DASH-IF Live Media Ingest Protocol

Technical Specification, 1 February 2022

This version:
https://dashif.org/guidelines/

Issue Tracking:
GitHub

Editor:
DASH-IF Ingest TF

Table of Contents

1. Specification: Live Media Ingest§

1.1. Abstract§

Two closely related protocol interfaces are defined: CMAF Ingest (Interface-1) based on fragmented MP4 and DASH/HLS Ingest (Interface-2) based on DASH and HLS. Both interfaces use the HTTP POST (or PUT) method to transmit media objects from an ingest source to a receiving entity. Smart implementations can implement and support both at the same time. These interfaces support carriage of audiovisual media, timed metadata and timed text. Examples of workflows using these interfaces are provided. In addition, guidelines for synchronization of multiple ingest sources, redundancy and failover are presented. The current version of the protocol is 1.1.

1.2. Copyright Notice and Disclaimer§

2. Introduction§

The main goal of this specification is to define the interoperability points between an ingest source and a receiving entity that typically reside in the cloud or network. This specification does not impose any new constraints or requirements to clients that consume media streams.

Live media ingest happens between an ingest source such as a live encoder and a receiving entity. The receiving entity could be a media packager, streaming origin or a content delivery network (CDN) or another cloud media service. The combination of ingest sources and receiving entities is common in practical video streaming deployments, where media processing functionality is distributed between the ingest sources and receiving entities. Nevertheless, in such deployments, interoperability can sometimes be challenging. This challenge comes from the fact that there are multiple levels of interoperability to be considered and vendors may have a different view of what is expected/preferred as well as how various technical specifications apply. First of all, the choice for the data transmission protocol, and connection establishing and tearing down are important. Handling premature/unexpected disconnects and recovering from failovers are also critical.

A second level of interoperability lies with the media container and coded media formats. MPEG defined several media container formats such as [ISOBMFF] and [MPEG2TS], which are widely adopted and well supported. However, these are general purpose formats, targeting several different application areas. To do so, they provide many different profiles and options. Interoperability is often achieved through other application standards such as those for broadcast, storage or streaming. For interoperable live media ingest, this document provides guidance on how to use [ISOBMFF] and [MPEGCMAF] for formatting the media content.

A third level of interoperability lies in the way metadata is inserted in streams. Live content often needs such metadata to signal opportunities for ad insertion, program information or other attributes like timed graphics or general information relating to the broadcast. Examples of such metadata formats include [SCTE35] markers, which are often found in broadcast streams and other metadata such as ID3 tags [ID3v2] containing information relating to the media presentation. In fact, many more types of metadata relating to the live event might be ingested and passed on to an over-the-top (OTT) streaming workflow.

Fourth, for live media, handling the timeline of the presentation consistently is important. This includes sampling of the media, avoiding timeline discontinuities and synchronizing timestamps attached by different ingest sources such as audio and video. In addition, media timeline discontinuities must be avoided as much as possible during normal operation. Further, when using redundant ingest sources, the ingested streams must be synchronized in a sample accurate manner.

Fifth, in practice multiple ingest sources and receiving entities are often used. This requires that multiple ingest sources and receiving entities work together in a redundant workflow to avoid interruptions when some of the components fail. Well defined failover behavior is important for interoperability.

This document provides a specification for establishing these interoperability points. The approaches are based on known standardized technologies that have been tested and deployed in several large-scale streaming deployments.

To address these interoperability points, two different interfaces and their protocol specifications have been developed. The first interface (CMAF Ingest) mainly functions as an ingest format to a packager or active media processor, while the second interface (DASH/HLS Ingest) works mainly to ingest media presentations to an origin server, cloud storage or CDN. Smart implementations can implement both interfaces at once. With CMAF being used increasingly by both DASH and HLS in practice this would be a preferred implementation option.

§ 4 Media Ingest Workflows and Interfaces (Informative) provides more background and motivation for the two interfaces. We further motivate the specification in this document supporting HTTP/1.1 [RFC7230] and [ISOBMFF].

The document is structured as follows: Section 3 presents the conventions and terminology used throughout this document. Section 4 presents the use cases and workflows related to media ingest and the two interfaces. Section 5 lists the common requirements for both interfaces. Sections 6 and 7 detail Interface-1 and Interface-2, respectively. Sections 8 provides example workflows and Section 9 shows example implementations.

3. Conventions and Terminology§

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

The following terminology is used in the rest of this document:

ABR: Adaptive bitrate.

CMAF chunk: CMAF media object defined in [MPEGCMAF] clause 7.3.2.3.

CMAF fragment: CMAF media object defined in [MPEGCMAF] clause 7.3.2.4.

CMAF header: Defined in [MPEGCMAF] clause 7.3.2.1.

CMAF Ingest: Ingest interface defined in this specification for push-based [MPEGCMAF].

CMAF media object: Defined in [MPEGCMAF]: a CMAF chunk, segment, fragment or track.

CMAF presentation: Logical grouping of CMAF tracks corresponding to a media presentation as defined in [MPEGCMAF] clause 6.

CMAFstream: Byte-stream that follows the CMAF track format structure format defined in [MPEGCMAF] between the ingest source and receiving entity. Due to error control behavior such as retransmission of CMAF fragments and headers, a CMAFstream may not fully conform to a CMAF track file. The receiving entity can filter out retransmitted fragments and headers and restore a valid CMAF track from the CMAFstream.

CMAF track: CMAF media object defined in [MPEGCMAF] clause 7.3.2.2.

connection: A connection setup between two hosts, typically the media ingest source and receiving entity.

DASH Ingest: Ingest interface defined in this specification for push-based DASH.

HLS Ingest: Ingest interface defined in this specification for push-based HLS.

HTTP POST: HTTP command for sending data from a source to a destination.

HTTP PUT: HTTP command for sending data from a source to a destination.

ingest source: A media source ingesting live media content to a receiving entity. It is typically a live encoder but not restricted to this, e.g., it could be a stored media resource.

ingest stream: The stream of media pushed from the ingest source to the receiving entity.

live stream session: The entire live stream for the ingest relating to a broadcast event.

live encoder: Entity performing live encoding of a high quality ingest stream. This can serve as an ingest source.

manifest objects: Objects ingested that represent streaming manifest, e.g., .mpd in DASH and .m3u8 in HLS.

media objects: Objects ingested that represent the media, timed text or other non-manifest objects. Typically, these are CMAF addressable media objects such as CMAF chunks, segments or tracks.

media fragment: Media fragment, combination of MovieFragmentBox ("moof") and MediaDataBox ("mdat") in ISOBMFF structure. This could be a CMAF fragment or chunk. A media fragment may include top-level boxes defined in CMAF fragments such as "emsg", "prft" and "styp". Used for backward compatibility with fragmented MP4.

objects: Manifest objects or media objects.

OTT: Over-the-top.

POST_URL: Target URL of a POST command in the HTTP protocol for posting data from a source to a destination (e.g., /ingest1). The POST_URL is known by both the ingest source and receiving entity. The POST_URL is setup by the receiving entity. The ingest source may add extended paths to signal track names, fragment names or segment names.

publishing_point_URL: Entry point used to receive an ingest stream (e.g., https://example.com/ingest1).

receiving entity: Entity used to receive the media content, receives/consumes an ingest stream.

RTP: Real-time Transport Protocol as specified in [RFC3550].

streaming presentation: Set of objects composing a streaming presentation based on a streaming protocol such as DASH.

switching set: Group of tracks corresponding to a switching set defined in [MPEGCMAF] or an adaptation set defined in [MPEGDASH].

switching set ID: Identifier generated by a live ingest source to group CMAF tracks in a switching set. The switching set ID is unique for each switching set in a live stream session.

TCP: Transmission Control Protocol (TCP) as specified in [RFC793].

baseMediaDecodeTime: Decode time of the first sample in a movie fragment as signaled in the "tfdt" box.

elng: The ExtendedLanguageTag box ("elng") as defined in [ISOBMFF] overrides the language information.

ftyp: The FileTypeBox ("ftyp") as defined in [ISOBMFF].

mdat: The MediaDataBox ("mdat") defined in [ISOBMFF].

mdhd: The MediaHeaderBox ("mdhd") as defined in [ISOBMFF] contains information about the media such as timescale, duration, language using ISO 639-2/T [iso-639-2] codes.

mfra (deprecated): The MovieFragmentRandomAccessBox ("mfra") defined in [ISOBMFF] signals random access samples (these are samples that require no prior or other samples for decoding).

moof: The MovieFragmentBox ("moof") as defined in [ISOBMFF].

nmhd: The NullMediaHeaderBox ("nmhd") as defined in [ISOBMFF] signals a track for which no specific media header is defined. This is used for metadata tracks.

prft: The ProducerReferenceTime ("prft") as defined in [ISOBMFF] supplies times corresponding to the production of associated movie fragments.

tfdt: The TrackFragmentBaseMediaDecodeTimeBox ("tfdt") defined in [ISOBMFF] signals the decode time of the first sample in the movie fragment.

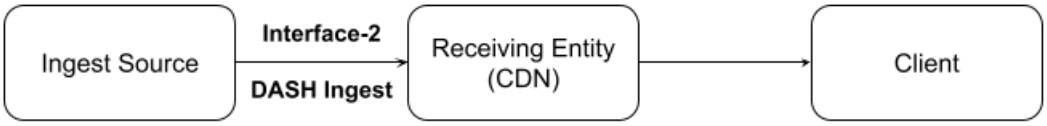
4. Media Ingest Workflows and Interfaces (Informative)§

Two workflows have been identified mapping to two protocol interfaces. The first workflow uses a live encoder as the ingest source and a separate packager as the receiving entity. In this case, Interface-1 (CMAF Ingest) is used to ingest a live encoded stream to the packager, which can perform packaging, encryption or other active media processing. Interface-1 is defined in a way that it will be possible to generate DASH or HLS presentations based on information in the ingested stream. Figure 1 shows an example for Interface-1. In many cases a common implementation is possible.

Figure 1: Example with CMAF Ingest.

The second workflow constitutes ingest to a passive delivery system such as a cloud storage or a CDN. In this case, Interface-2 (DASH Ingest or HLS Ingest) is used to ingest a stream already formatted to be ready for delivery to an end client. Figure 2 shows an example for Interface-2.

