event becomes active.
3. Event duration (*DU*): the duration for which the Event is active

An inband Event is inserted in the beginning of a Segment. Since each media segment has an earliest presentation time equal to (*AT*), *AT* of the Segment carrying the Event Message box can be considered as the location of that box on the media timeline. DASH Player has to fetch and parse the Segment before or at its *AT* (at *AT* when it’s assumed that the decoding and rendering of the segment incurs practically zero delay). Therefore, the Event inserted in a Segment at its *AT* time will be ready to be processed and fetched no later than *AT* on the media timeline.

The second timing parameter is Event Presentation/Start Time (*ST*). *ST* is the moment in the media timeline that the Event becomes active. This value can be calculated using the parameters included in Event Message box.

The third parameter is Event Duration (*DU*), the duration for which the Event is considered to be active. *DU* is also signaled in the Event Message box using a specific value.

2.2. Event message box format and event timing parameters§

Table 1 shows the emsg box format in DASH:

```
aligned(8) class DASHEventMessageBox extends FullBox ('emsg', version, flags = 0){
 if (version==0) {
  string                                   scheme_id_uri;
  string                                   value;
  unsigned int(32)                         timescale;
  unsigned int(32)                         presentation_time_delta;
  unsigned int(32)                         event_duration;
  unsigned int(32)                         id;
 } else if (version==1) {
  unsigned int(32)                         timescale;
  unsigned int(64)                         presentation_time;
  unsigned int(32)                         event_duration;
  unsigned int(32)                         id;
  string                                   scheme_id_uri;
  string                                   value;
 }
 unsigned int(8)                           message_data();
}
```

Figure 3 *The emsg box format and parameters*

The *ST* of an event can be calculated using values in its emsg box:

$$ST = \begin{cases} AT + \frac{presentation\_time\_delta}{timescale} \space \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad version=0\\ PeriodStart + \frac{SegmentBase@presentationTimeOffset}{SegmentBase@timescale} + \frac{presentation\_time}{timescale}\qquad \qquad version=1 \end{cases} $$

Figure 4 *Event Start Time of an inband event*

Where *PeriodStart* is the corresponding Period's start time, and `presentationTimeoffset` and `timescale` are the Presentation Time Offset (PTO) and time scale of the corresponding Represenation.

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

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

Note: If various Adaptation Sets carry the same events, different Adaptation Sets/Representations with different PTOs, the presentation_time_delta and/or presentation_time values might be different per Adaptation Set/Representation, i.e. the same emsg box can not be replicated over multiple Representations and/or Adaptations Sets.

Note: In the case of CMAF, *PeriodStart* is the CMAF track's earliest presentation time.

In this document, 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()

2.3. MPD Events timing model§

MPD Events carry the similar data model as inband Events. However, the former type is are carried in the MPD, under the Period elements. Each Period event has `<EventStream>` element(s), defining the `schemeIdUri`, `value` , `timescale` and a sequences of `<Event>` elements. Each event may have `presentationTime`, `duration`, `id` and `messageData` attributes, as shown in Table 2.

Element or Attribute Name	Use	Description
`EventStream`		specifies event Stream
`@xlink:href`	O	specifies a reference to an external **`EventStream`** element
`@xlink:actuate`	OD default: `onRequest`	specifies the processing instructions, which can be either "`onLoad`" or "`onRequest`". This attribute shall not be present if the `@xlink:href` attribute is not present.
`@schemeIdUri`	M	identifies the message scheme. The string may use URN or URL syntax. When a URL is used, it is recommended to also contain a month-date in the form mmyyyy; the assignment of the URL must have been authorized by the owner of the domain name in that URL on or very close to that date. A URL may resolve to an Internet location, and a location that does resolve may store a specification of the message scheme.
`@value`	O	specifies the value for the event stream element. The value space and semantics must be defined by the owners of the scheme identified in the `@schemeIdUri` attribute.
`@timescale`	O	specifies the timescale in units per seconds to be used for the derivation of different real-time duration values in the **`Event`** elements. If not present on any level, it shall be set to 1.
`@presentationTimeOffset`	OD Default: 0	specifies the presentation time offset of this Event Stream that aligns with the start of the Period. Any Event contained in this Event Stream is mapped to the Period timeline by using the Event presentation time corrected by the value of the presentation time offset. The value of the presentation time offset in seconds is the division of the value of this attribute and the value of the @timescale attribute.
`<Event>`	0 ... N	specifies one event. For details see Table 5.31. Events in Event Streams shall be ordered such that their presentation time is non-decreasing.