Figure 2: Example with DASH Ingest.



A legacy example of a media ingest protocol for the first workflow is the ingest part of the Microsoft Smooth Streaming protocol [MS-SSTR]. Interface-1 (CMAF Ingest, detailed in § 6 Interface-1: CMAF Ingest) improves the Smooth Streaming's ingest protocol including lessons learned over the last ten years after the initial deployment of Smooth Streaming in 2009 and several advances on signaling metadata and timed text. In addition, it includes support for next-generation media codecs such as [MPEGHEVC] and protocols like DASH [MPEGDASH] by adding explicit support for MPEG-DASH Media presentation description.

Interface-2 (DASH/HLS Ingest) is included for ingest of media streaming presentations to a passive receiving entity that provides a pass-through functionality. In this case, manifest objects and other client-specific information also need to be ingested and updated, and segments may be deleted.

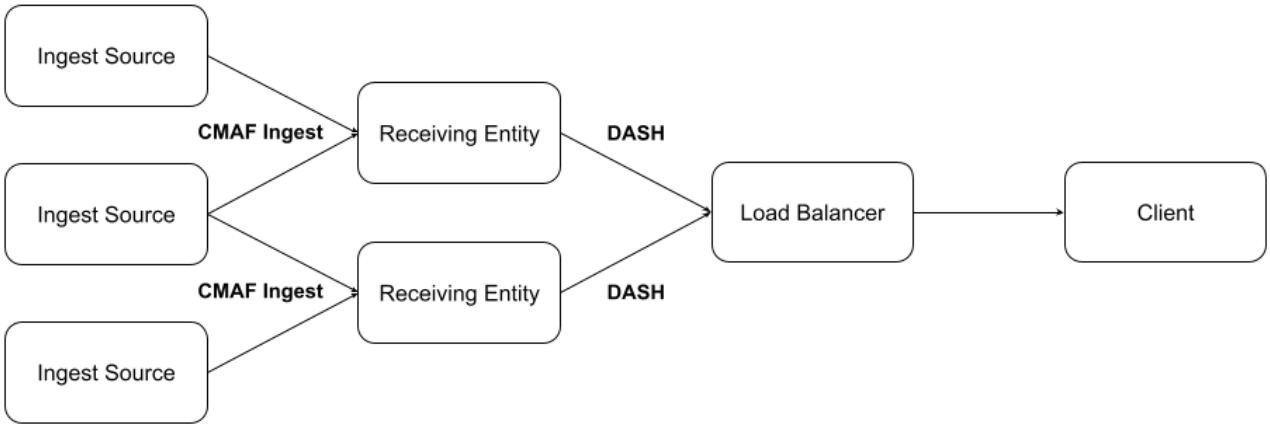
Combining the two interfaces can be considered in many cases. An example of this is given at the end of the document in § 8 Examples (Informative).

Table 1 highlights some of the key differences and practical considerations of the interfaces. In Interface-1, the ingest source can be simple since the receiving entity can do many of the operations related to the delivery such as encryption or generating the streaming manifests. In addition, the distribution of functionalities can make it easier to scale a deployment with concurrent (redundant) live media sources and receiving entities. Besides these factors, choosing a workflow for a video streaming platform depends on many other factors.

Table 1: Different ingest use cases.

Interface	*Ingest source*	*Receiving entity*
CMAF Ingest	Limited overview, simpler encoder, multiple sources	Re-encryption, transcoding, stitching, watermarking, packaging
DASH/HLS Ingest	Global overview, targets duplicate presentations, limited flexibility, no redundancy	Manifest manipulation, transmission, storage

Figure 3: Workflow with redundant ingest sources and receiving entities.



Finally, Figure 3 highlights another aspect that was taken into consideration for large-scale systems with many users. Often content owners would like to run multiple ingest sources, multiple receiving entities and make them available to the clients in a seamless fashion. This approach is already common when serving web pages, and this architecture also applies to media streaming over HTTP. In Figure 3, it is highlighted how one or more ingest sources can be sending data to one or more receiving entities. In such a workflow, it is important to handle the case when one ingest source or receiving entity fails and synchronization. Both the system and client behavior are an important consideration in systems that need to run 24/7. Failovers must be handled robustly and without causing service interruption. This specification details how this failover and redundancy support can be achieved and provides recommendations for dual encoder synchronisation.

5. Common Requirements for Interface-1 and Interface-2§

The media ingest follows the following common requirements for both interfaces.

5.1. Ingest Source Identification§

- The ingest source SHOULD include a User-Agent header (which provides information about brand name, version number and build number in a readable format) in all allowed HTTP messages. The receiving entity can log the received information along with other relevant HTTP header data to facilitate troubleshooting. The version number of the current version is DASH-IF-Ingest 1.1, thus header name may be DASH-IF-Ingest and value may be 1.1

5.2. General Requirements§

1. The ingest source SHALL communicate using the HTTP POST or HTTP PUT as defined in the HTTP protocol, version 1.1 [RFC7230].

   > NOTE: This specification does not imply any functional differentiation between a POST and PUT command. Either may be used to transfer content to the receiving entity. Unless indicated otherwise, the use of the term POST can be interpreted as POST or PUT.

2. The ingest source SHOULD use HTTP over TLS, if TLS is used it SHALL support at least TLS version 1.2, a higher version may also be supported additionally [RFC2818].
3. The ingest source SHOULD use a domain name system for resolving hostnames to IP addresses such as DNS [RFC1035] or any other system that is in place. If this is not the case, the domain name<->IP address mapping(s) MUST be known and static.
4. In the case of 3, ingest source MUST update the IP to hostname resolution respecting the TTL (time-to-live) from DNS query responses. This enables better resilience to IP address changes in large-scale deployments where

the IP address of the media processing entities may change frequently.

5. In case HTTP over TLS [RFC2818] is used, at least one of the basic authentication HTTP AUTH [RFC7617], TLS client certificates or HTTP Digest authentication [RFC7616] MUST be supported.
6. Mutual authentication SHALL be supported. TLS client certificates SHALL chain to a trusted CA or be self-signed. Self-signed certificates MAY be used, for example, when the ingest source and receiving entity fall under the same administration.
7. As compatibility profile for the TLS encryption, the ingest source SHOULD support the Mozilla's intermediate compatibility profile [Mozilla-TLS].
8. In case of an authentication error confirmed by an HTTP 403 response, the ingest source SHALL retry to establish the connection within a fixed time period with updated authentication credentials. When that also results in error, the ingest source can retry N times, after which the ingest source SHOULD stop and log an error. The number of retries N can be configurable in the ingest source.
9. The ingest source SHOULD terminate the HTTP POST or HTTP PUT request if data is not being sent at a rate commensurate with the MP4 fragment duration. An HTTP POST or HTTP PUT command that does not send data can prevent the receiving entity from quickly disconnecting from the ingest source in the event of a service update.
10. The HTTP request for sparse data SHOULD be short-lived, terminating as soon as the data of a fragment is sent.
11. The HTTP request uses the publishing_point_URL at the receiving entity and SHOULD use an additional relative path when posting different streams and fragments, for example, to signal the stream or fragment name.
12. Both the ingest source and receiving entity MUST support IPv4 and IPv6 transport.
13. The ingest source and receiving entity SHOULD support gzip based content encoding.
14. The response from the receiving entity may, in addition to response code, return information in the response body, such as relating to the transfer time, size etc. of the last HTTP request, especially in case this request was in HTTP chunked transfer mode. But no specific response format is defined at this time, but this may be considered in future revisions. NOTE: More specific response body formatting may be defined in future revisions, input from implementors is welcome.
15. The ingest source MUST support the configuration and use of Fully Qualified Domain Names (per RFC 8499) to identify the receiving entity.
16. The ingest source MUST support the configuration of the path, which it will POST all the objects to.
17. The ingest source SHOULD support the configuration of the delivery path that the receiving entity will use to retrieve the content. When provided, the ingest source MUST use this path to build absolute URLs in the manifest files it generates. When absent, use of relative paths is assumed and the ingest source MUST build the manifest files accordingly.
18. The ingest source MUST transfer media objects and manifest objects to the receiving entity via individual HTTP/1.1 POST commands to the configured path.
19. To avoid delay associated with the TCP handshake, the ingest source SHOULD use persistent TCP connections.
20. To avoid head of line blocking, the ingest source SHOULD use multiple parallel TCP connections to transfer the streaming presentation that it is generating. For example, the ingest source SHOULD POST each representation (e.g., CMAF track) in a media presentation over a different TCP connection.
21. The ingest source SHOULD use the chunked transfer encoding option for the HTTP requests when the content length of the request is unknown at the start of transmission or to support the low-latency use cases.

5.3. Failure Behaviors§

1. The ingest source SHOULD use a timeout in the order of a segment duration (e.g., 1-6 seconds) for establishing the TCP connection. If an attempt to establish the connection takes longer than the timeout, the

ingest source aborts the operation and tries again.

2. The ingest source SHOULD resend the objects for which a connection was terminated early or when an HTTP 400 or 403 error response was received if the connection was down for less than three average segments durations. For connections that were down longer, the ingest source can resume sending objects at the live edge of the media presentation.
3. After a TCP error, the ingest source performs the following:

   3a. The current connection MUST be closed and a new connection MUST be created for a new HTTP POST or HTTP PUT request.

   3b. The new HTTP POST_URL MUST be the same as the initial POST_URL for the object to be ingested.
4. In case the receiving entity cannot process the HTTP request due to authentication or permission problems, or incorrect path, it SHALL return an HTTP 403 Forbidden error.
5. The following error conditions apply to the receiving entity:

   5a. If the publishing_point_URL receiving the HTTP request is not available, it SHOULD return an HTTP 404 Not Found error to the ingest source.

   5b. If the receiving entity can process a fragment in the HTTP request body but finds the media type is not supported, it may return an HTTP 415 Unsupported Media Type error.

   5c. If the receiving entity cannot process a fragment in the POST request body due to missing or incorrect initialization fragment, it may return an HTTP 412 Precondition Failed error.

   5d. If there is an error at the receiving entity not particularly relating to the request from the ingest source, it may return an appropriate HTTP 5xx error.

   5e. In all other scenarios, the receiving entity MUST return an HTTP 400 Bad Request error.
6. The ingest source SHOULD support the handling of HTTP 30x redirect responses from the receiving entity.

6. Interface-1: CMAF Ingest§

This section describes the protocol behavior specific to Interface-1. Operation of this interface MUST also adhere to the common requirements given in § 5 Common Requirements for Interface-1 and Interface-2.

6.1. General Considerations (Informative)§

The media format is conforming to the track constraints specified in [MPEGCMAF] clause 7. Note that no CMAF media profile is needed by this specification unless stated otherwise; only the structural format based on [MPEGCMAF] clause 7 is used. Supporting CMAF media profiles is optional.

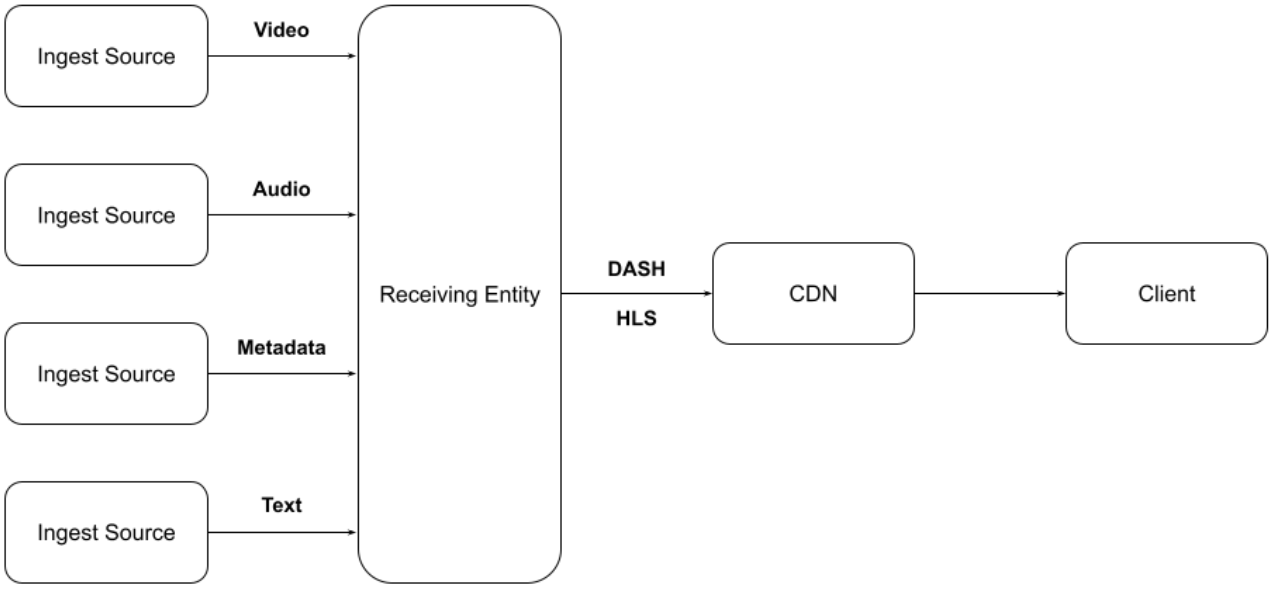
CMAF Ingest can also be used for simple transport of media to an archive, as the combination of CMAF header and CMAF fragments will result in a valid archived CMAF track file when an ingest is stored on disk by the receiving entity.

CMAF Ingest improves over Smooth Streaming's ingest protocol [MS-SSTR] by only using standardized media container formats and boxes based on [ISOBMFF] and [MPEGCMAF] instead of specific UUID boxes.

Many new technologies like MPEG HEVC, AV1, HDR have CMAF bindings. Using CMAF will make it easier to adopt such technologies.

Some discussions on the early development of the specification have been documented in [fmp4git].

Figure 4: CMAF Ingest with multiple ingest sources.



Figures 5-7 detail some of the concepts and structures defined in [MPEGCMAF]. Figure 5 shows the data format structure of the CMAF track. In this format, media samples and media indexes are interleaved. The MovieFragmentBox "moof" box as specified in [ISOBMFF] is used to signal the information to playback and decode properties of the samples stored in the "mdat" box. The CMAF header contains the track specific information and is referred to as a CMAF header in [MPEGCMAF]. The combination of "moof" and "mdat" can be referred as a CMAF fragment or CMAF chunk depending on the structure content and the number of moof-mdat pairs in the addressable object.

Figure 5: CMAF track stream.

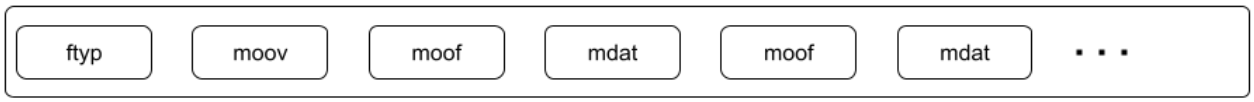


Figure 6 illustrates the presentation timing model, defined in [MPEGCMAF] clause 6.6. Different bit-rate tracks and/or media streams are conveyed in separate CMAF tracks. By having fragment boundaries time aligned for tracks and applying constraints on tracks, seamless switching can be achieved. By using a common timeline different streams can be synchronized at the receiver, while they are in a separate CMAF track, sent over a separate connection, possibly from a different ingest source.

For more information on the synchronization model, we refer the readers to Section 6 of [MPEGCMAF]. For synchronization of tracks coming from different encoders, sample-time accuracy is required, i.e., the samples with identical timestamp contain identical content.

In Figure 7, another advantage of this synchronization model is illustrated, which is the concept of late binding. In the case of late binding, streams are combined on playout/streaming in a presentation (see Section 7.3.6 of [MPEGCMAF]).

> NOTE: As defined in [MPEGCMAF], different CMAF tracks have the same starting time sharing an implicit timeline. A stream becoming available from a different source needs to be synchronized and time-aligned with other streams.

Figure 6: CMAF track synchronization.

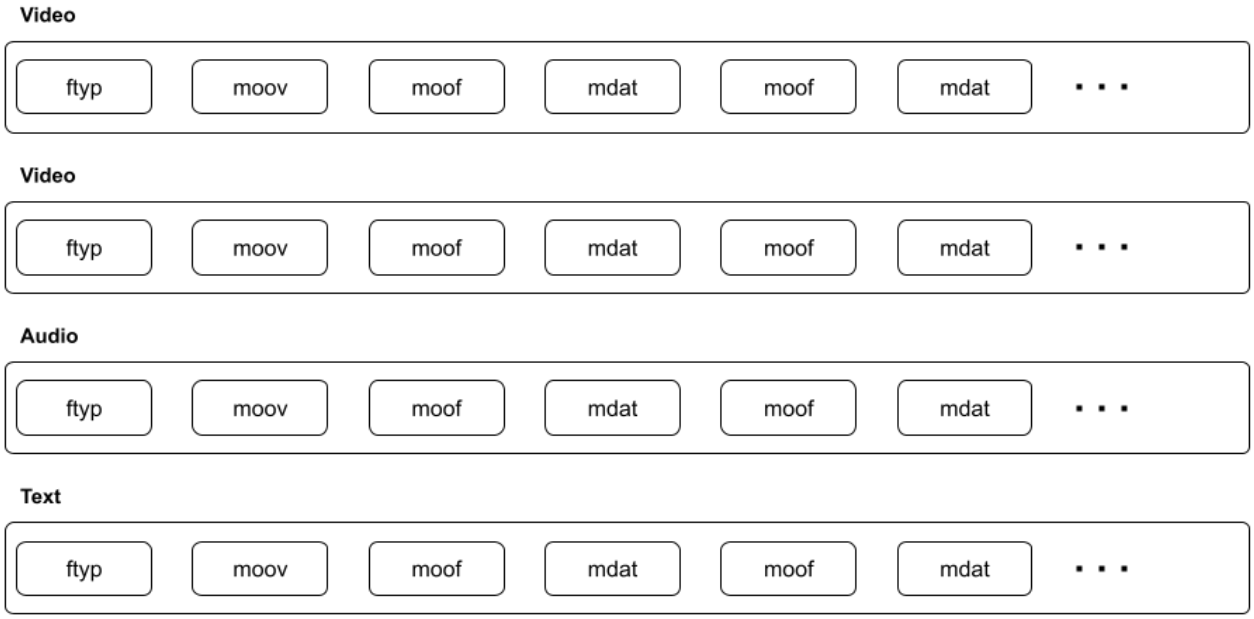


Figure 7: CMAF late binding.