Legend:

For attributes: M=Mandatory, O=Optional, OD=Optional with Default Value, CM=Conditionally Mandatory.

For elements: <minOccurs>...<maxOccurs> (N=unbounded)

Elements are `bold`; attributes are non-bold and preceded with an @.

Element or Attribute Name	Use	Description
`Event`		specifies an event and contains the message of the event, formatted as a string. The content of this element depends on the event scheme.
@`presentationTime`	OD default: 0	specifies the presentation time of the event relative to the start of the Period. The value of the presentation time in seconds is the division of the value of this attribute and the value of the `@timescale` attribute. If not present, the value of the presentation time is 0.
@`duration`	O	specifies the presentation duration of the event. The value of the duration in seconds is the division of the value of this attribute and the value of the `@timescale` attribute. If not present, the value of the duration is unknown.
@`id`	O	specifies an identifier for this instance of the event. Events with equivalent content and attribute values in the **`Event`** element shall have the same value for this attribute. The scope of the @id for each Event is with the same `@schemeIdURI` and `@value` pair.
@`contentEncoding`	O	specifies if the information in the body and the information in the @messageData is encoded. If present, the following values are possible: "base64" the content is encoded as described in IETF RFC 4648 prior to adding it to the field. If this attribute is present, the DASH client is expected to decode the message data and only provide the decoded message to Application.
@`messageData`	O	specifies the value for the event stream element. The value space and semantics must be defined by the owners of the scheme identified in the `@schemeIdUri` attribute. NOTE: this attribute is an alternative to specifying a complete XML element(s) in the Event. It is useful when an event leans itself to a compact string representation

Legend:

For attributes: M=Mandatory, O=Optional, OD=Optional with Default Value, CM=Conditionally Mandatory.

For elements: <minOccurs>...<maxOccurs> (N=unbounded)

Elements are `bold`; attributes are non-bold and preceded with an @.

***Figure 5** MPD Event elements*

As is shown in Figure 3, each MPD Event has three associated timing parameters along the media timeline:

1. The PeriodStart Time (*AT*) 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 attribute <{Event@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 `timescale` to scale the above parameters.

Figure 3 demonstrates these parameters in the media timeline.

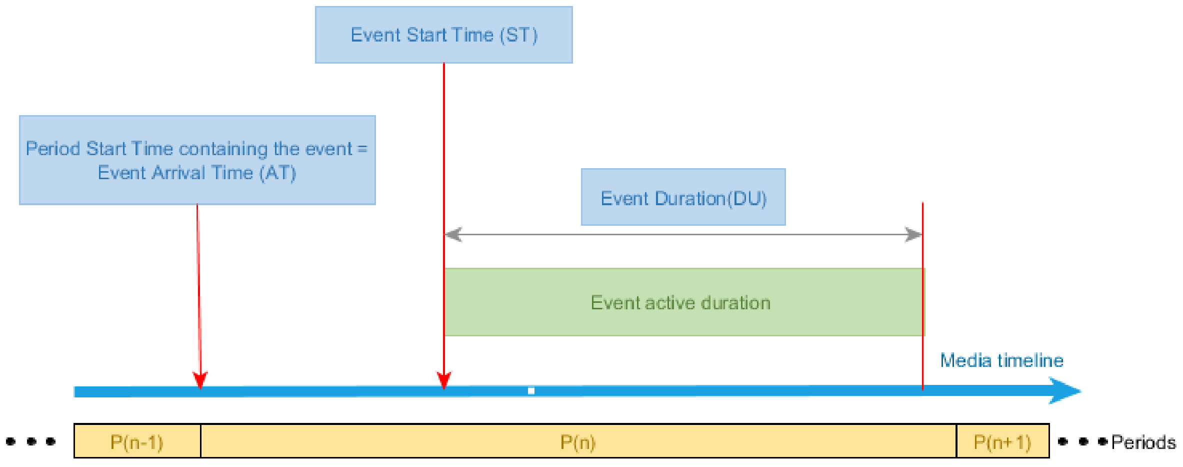

***Figure 6** MPD events timing model*

The *ST* of an MPD event can be calculated using values in its `<EventStream>` and `<Event>` elements:

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

***Figure 7** Event Start Time of MPD event*

In this document, 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* = `schemeIdUri`
- *value* = `value`
- *presentation_time* = *ST*
- *duration* = `duration`/`timescale`
- *id* = `id`

- *message_data* = decode64(messageData)

In which decode64() function is:

$$decode64(x) = \begin{cases} x\space\qquad\qquad\qquad\qquad\qquad \space \space \space \space @contentEncoding\space Not \space Present\\ base64 \space decoding \space of \space (x) \qquad @contentEncoding \space = \space base64 \end{cases} $$

Figure 8 *decode64 function*

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

2.4. Timed metadata sample timing model§

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

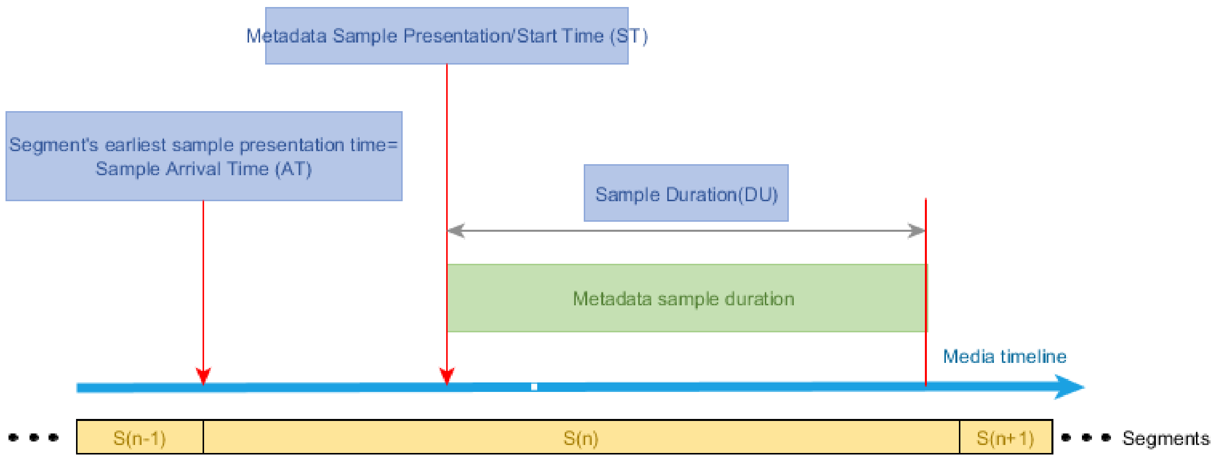
Metadata Sample Presentation/Start Time (ST)
Segment's earliest sample presentation time= Sample Arrival Time (AT)
Sample Duration(DU)
Metadata sample duration
Media timeline
S(n-1)
S(n)
S(n+1)
Segments

Figure 9 *Timing parameters of a timed metadata sample on the media timeline*

As shown in this figure, the metadata sample timing including metadata sample presentation time (*ST*) and metadata sample duration (*DU*). Also one or multiple metadata samples are included in a segment with Segment start time (*AT*).

Note that the metadata sample duration can not go beyond segment duration, i.e. to the next segment. In the case of CMAF, the same constraints is maintained for CMAF Chunks.

In this document, 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* = **timed metadata track URI**
- *timescale* = timed metadata track timescale in mdhd box.
- *presentation_time* = **timed metadata sample presentation time**/*timescale*
- *duration* = **timed metadata sample duration**/*timescale*
- *message_data* = **timed metadata sample data in mdat**