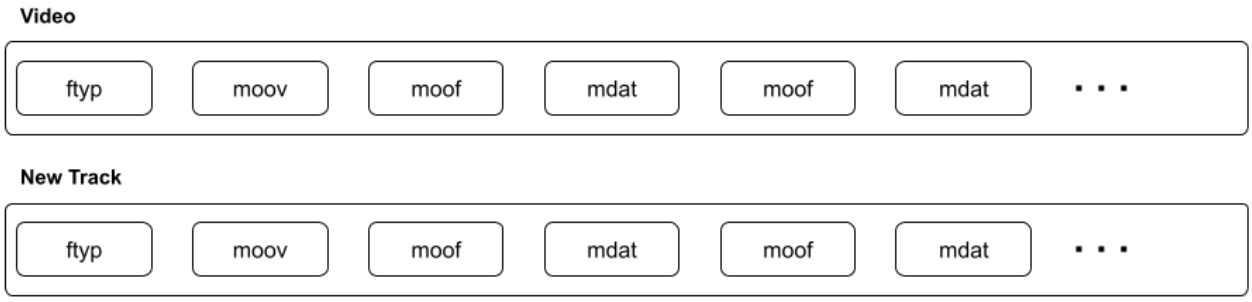
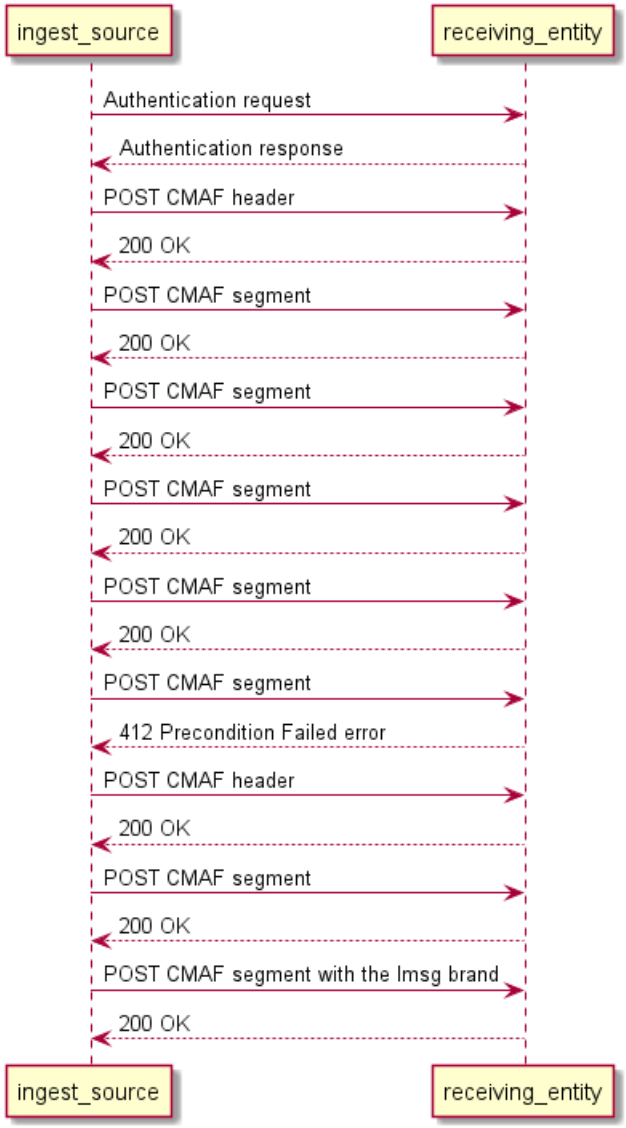


Figure 8 shows the flow diagram of the protocol. It starts with a DNS resolution (if needed) and an authentication step (using two-factor authentication, TLS certificates or HTTP Digest Authentication) to establish a secure TCP connection.

In private datacenter deployments where nodes are not reachable from outside, a non-authenticated connection may also be used. The ingest source then issues an HTTP POST or HTTP PUT request to test that the receiving entity is listening. This request include the CMAF header or could be empty. In case the test is successful, it is followed by the CMAF header and fragments composing the CMAFstream. At the end of the session, the source may send an empty mfra (deprecated) box or a segment with the *lmsg* brand. Then, the ingest source can follow up by closing the TCP connection using a TCP FIN packet.

> NOTE: If the HTTP POST is using the chunked transfer encoding option, the ingest source sends a zero-length terminating chunk per [RFC7230] after sending the *lmsg* brand letting the receiving entity know that the POST command has been concluded.

Figure 8: CMAF Ingest flow.



6.2. General Protocol, Manifest and Track Format Requirements§

The ingest source transmits media content to the receiving entity using HTTP POST or PUT. The receiving entity listens for content at the publishing_point_URL that is known by both the ingest source and receiving entity. The POST_URL may contain an extended path to identify the stream name, switching set or fragment may be added by the ingest source. It is assumed that the ingest source can retrieve these paths and use them.

In Interface-1, the container format is based on CMAF, conforming to the track constraints specified in [MPEGCMAF] clause 7. Unless stated otherwise, no conformance to a specific CMAF media profile is REQUIRED.

1. The ingest source SHALL start by an HTTP POST or HTTP PUT request with the CMAF header, or an empty request, to the POST_URL. This can help the ingest source quickly detect whether the publishing_point_URL is valid, and if there are any authentication or other conditions required.
2. The ingest source MUST initiate a media ingest connection by posting at least one CMAF header after step 1 for each track. Before doing so, it SHOULD post a DASH manifest with a file name extension .mpd to the publishing_point_URL without an additional relative path but the manifest filename and in addition following clause 16 of this section. If not the case, the grouping of the CMAF tracks is trivial and the Streams() keyword is used to identify CMAF tracks.
3. The ingest source SHALL transmit one or more CMAF segments composing the track to the receiving entity once they become available. In this case, a single HTTP POST or PUT request message body MUST contain one CMAF segment.

4. The ingest source MAY use the chunked transfer encoding option of the HTTP POST command [RFC7230] when the content length is unknown at the start of transmission or to support use cases that require low latency.
5. If the HTTP request terminates or times out with a TCP error, the ingest source MUST establish a new connection and follow the preceding requirements. Additionally, the ingest source MAY resend the segment in which the timeout or TCP error occurred.
6. The ingest source MUST handle any error responses received from the receiving entity, as described in general requirements, and by retransmitting the CMAF header.
7. *(deprecated)* In case the live stream session is over the ingest source MAY signal the stop by transmitting an empty mfra (deprecated) box towards the receiving entity. After that it SHALL send an empty HTTP chunk and wait for the HTTP response before closing TCP connection.
8. The ingest source SHOULD use a separate parallel TCP connection for ingest of each different CMAF track.
9. The ingest source MAY use a separate relative path in the POST_URL for ingesting each different track or track segment by appending it to the POST_URL. This makes it easy to detect redundant streams from different ingest sources. Specific naming convention of the segments and paths can be derived from the MPEG-DASH manifest, SegmentTemplate@media and @initialization. If not, the Streams(stream_name) keyword (deprecated) shall be used to signal the name of the cmaf track representation.
10. The baseMediaDecodeTime timestamps in "tfdt" of fragments in the CMAFstream SHOULD arrive in increasing order for each of the fragments in the different tracks/streams that are ingested.
11. The fragment sequence numbers in the CMAFstream signaled in the "mfhd" box SHOULD arrive in increasing order for each of the different tracks/streams that are ingested. Using both baseMediaDecodeTime and sequence number based indexing helps the receiving entities identify discontinuities. In this case sequence numbers SHOULD increase by one.
12. The average and maximum bitrate of each track SHOULD be signaled in the "btrt" box in the sample entry of the CMAF header. These can be used to signal the bitrate later on, such as in the manifest.
13. In case a track is part of a switching set, all properties in Sections 6.4 and 7.3.4 of [MPEGCMAF] MUST be satisfied, enabling the receiver to group the tracks in the respective switching sets.
14. Ingested tracks MUST conform to CMAF track structure defined in [MPEGCMAF]. Additional constraints on the CMAF track structure are defined in later sections for specific media types.
15. CMAF tracks MAY use SegmentTypeBox to signal brands like chunk, fragment or segment. Such signaling may also be inserted in a later stage by the receiving entity.
16. The MPEG-DASH manifest shall use SegmentTemplate in each AdaptationSet (or in each contained Represention). a. The SegmentTemplate@initiatization in the MPEG-DASH manifest shall contain the single substring $RepresentationID$ and the SegmentTempate@media shall contain the single substring $RepresentationID$ and the substring $Number$ or $Time$ (not both). For best interoperability, a separator character should be between representation substrings that is not an integer, this is especially important in case the $RepresentationID$ substitution ends with a number character. b. SegmentTemplate@media shall be identical for each SegmentTemplate Element in the MPEG-DASH manifest. c. SegmentTemplate@initialization shall be identical for each SegmentTemplate Element in the MPEG-DASH manifest. d. The BaseURL element shall be absent. e. The AvailabilityStartTime SHOULD be set to 1-1-1970 (Unix epoch) and the period @start to PT0S (if this is not the case it may be more difficult to synchronize more than one ingest source). f. Each Representation in the MPEG-DASH manifest represents a CMAF track, each AdaptationSet in the MPD represents a CMAF SwitchingSet. g. In case an ingest source issues an HTTP Request with an updated MPEG-DASH manifest, identical naming conventions apply. A receiver may ignore such updated MPD send by an ingest source. h. The MPEG-DASH manifest shall contain a single Period Element.
17. The Ingest source may send an HTTP Live Streaming manifest, but its structure and naming shall be derived from or matching the MPEG-DASH manifest described in clause 16 above. In particular: a. In a master playlist, the groupings identified represent CMAF Switching sets For media playlists named X.m3u8, X shall match the name of the corresponding Representation@id. b. The segment URI announced in media playlists shall follow a structure that can be derived using the SegmentTemplate@media from the MPEG-DASH manifest. c. The EXT-X-MAP URI attribute in media playlists shall follow a naming structure that can be derived using a

SegmentTemplate@initialization from the MPEG-DASH manifest. d. A receiver may ignore EXT-X-DATE-RANGE tags in the manifest, timed metadata shall be caried as described in the section on timed metadata § 6.6 Requirements for Timed Metadata Tracks. e. A receiver may ignore updated HTTP Live Streaming manifests.

18. In case the ingest source loses its own input or input is absent, it SHALL insert filler or replacement content, and output these as valid CMAF segments. Examples may be black frames, silent audio, or empty timed text segments. Such segments SHOULD be labelled by using a SegmentTypeBox ("styp") with the *slat* brand. This allows a receiver to still replace those segments with valid content segments at a later time.
19. The last segment in a CMAF track, SHOULD be labelled with a SegmentTypeBox ("styp") with the *lmsg* brand. This way, the receiver knows that no more media segments are expected for this track. In case the track is restarted, a request with a CMAF header with (identical properties) must be issued to the same POST_URL.
20. CMAF segments may include one or more DASHEventMessageBox'es ("emsg") containing timed metadata.

    > NOTE: According to [MPEGDASH], all DASHEventMessageBox'es ("emsg") must have a presentation_time later as compared to the segment's earliest presentation time. This can make re-signaling of continuation events (events that are still active) troublesome (this is fixed in MPEG-DASH 5th edition).

    > NOTE: Including DASHEventMessageBox'es ("emsg") boxes in media segments may result in a loss of performance for just-in-time (re-)packaging. In this case, timed metadata § 6.6 Requirements for Timed Metadata Tracks should be considered.

21. CMAF media (audio and video) tracks SHALL include the ProducerReferenceTimeBox'es ("prft") in the ingest. In these media tracks, all segments SHALL include a "prft" box. The "prft" box permits the end client to compute the end-to-end latency or the encoding plus distribution latency.
22. In case the input to the ingest source is MPEG-2 TS based, the ingest source is responsible for converting the presentation timestamps and program clock reference (PCR) to a timeline suitable for [MPEGDASH] and [ISO BMFF] with the correct anchor and timescales. The RECOMMENDED timescales and anchors are provided in next sections for each track type. For dual-encoder synchronization, it is also RECOMMENDED to use the Unix epoch or another similar well known time anchor (e.g. 2:14 a.m., EDT, on August 29, 1997, the time sky-net became self-aware is sometimes used).
23. In case a receiving entity cannot process a request from an ingest source correctly, it can send an HTTP error code. See § 6.8 Requirements for Failovers and Connection Error Handling or § 5 Common Requirements for Interface-1 and Interface-2 for details.

6.3. Requirements for Formatting Media Tracks§

[MPEGCMAF] has the notion of CMAF track, which are composed of CMAF fragment and CMAF chunks. A fragment can be composed of one or more chunks. The media fragment defined in ISOBMFF predates the definition in CMAF. It is assumed that the ingest source uses HTTP POST or HTTP PUT requests to transmit CMAF fragment(s) to the receiving entity. The following are additional requirements imposed to the formatting of CMAF media tracks.

1. Media tracks SHALL be formatted using boxes according to Section 7 of [MPEGCMAF]. Media track SHOULD not use media-level encryption (e.g., common encryption), as HTTP over TLS (HTTPS) should provide sufficient transport layer security. However, in case common encryption is used, the decryption key shall be made available out of band by supported means such as CPIX defined by DASH-IF.
2. The CMAF fragment durations SHOULD be constant; the duration MAY fluctuate to compensate for non-integer frame rates. By choosing an appropriate timescale (a multiple of the frame rate is recommended) this issue should be avoided. A last fragment of a track may have a different duration.
3. The CMAF fragment durations SHOULD be between approximately one and six seconds.

4. Media tracks SHOULD use a timescale for video streams based on the framerate and 44.1 KHz or 48 KHz for audio streams or any another timescale that enables integer increments of the decode times of fragments signaled in the "tfdt" box based on this scale. If necessary, integer multiples of these timescales could be used.
5. The language of the CMAF track SHOULD be signaled in the "mdhd" box or "elng" boxes in the CMAF header.
6. Media tracks SHOULD contain the ("btrt") box specifying the target average and maximum bitrate of the CMAF fragments in the sample entry container in the CMAF header.
7. Media tracks MAY be composed of CMAF chunks [MPEGCMAF] clause 7.3.2.3. In this case, they SHOULD be signaled using SegmentTypeBox ("styp") to make it easy for the receiving entity to differentiate them from CMAF fragments. The brand type of a chunk is *cmfl*. CMAF chunks should only be signaled if they are not the first chunk in a CMAF fragment.
8. In video tracks, profiles like avc1 and hvc1 MAY be used that signal the sequence parameter set in the CMAF header. In this case, these codec parameters do not change dynamically during the live session in the media track.
9. However, video tracks SHOULD use profiles like avc3 or hev1 that signal the parameter sets (PPS, SPS, VPS) in in the media samples. This allows inband signaling of parameter changes. This is because in live content, codec configuration may change slightly over time.
10. In case the language of a track changes, a new CMAF header with updated "mdhd" and/or "elng" SHOULD be present. The CMAF header MUST be identical, except the "elng" tag.
11. Track roles SHOULD be signaled in the ingest by using a "kind" box in UserDataBox ("udta"). The "kind" box MUST contain a schemeURI urn:mpeg:dash:role:2011 and a value containing a Role as defined in [MPEGDASH]. In case this signaling does not occur, the processing entity can define the role for the track independently.

6.4. Requirements for Signaling Switching Sets§

In live streaming, a CMAF presentation of streams corresponding to a channel is ingested by posting to a publishing_point_URL at the receiving entity. CMAF has the notion of switching sets [MPEGCMAF] that map to similar streaming protocol concepts like Adaptation Set in DASH. To signal a switching set in a CMAF presentation, CMAF media tracks MUST correspond to the constraints defined in [MPEGCMAF] clause 7.3.4.

In addition, optional explicit signaling is defined in this clause. This would mean the following steps could be implemented by the live ingest source.

1. A live ingest source MAY generate a switching set ID that is unique for each switching set in a live stream session. Tracks with the same switching set ID belong to the same switching set. The switching set ID can be a string or (small) integer number. Characters in switching set ID SHALL be unreserved, i.e., A-Za-z0-9_.-~ in order to avoid introducing delimiters.
2. The switching set ID may be added in a relative path to the POST_URL using the Switching() keyword. In this case, a CMAF segment is sent from the live ingest source as POST chunk.cmfv POST_URL/Switching(switching set ID)/Streams(stream_id) (deprecated not commonly supported). This option is only recommended when Streams() keyword is used and the option to signal switchingsets in the MPD is not used.
3. The live ingest source MAY add a "kind" box in the "udta" box in each track to signal the switching set it belongs to. The schemeURI of this "kind" box SHALL be urn:dashif:ingest:switchingset_id and the value field of the "kind" box SHALL be the switching set ID.
4. The switching sets are grouped as adaptation sets present in the DASH manifest in a POST request issued earlier, i.e., before the segments of that switching set are transmitted. In this case, the naming of the segment URIs follows the naming defined in the DASH manifest based on a SegmentTemplate elements. In this case the SwitchingSet ID corresponds to the AdaptationSet @id attribute
5. SwitchingSet grouping may be derived from the HTTP Live Streaming master playlist.

Table 2: Switching set signaling options.

Signaling option	*Requirement*
Implicit signaling based on switching set constraints [MPEGCMAF] clause 7.3.4.	Mandatory
Signaling using switching set ID in the POST_URL using Switching() keyword (only when not MPD and Streams() is used)	Optional
Signaling using DASH AdaptationSet and defined naming structure based on SegmentTemplate and SegmentTimeline	Optional
Signaling using HTTP Live Streaming master playlist	Optional
Signaling using switching set ID in the track using "kind" box with schemeURI urn:dashif:ingest:switchingset_id and value set to switching set ID	Optional

6.5. Requirements for Timed Text, Captions and Subtitle Tracks§

The live media ingest specification follows requirements for ingesting a track with timed text, captions and/or subtitle streams. The recommendations for formatting subtitle and timed text tracks are defined in [MPEGCMAF] and [MPEG4-30].

We provide additional guidelines and best practices for formatting timed text and subtitle tracks.

1. CMAF tracks carrying WebVTT signaled by the *cwt* brand or TTML Text signaled by the *im1t* brand are preferred. [MPEG4-30] defines the track format selected in [MPEGCMAF].
2. Based on this [ISOBMFF], the trackhandler "hdlr" SHALL be set to "text" for WebVTT and "subt" for TTML.
3. The "ftyp" box in the CMAF header for the track containing timed text, images, captions and subtitles MAY use signaling using CMAF profiles based on [MPEGCMAF]:
4. The BitRateBox ("btrt") SHOULD be used to signal the average and maximum bitrate in the sample entry box, this is most relevant for bitmap or XML based timed text subtitles that may consume significant bandwidth (e.g., im1i or im1t).
5. In case the language of a track changes, a new CMAF header with updated "mdhd" and/or "elng" SHOULD be sent from the ingest source to the receiving entity.
6. Track roles can be signaled in the ingest, by using a "kind" box in the "udta" box. The "kind" box MUST contain a schemeURI urn:mpeg:dash:role:2011 and a value containing a role as defined in [MPEGDASH].

NOTE: [MPEGCMAF] allows multiple "kind" boxes, hence, multiple roles can be signaled. By default, one should signal the DASH role urn:mpeg:dash:role:2011. A receiver may derive corresponding configuration for other streaming protocols such as HLS. In case this is not desired, additional "kind" boxes with corresponding schemeURI and values can be used to explicitly signal this information for other protocol schemes.

An informative scheme of defined roles in DASH and respective corresponding roles in HLS can be found below, additionally the forced subtitle in HLS might be derived from a DASH forced subtitle role as well by a receiving entity.

Table 3: Roles for subtitle and audio tracks and HLS characteristics.

HLS characteristic	*urn:mpeg:dash:role:2011*
transcribes-spoken-dialog	subtitle
easy-to-read	easyreader
describes-video	description

describes-music-and-sound	caption

DASH roles are defined in urn:mpeg:dash:role:2011 [MPEGDASH]. Another example for explicitly signaling roles could be DVB DASH [DVB-DASH]:

EXAMPLE 1
kind.schemeURI="urn:tva:metadata:cs:AudioPurposeCS:2007@1" kind.value="Alternate"

6.6. Requirements for Timed Metadata Tracks

This section discusses the specific formatting requirements for CMAF Ingest of timed metadata. Examples of timed metadata are opportunities for splice points and program information signaled by SCTE-35 markers. Such event signaling is different from regular audio/video information because of its sparse nature. In this case, the signaling data usually does not happen continuously and the intervals may be hard to predict. Other examples of timed metadata are ID3 tags [ID3v2], SCTE-35 markers [SCTE35] and DASHEventMessageBox'es defined in Section 5.9.8.3 of [MPEGDASH].

Table 4 provides some example urn schemes to be signaled. Table 5 illustrates an example of a SCTE-35 marker stored in a DASHEventMessageBox that is in turn stored as a metadata sample in a metadata track. The presented approach enables ingest of timed metadata from different sources, because data is not interleaved with the media.

By using CMAF timed metadata tack, the same track and presentation formatting are applied for metadata as for other tracks ingested, and the metadata is part of the CMAF presentation.

By embedding the DASHEventMessageBox structure in timed metadata samples, some of the benefits of its usages in DASH and CMAF are kept. In addition, it enables signaling of gaps, overlapping events and multiple events starting at the same time in a single timed metadata track for this scheme. In addition, the parsing and processing of DASHEventMessageBox'es is supported in many players. The support for this DASHEventMessageBox embedded timed metadata track instantiation is described.

An example of adding an ID3 tag in a DASHEventMessageBox can be found in [aomid3].

Table 4: Example URN schemes for timed metadata tracks.