3. Events and timed metadata sample dispatch timing modes§

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

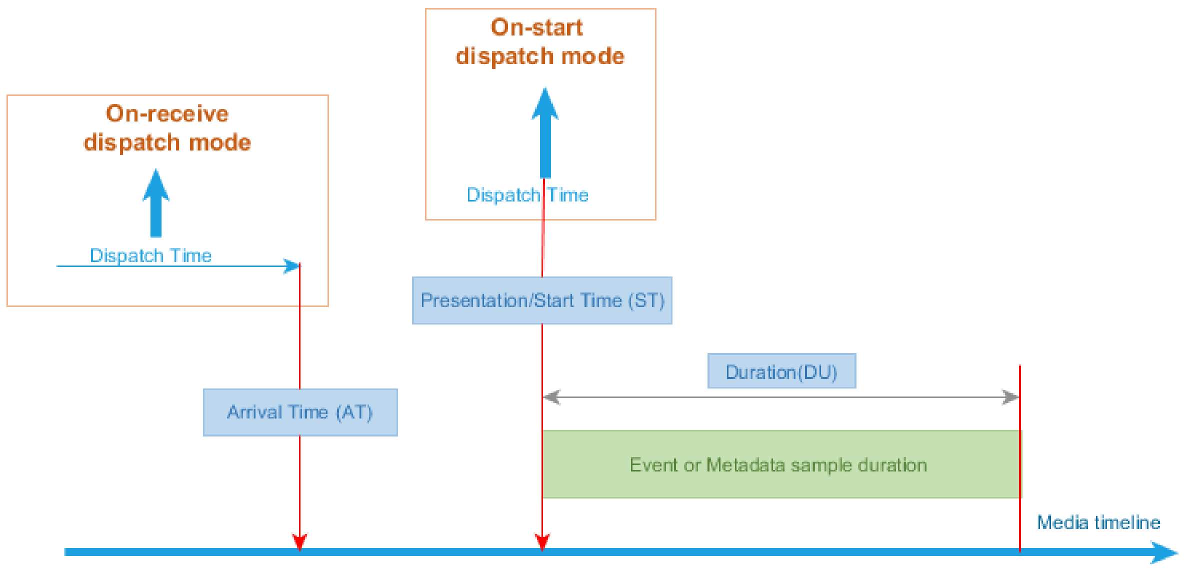

Figure 10 *The Application events and timed metadata dispatch modes*

In this figure, two modes are shown:

1. **on-receive** Dispatch Mode: Dispatching at *AT* or earlier. Since the segment carrying an emsg/metadata sample has to be parsed before (or assuming zero decode/rendering delay as the latest at) *AT* on the media timeline, the event/metadata sample shall be dispatched at this time or before to Application in this mode. Application has a duration of *ST-AT* for preparing for the event. In this mode, the client doesn't need to maintain states of Application events or metadata samples either. Application may have to maintain the state for any event/metadata sample, its *ST* and *DU*, and monitor its activation duration, if it needs to. Application also needs to schedule each event/sample at its *ST*, so it must be time-aware to properly make use of these timing parameters.
2. **on-start** Dispatch Mode: Dispatching exactly at *ST*, which is the start/presentation time of the event/metadata sample. DASH Player shall calculate the *ST* for each parsed event/metadata sample and dispatch the *message_data* at this exact moment. In this mode, since Application receives the event/sample at its start/presentation time, it needs to act on the received data right away, i.e. no advanced notice is given to Application in this mode. Application however may not need to maintain a state for the events and timed metadata samples, if the durations and/or the sequence and order of events/samples are not important to Application. Depending on the nature, meaning and relationship between different event instances/metadata samples, Application may need to maintain the state for them.

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 disp atch the event exactly once. In the nromal playback, the player dispatches the event at *ST*. However if DASH Player for instance seek to a moment after *ST* and during the above duration, then it must dispatch the event immidiately. In the case of on-receive, *duration* is a property of event instance and is defined by the *scheme_id* owner.

3.1. The Dispatch Processing Model§

3.1.1. Prerequisite§

Application subscribes to specific event stream as described in §4 Prose description of APIs.

The processing model varies depending on *dispatch_mode*.

DASH Player shall follow the processing model outlined in this section.

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

3.1.2. Common process§

DASH Player shall implement 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.

3.1.3. on-receive processing§

DASH Player shall implement the following process when *dispatch_mode* = *on_receive*:

- Dispatch the event/timed metadata, including *ST*, *id*, *DU*, *timescale* and *message_data* as described in §4 Prose description of APIs.