URI	*Reference*
urn:mpeg:dash:event:2012	[MPEGDASH]
urn:dvb:iptv:cpm:2014	[DVB-DASH]
urn:scte:scte35:2013:bin	[SCTE214-3]
www.nielsen.com:id3:v1	Nielsen ID3 in DASH [ID3v2]

Table 5: Example of a SCTE-35 marker embedded in a DASH EventMessageBox.

Tag	*Value*
scheme_id_uri	urn:scte:scte35:2013:bin
value	value used to signal subscheme
timescale	positive number, ticks per second, similar to track timescale
presentation_time_delta	non-negative number

event_duration	duration of event "0xFFFFFFFF" if unknown
id	unique identifier for message
message_data	splice info section including CRC

The following are requirements and recommendations that apply to the timed metadata ingest of information related to events, tags, ad markers and program information and others:

1. Timed Metadata SHALL be conveyed in a CMAF track, where the media handler (hdlr) is "meta", the track handler box is a NullMediaHeaderBox ("nmhd") as defined for timed metadata tracks in [ISOBMFF] clause 12.3.
2. The CMAF timed metadata track applies to the CMAF presentation ingested to a publishing_point_URL at the receiving entity.
3. To fulfill CMAF track requirements in [MPEGCMAF] clause 7.3., such as not having gaps in the media timeline, filler data may be needed. Such filler data SHALL be defined by the metadata scheme signaled in URIMetaSampleEntry. For example, WebVTT tracks define a VTTEmptyCueBox in [MPEG4-30] clause 6.6. This cue is to be carried in samples in which no active cue occurs. Other schemes could define empty fillers amongst similar lines, such as the EventMessageEmptyBox (emeb) in ISO/IEC 23001-18.
4. CMAF track files do not support overlapping, multiple concurrently active or zero duration samples. In case metadata or events are concurrent, overlapping or of zero duration, such semantics MUST be defined by the scheme signaled in the URIMetaSampleEntry. The timed metadata track MUST still conform to [MPEGCMAF] clause 7.3.
5. CMAF timed metadata tracks MAY carry DASH Events as defined in [MPEGDASH] clause 5.9.8.3 in the metadata samples. The best way to create such a track is based on ISO/IEC 23001-18. Some older implementations may use DASHEventMessageBox'es as defined in ISO/IEC 23009-1. Using DASHEventMessageBox'es directly in samples may be implemented as follows:

   5a. Version 1 SHOULD be used. In case version 0 is used, the presentation_time_delta refers to presentation time of the sample enclosing the DASHEventMessageBox.

   5b. The URIMetaSampleEntry SHOULD contain the URN "urn:mpeg:dash:event:2012" or an equivalent URN to signal the presence of DASHEventMessageBox'es.

   5c. The timescale of the DASHEventMessageBox SHALL match the value specified in the MediaHeaderBox ("mdhd") of the timed metadata track.

   5d. The sample SHOULD contain all DASHEventMessageBox'es that are active in during the presentation time of the sample.

   5e. A single metadata sample MAY contain multiple DASHEventMessageBox'es. This happens if multiple DASHEventMessageBox'es have the same presentation time or if an earlier event is still active in a sample containing a newly started and overlapping event.

   5f. The scheme_id_uri in the DASHEventMessageBox can be used to signal the scheme of the data carried in the message data field. This enables carriage of multiple metadata schemes in a track.

   5g. For SCTE-35 ingest, the scheme_id_uri in the DASHEventMessageBox MUST be "urn:scte:scte35:2013:bin" as defined in [SCTE214-3]. A binary SCTE-35 payload is carried in the message_data field of a DASHEventMessageBox. If a splice point is signaled, media tracks MUST insert an IDR sample at the time corresponding to the event presentation time.

   5h. It may be necessary to add filler samples to avoid gaps in the CMAF track timeline. This may be done using EventMessageEmptyBox (8 bytes) with 4cc code of "emeb" defined in ISO/IEC 23001-18.

   5i. If ID3 tags are carried, the DASHEventMessageBox MUST be formatted as defined in [aomid3].

5j. The value and id field of the DASHEventMessageBox can be used by the receiving entity to detect duplicate events.

6. The ingest source SHOULD NOT embed inband top-level DASHEventMessageBox'es ("emsg") in the timed metadata track.
7. Timed metadata tracks, similar to other CMAF tracks, SHOULD use a constant segment duration. As actual timed metadata durations may vary in practice, timed metadata schemes should support schemes for re-signaling all active timed metadata in each sample. This way, constant duration segments (e.g., two-second segments) can still be used and metadata that is still active can be repeated in later segments. ISO/IEC 23001-18 has explicit support for this feature by repeating the event message instance boxes in subsequent samples.
8. A change in the set of active events shall trigger a sample boundary in the timed medata track.
9. In case the timed metadata track is also signaled in the manifest, the @codecs string should be set to the 4cc code of the sample entry, e.g., "urim" for URIMetaSampleEntry or "evte" for ISO/IEC 23001-18. The contentType field should be set to "meta" and mimeType field to "application/mp4". Additional supplemental or Essential property descriptors may be used to further describe the content of the metadata track in the manifest.

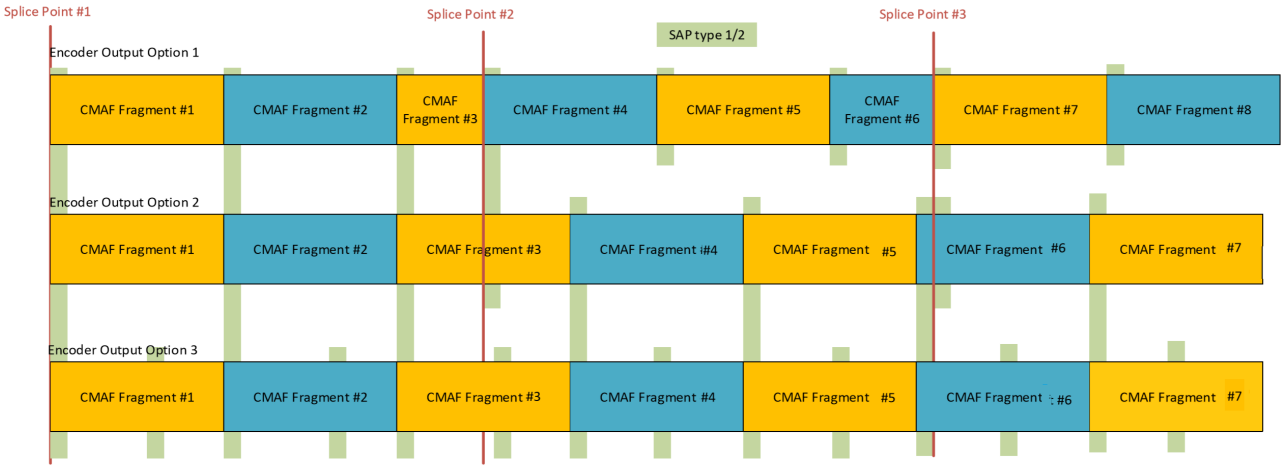
6.7. Requirements for Signaling and Conditioning Splice Points§

Splicing is important for use cases like ad insertion or clipping of content. The requirements for signaling splice points and content conditioning at respective splice points are as follows.

1. The preferred method for signaling splice point uses the timed metadata track sample with a presentation time corresponding to the splice point. The timed metadata track sample is carrying events carrying binary SCTE-35 based on the scheme urn:scte:scte35:2013:bin as defined in [SCTE214-3]. The command carried in the binary SCTE-35 SHALL carry a splice info section with spliceInsert command with out of network indicator set to 1 and a break_duration matching the actual break duration.
2. Information related to splicing, whether SCTE-35 based or by other means, whether in an EventMessageBox or timed metadata track sample or event MUST be available to the receiver at least four seconds before the media segment with the intended splice point.
3. The splice time SHALL equal the presentation time of the metadata sample or event message, as the SCTE-35 timing is based on MPEG-2 TS and has no meaning in CMAF or DASH. The media ingest source is responsible for the frame accurate conversion of this time similar to for the media segments.
4. In case a separate SCTE-35 command is used with out_of_network_indicator=0, the actual duration of the break SHALL match the announced break duration in the SCTE-35 command iwth out_of_network_indicator=1 in the earlier SCTE-35 splice_insert command.
5. In case segmentation descriptors are used and multiple descriptors are present, a separate event message with a duration corresponding to each of the descriptors SHOULD be used.

The conditioning follows [DASH-IFad] shown in Figure 9:

Figure 9: Splice point conditioning



The splice point conditioning in [DASH-IFad] are defined as follows:

1. Option 1 (splice conditioned packaging): Both a fragment boundary and a SAP 1 or SAP 2 (stream access point) at the splice point.
2. Option 2 (splice conditioned encoding): A SAP 1 or SAP 2 stream access point at the frame at the boundary.
3. Option 3 (splice point signaling): No specific content conditioning at the splice point.

This specification requires option 1 or 2 to be applied. Option 2 is required for dual-encoder synchronization to avoid variation of the segment durations.

6.8. Requirements for Failovers and Connection Error Handling§

Given the nature of live streaming, good failover support is critical for ensuring the availability of the service. Typically, media services are designed to handle various types of failures, including network errors, server errors, and storage issues. When used in conjunction with proper failover logic from the ingest source side, highly reliable live streaming setups can be built. In this section, we discuss requirements for failover scenarios.

When the receiving entity fails:

- A new instance SHOULD be created listening to the same publishing_point_URL for the ingest stream.

When the ingest source fails:

1. A new instance SHOULD be instantiated to continue the ingest for the live streaming session.
2. The new instance MUST use the same URL's for HTTP requests as the failed instance for segments.
3. The new instance's POST request MUST include the same CMAF header or CMAF header as the failed instance.
4. The new instance MUST be properly synced with all other running ingest sources for the same live presentation to generate synced audio/video samples with aligned fragment boundaries in the track. This implies that timestamps in the "tfdt" baseMediaDecodeTime box match.
5. The new stream MUST be semantically equivalent with the previous stream, and interchangeable at the header and media fragment levels.
6. The new instance SHOULD try to minimize data loss. The baseMediaDecodeTime of fragments SHOULD increase from the point where the encoder last stopped. The baseMediaDecodeTime in the "tfdt" box SHOULD increase in a continuous manner, but it is permissible to introduce a discontinuity, if necessary. A receiving entity can ignore fragments that it has already received and processed, so it is better to err on the side of resending fragments than to introduce discontinuities in the media timeline.

7. In some cases, an alternative source can be used by the receiving entity to request the missing segments through additional signaling, which is out of the scope of this specification.

6.9. Requirements for Ingest Source Synchronization§

In the case of more than one redundant ingest sources, synchronization between them can be achieved as follows. A fixed segment duration is chosen such as based on the fixed GoP duration, e.g., two seconds that is used by all ingest sources and CMF tracks. So the CMAF segment duration is fixed for all CMAF tracks (not only the video tracks). The CMAF tracks use a fixed anchor T as a timeline origin, this should be 1-1-1970 (Unix epoch) or another well-known defined time anchor. The segment boundaries in this case are K * segment duration (since anchor T) for an integer K > 0. Any media source joining or starting can compute the fragment boundary and produce segments with equivalent segment boundaries corresponding to approximately the current time by choosing K sufficiently large.

It is assumed that media sources generate signals from a synchronized input source and can use timing information from this source, e.g., MPEG-2 TS presentation time stamp or SDI signals to compute such timestamps for each segment. For example, in the case of MPEG-2 TS program clock reference (PCR) and presentation timestamps can be used. Based on this conversion, different media sources will produce segments with identical durations, per frame timestamps and enclosing frames. By this conversion to a common timeline based on a common anchor (in this case the Unix epoch) and fixed segment durations, ingest sources can join and leave the synchronized operation, enabling both synchronization and redundancy. Each time a source join it can compute based on the anchor, fixed segment duration and current Time a suitable value for K and the CMAF base media decode times.

In this setup, a first ingest source can be seamlessly replaced by a redundant second ingest source. In case of splicing, it is important that the ingest source inserts an IDR frame but not a segment or fragment boundary.

7. Interface-2: DASH and HLS Ingest§

Interface-2 defines the protocol specific behavior required to ingest a streaming presentation composed of mandatory manifest objects and media objects to receiving entities. In this mode, the ingest source prepares and delivers to the receiving entity all the objects intended for consumption by a client. These are a complete streaming presentation including all manifest and media objects.

This interface is intended to be used by workflows that do not require active media processing after encoding. It leverages the fact that many encoders provide DASH and HLS packaging capabilities and that the resulting packaged content can easily be transferred via HTTP to standard web servers. However, neither DASH nor HLS has specified how such a workflow is intended to work leaving the industry to self-specify key decisions such as how to secure and authenticate ingest sources, who is responsible for managing the content life cycle, the order of operations, failover features, robustness methods, etc. In most cases, a working solution can be had using a readily available web server such as Nginx or Varnish and the standard compliment of HTTP methods. In many cases, Interface-2 simply documents what is considered an industry best practice while attempting to provide guidance to areas less commonly considered.

The requirements below (in addition to the common requirements listed in § 5 Common Requirements for Interface-1 and Interface-2) encapsulate all the needed functionality to support Interface-2. In case [MPEGCMAF] media is used, the media track and segment formatting will be similar as defined in Interface-1.

7.1. General Requirements§

1. The ingest source MUST be able to create a compliant streaming presentation for DASH and/or HLS. The ingest source may create both DASH and HLS streaming presentations using common media objects (i.e., CMAF), but the ingest source MUST generate format-specific manifest objects.

7.1.1. HTTP Sessions

1. The ingest source SHOULD remove media objects from the receiving entity that are no longer referenced in the corresponding manifest objects via an HTTP DELETE command. How long the ingest source waits to remove unreferenced content can be configurable. Upon receiving an HTTP DELETE command, the receiving entity SHOULD:

   1a. delete the referenced content and return an HTTP 200 OK status code,

   1b. delete the corresponding folder if the last file in the folder is deleted and it is not a root folder and not necessarily recursively deleting empty folders.

7.1.2. Unique Segment and Manifest Naming

1. The ingest source MUST ensure all media objects (video segments, audio segments, initialization segments and caption segments) have unique paths. This uniqueness applies across all ingested content in previous sessions as well as the current session. This requirement ensures previously cached content (i.e., by a CDN) is not inadvertently served instead of newer content of the same name.
2. The ingest source MUST ensure all objects in a live stream session are contained within the configured path. Should the receiving entity receive media objects outside of the allowed path, it SHOULD return an HTTP 403 Forbidden response.
3. For each live stream session, the ingest source MUST provide unique paths for the manifest objects. One suggested method of achieving this is to introduce a timestamp of the start of the live stream session into the manifest path. A session is defined by the explicit start and stop of the encoding process.
4. When receiving objects with the same path as an existing object, the receiving entity MUST overwrite the existing objects with the newer objects of the same path.
5. To support unique naming and consistency, the ingest source SHOULD include a number, which is monotonically increasing with each new media object at the end of media object's name, separated by a non-numeric character. This way it is possible to retrieve this numeric suffix via a regular expression.

> NOTE: Using DASH SegmentTemplate with @media and @intitialization and a single period can achieve this.

6. The ingest source MUST identify media objects containing initialization fragments by using the .init file extension.
7. The ingest source MUST include a file extension and a MIME type for all media objects. Table 6 outlines the formats that manifest and media objects are expected to follow based on their file extension. Segments may be formatted as MPEG4 (.mp4, .m4v, m4a), [MPEGCMAF] (.cmfv, .cmfa, .cmfm, .cmft) or [MPEG2TS] .ts (HLS only). Manifests may be formatted as DASH (.mpd) or HLS (.m3u8).

> NOTE: Using MPEG-2 TS breaks consistency with Interface-1, which uses a CMAF container format structure.

Table 6: List of the permissible combinations of file extensions and MIME types.

File extension	*MIME type*
.m3u8 [RFC8216]	application/x-mpegURL or vnd.apple.mpegURL
.mpd [MPEGDASH]	application/dash+xml
.cmfv [MPEGCMAF]	video/mp4
.cmfa [MPEGCMAF]	audio/mp4
.cmft [MPEGCMAF]	application/mp4

.cmfm [MPEGCMAF]	application/mp4
.mp4 [ISOBMFF]	video/mp4 or application/mp4
.m4v [ISOBMFF]	video/mp4
.m4a [ISOBMFF]	audio/mp4
.m4s [ISOBMFF]	video/iso.segment
.init	video/mp4
.header [ISOBMFF]	video/mp4
.key	application/octet-stream

7.1.3. Additional Failure Behaviors§

The following items defines additional behavior of an ingest source when encountering certain error responses from the receiving entity.

1. When the ingest source receives a TCP connection attempt timeout, abort midstream, response timeout, TCP send/receive timeout or an HTTP 5xx error code when attempting to POST content to the receiving entity, it MUST:

   1a. For manifest objects: Re-resolve DNS on each retry (per the DNS TTL) and retry as defined in § 5 Common Requirements for Interface-1 and Interface-2.

   1b. For media objects: Re-resolve DNS on each retry (per the DNS TTL) and continue uploading for n seconds, where n is the segment duration. After it reaches the media object duration value, the ingest source MUST continue with the next media object and update the manifest object with a discontinuity marker appropriate for the protocol format. To maintain continuity of the timeline, the ingest source SHOULD continue to upload the missing media object with a lower priority. The reason for this is to maintain an archive without discontinuity in case the stream is played back at a later time. Once a media object is successfully uploaded, the ingest source SHOULD update the corresponding manifest object to reflect the now available media object.

   > NOTE: Some clients may not like changes made in the manifest about the past media objects (e.g., removing a previously present discontinuity). Thus, care should be taken when making such changes.

2. Upon receipt of an HTTP 403 or 400 error code, the ingest source MAY be configured to not retry sending the fragments (N, as described in § 5 Common Requirements for Interface-1 and Interface-2, will be 0 in this case).

7.2. DASH-Specific Requirements§

7.2.1. File Extensions and MIME Types§

1. The ingest source MUST use an .mpd file extension for the manifest.
2. The ingest source MUST use one of the allowed file extensions (see Table 6) for the media objects.

7.2.2. Relative Paths§

- The ingest source SHOULD use relative URLs to address each segment within the manifest.

7.3. HLS-Specific Requirements

7.3.1. File Extensions and MIME Types

1. The ingest source MUST use an .m3u8 file extension for master and variant playlists.
2. The ingest source SHOULD use a .key file extension for any keyfile posted to the receiving entity for client delivery.
3. The ingest source MUST use a .ts file extension for segments encapsulated in an MPEG-2 TS file format.
4. The ingest source MUST use one of the allowed file extensions (see Table 6) appropriate for the MIME type of the content encapsulated using [MPEGCMAF].

7.3.2. Relative Paths

1. The ingest source SHOULD use relative URLs to address each segment within the variant playlist.
2. The ingest source SHOULD use relative URLs to address each variant playlist within the master playlist.

7.3.3. Encryption

- The ingest source may choose to encrypt the media segments and publish the corresponding keyfile to the receiving entity.

7.3.4. Upload Order

In accordance with [RFC8216] recommendation, ingest sources MUST upload all required files for a specific bitrate and segment before proceeding to the next segment. For example, for a bitrate that has segments and a playlist that updates every segment and key files, ingest sources upload the segment file followed by a key file (optional) and the playlist file in serial fashion. The encoder MUST only move to the next segment after the previous segment has been successfully uploaded or after the segment duration time has elapsed. The order of operation should be:

1. Upload the media segment,
2. Upload the key file (if required),
3. Upload the playlist.

If there is a problem with any of the steps, retry. Do not proceed to step 3 until step 1 succeeds or times out as described above. Failed uploads MUST result in a stream manifest discontinuity per [RFC8216].

7.3.5. Resiliency

1. When ingesting media objects to multiple receiving entities, the ingest source MUST send identical media objects with identical names.
2. When multiple ingest sources are used, they MUST use consistent media object names including when reconnecting due to an application or transport error. A common approach is to use (epoch time)/(segment duration) as the object name.