3.1.4. on-start processing§

DASH Player shall implement the following process when *dispatch_mode* = *on_start*:

1. Derive the event instance/metadata sample's *ST*
2. If the current presentation time value is smaller than C, 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 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 Active Event Table.
6. Dispatch the event/metadata *message_data* at time *ST*, or immediately if current presentation time is larger then *ST*, as described in §4 Prose description of APIs.

3.2. The 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 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.

4. Prose description of APIs§

The event/timed metadata API is an interface defined between a “DASH player” as defined in DASH-IF, or 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 at Figure 1.

Note: In this document, 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, DASH Player provides a list of *scheme_id*/(*value*) listed in MPD when it receives it. This list includes all events as well as *scheme_id* of all timed metadata tracks. At this point Application is aware of the possible events and metadata delivered by DASH Player.

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

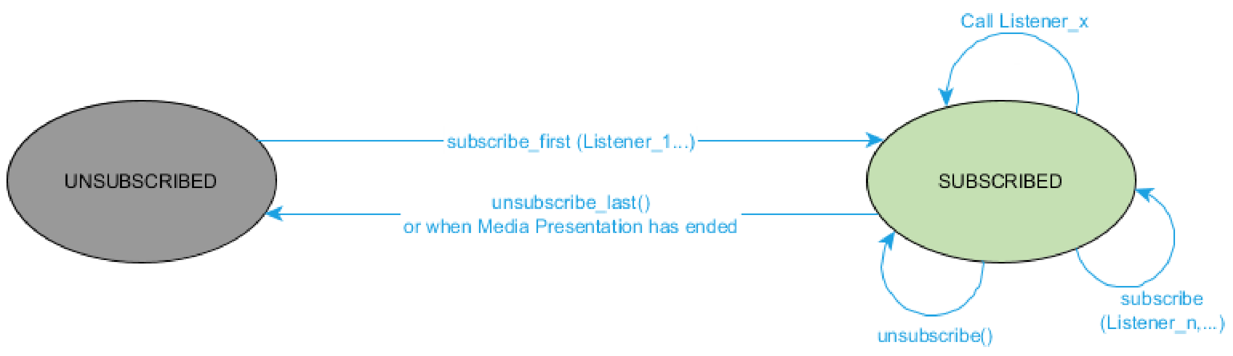

Figure 11 *State Diagram of DASH Player for the event/timed metadata API.*

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:

- *app_id* – (Optional) A unique ID for the Application subscribing to data dispatch from DASH Player. Depending on the platform/implementation this identifier may be used by DASH Player to maintain state information.
- *scheme_uri* – A unique identifier scheme for the associated DASH Event/metadata stream of interest to the Application. This string may use a URN or a URL syntax, and may correspond to either an MPD Event, an inband Event, or a timed metadata stream identifier. The *scheme_uri* may be formatted as a regular expression (regex). If a value of NULL is passed for *scheme_uri*, then Application subscribes to all existing event 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.
- *callback_function* – 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 amy explicit signaling for the desired dispatch mode in MPD or timed metadata track. In the current design, Application relay 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.

Note: (Editor’s Note-to be removed at the end of Community Review Period) If any service provider or application developer beleives an explicit signaling of dispatch mode is needed for some use-cases, they are requested to provide such use-case during Community Review Period of this document to DASH-IF for considering introducing a @dispatchMode attribute in MPD and submitting the request to MPEG.

the DASH-IF beleives an explicit signaling of the dispatch mode is benifitial and will request MPEG to add the support for it. Otherwise, either DASH-IF addes extensions or signaling of the dispatch mode would be considered out-of-band.

Upon successful execution of the event/timed metadata subscription call (for which DASH Player will return a corresponding acknowledgment), 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 is 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 callbackFunction 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 3 below:

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

Figure 12 *Event/timed metadata API parameters and datatypes*

When the duration of the event is unknown, the vairable *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:

- *app_id* (Optional)
- *scheme_uri* - A unique identifier scheme for the associated DASH Event stream of interest to the Application.
- *value*
- *callback_function*

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*).

Conformance§

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

¶EXAMPLE 1
This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

Note, this is an informative note.

Index§

Terms defined by this specification§

References§

Normative References§

[RFC2119]
S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

↑

→