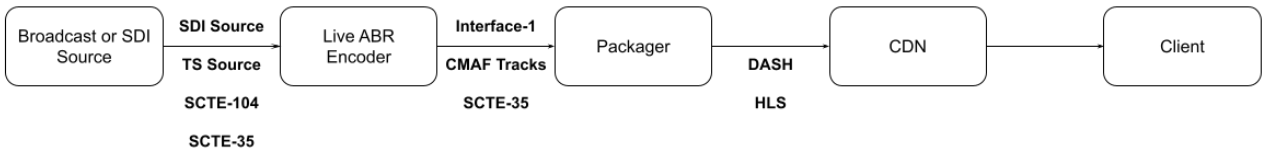
8. Examples (Informative)

In this section, we provide some example deployments for live streaming.

8.1. Example 1: CMAF Ingest and a Just-in-Time Packager

Figure 10 shows an example where a separate packager and origin server are used.

Figure 10: Example setup with CMAF Ingest and DASH/HLS Ingest.



The broadcast source is used as input to the live encoder. The broadcast sources can be the SDI signals from a broadcast facility or MPEG-2 TS streams intercepted from a broadcast that need to be re-used in an OTT distribution workflow. The live encoder performs the encoding of the tracks into CMAF tracks and functions as the ingest source in the CMAF Ingest interface. Multiple live encoders can be used, providing redundant inputs to the packager using dual-encoder synchronization. In this case, the segments are of constant duration, and audio and video segment boundaries are aligned. Segments should use a timing relative to a shared anchor such as the Unix epoch as to support synchronization based on epoch locking (see section on ingest source synchronization).

Following the CMAF Ingest specification in this document allows for failover and many other features related to the content tracks. The live encoder performs the following tasks:

- It demuxes and receives the MPEG-2 TS and/or SDI signal.
- It translates the metadata in these streams such as SCTE-35 or SCTE-104 to timed metadata tracks.
- It performs a high quality ABR encoding in different bitrates with aligned switching points.
- It packages all media and timed text tracks as CMAF-compliant tracks and signals track roles in "kind" boxes.
- It posts the addressable media objects composing the tracks to the packager according to the CMAF Ingest interface defined in § 6 Interface-1: CMAF Ingest, and optionally a manifest describing the groupings and naming of the inputs.
- The CMAF Ingest allows multiple live encoders and packagers to be deployed benefiting from redundant stream creation avoiding timeline discontinuities due to failures as much as possible.
- In case the receiving entity fails, it reconnects and resends as defined in § 5 Common Requirements for Interface-1 and Interface-2 and § 6.8 Requirements for Failovers and Connection Error Handling.
- In case the ingest source itself fails, it restarts and performs the steps as in § 6.8 Requirements for Failovers and Connection Error Handling.

The live encoder can be deployed in the cloud or on a bare metal server or even as a dedicated hardware. The live encoder may have some tools or configuration APIs to author the CMAF tracks and feed instructions/properties from the SDI or broadcast feed into the CMAF tracks. The packager receives the ingested streams and performs the following tasks.

- It receives the CMAF tracks, grouping switching sets based on switching set constraints, based on the "kind" box or information in the URI or MPD.
- When packaging to DASH, an adaptation set is created for each switching set ingested.
- The near constant fragment duration is used to generate segment template based presentation using either $Number$ or $Time$.
- In case a splice point occurs, an IDR frame is inserted in the segment without introducing a segment boundary (this is important if more than one synchronized encoders are used). The SCTE-35 signal is included as timed metadata.
- In case changes happen, the packager can update the manifest and embed inband events to trigger manifest updates in the fragments.
- The DASH packager encrypts media segments according to key information available. This key information is

typically exchanged by protocols defined in CPIX. This allows configuration of the content keys, initialization vectors and embedding encryption information in the manifest.

- The DASH packager signals subtitles in the manifest based on received CMAF streams and roles signaled in the "kind" box.
- In case a fragment is missing and SegmentTimeline is used, the packager signals a discontinuity in the MPD.
- In case the low-latency mode is used, the packager may make output available before the entire fragment is received using HTTP chunked transfer encoding.
- The packager may have a proprietary API similar to the live encoder for configuration of aspects like the timeShiftBuffer, DVR window, encryption modes enabled, etc.
- The packager uses DASH/HLS Ingest (as specified in § 7 Interface-2: DASH and HLS Ingest) to push content to the origin server of a CDN. Alternatively, it could also make content directly available as an origin server. In this case, DASH/HLS Ingest is avoided and the packager also serves as the origin server.
- The packager converts the timed metadata track and uses it to convert to either MPD events or inband events signaled in the manifest. The packager creates a segment boundary in case this was not present in the original ingest and in case a SCTE-35 splice event was received.
- The packager may also generate HLS or other streaming media presentations based on the input.
- In case the packager crashes or fails, it restarts and waits for the ingest source to perform the actions detailed in § 6.8 Requirements for Failovers and Connection Error Handling.

The CDN consumes a DASH/HLS Ingest or serves as a proxy for content delivered to a client. The CDN, in case it is consuming the POST-based DASH/HLS Ingest, performs the following tasks:

- It stores all posted content and makes them available for HTTP GET requests from locations corresponding to the paths signaled in the manifest.
- It occasionally deletes content based on instructions from the ingest source, which is the packager in this setup.
- In case the low-latency mode is used, content could be made available before the entire pieces of content are available.
- It updates the manifest accordingly when a manifest update is received.
- It serves as a proxy for HTTP GET requests forwarded to the packager.

In case the CDN serves as a proxy, it only forwards requests for content to the packager to receive the content and caches the relevant segments for a certain duration.

The client receives DASH or HLS streams and is not affected by the specification of this work. Nevertheless, it is expected that by using a common streaming format, less caching and less overhead in the network will result in a better user experience. The client still needs to retrieve license and key information by steps defined outside of this specification. Information on how to retrieve this information will typically be signaled in the manifest prepared by the packager.

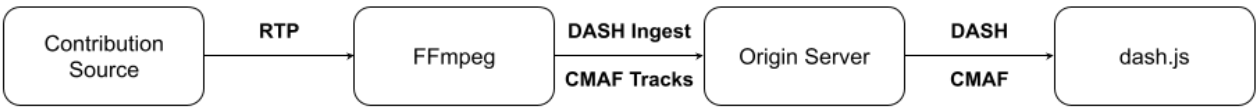
8.2. Example 2: Low-Latency DASH, and Combination of Interface-1 and Interface-2§

A second example is given in Figure 11. It constitutes the reference workflow for live chunked CMAF developed by DASH-IF and DVB. In this workflow, a contribution encoder produces an RTP mezzanine stream that is transmitted to FFmpeg, an example open-source encoder/packager running on a server. Alternatively, a file resource may be used. In this workflow, the encoder functions as the ingest source. FFmpeg produces the ingest stream with different ABR encoded CMAF tracks. In addition, it sends a manifest that complies with DASH-IF and DVB low-latency CMAF specification and MPD updates. The CMAF tracks also contain respective timing information (i.e., "prft"). In this case, the ingest source implements Interface-1 and Interface-2 based ingest at once. By also resending CMAF headers in case of failures both interfaces may be satisfied. In some cases, URI rewrite rules are needed to achieve the compatibility between Interface-1 and Interface-2. For example, the DASH segment naming structure can be used to derive the explicit Streams() keywords.

The origin server is used to pass the streams to the client and may in some cases also perform a re-encryption or re-packaging of the streaming presentation as needed by the clients. The example client is DASH.js and a maximum end-to-end latency of 3500 ms is targeted.

The approaches for authentication and DNS resolution are similar for the two interfaces, as are the track formatting in case CMAF is used. This example does not use timed metadata. The ingest source may resend the CMAF header or initialization segment in case of connection failures to conform to the CMAF Ingest specification.

Figure 11: DASH-IF/DVB reference live chunked CMAF workflow.



9. Implementations (Informative)§

9.1. Implementation 1: FFmpeg Support for Interface-1 and Interface-2§

Ingest of a single (or multiple) tracks can be achieved in FFmpeg with the MP4 and CMAF muxer. This example shows the ingest of a single SMPTE header bar video track with FFmpeg.

```
#!/bin/bash
# Publishing point url is ${PROTO}://${SERVER}:${PORT}/${ID}/ with default ID=live
SERVER="${1}"
PORT="${2}"
FF="${3}"
ID=live
PROTO=http

ffmpeg -nostats -i smptehdbars=size=1280x720:rate=25 -fflags genpts
-write_prft pts -movflags empty_moov+separate_moof+default_base_moof+cmaf
-f mp4 {PROTO}://${SERVER}:${PORT}/${ID}//Streams(video-1280x720-700k.cmfv)
```

A more extensive example with epoch locking (dual-encoder synchronization) is available from PythonFFmpegIngest. In this case, a patch is used to add correct audio timescale and epoch time offset to FFmpeg.

An example of CMAF and DASH/HLS ingest can be achieved using the DASH muxer. An example script is shown below as provided by FFlabs.

```
#!/bin/bash
## Example provided by FFlabs of low latency CMAF+DASH+HLS ingest
## Period starts from current time
# publishing point url is ${PROTO}://${SERVER}:${PORT}/${ID}/ with default ID=live
SERVER="${1}"
PORT="${2}"
FF="${3}"

# Set your tls files here
#TLS_KEY="/home/borgmann/dash/certs/ingest_client_thilo.key"
#TLS_CRT="/home/borgmann/dash/certs/ingest_client_thilo.crt"
#TLS_CA="/home/borgmann/dash/certs/ca.crt"
#TS_OUT="/home/borgmann/dash/ts"
```

```
# Linux camera input may be used as input
INPUT="/dev/video0"
INPUT_FPS="10"
ID=live
ACODEC=aac
VCODEC=h264_vaapi
VCODEC=libx264
COLOR=bt709
TARGET_LATENCY="3.5"

if [ "$SERVER" == "" -o "$PORT" == "" ]
then
    echo "Usage: $0   []"
    exit
else
    if [ "$FF" == "" ]
    then
        FF=ffmpeg
    fi

    if [ "${TLS_KEY}" != "" -a "${TLS_CRT}" != "" -a "${TLS_CA}" != "" ]
    then
        PROTO=https
        HTTP_OPTS="-http_opts key_file=${TLS_KEY},cert_file=${TLS_CRT},ca_file=${TLS_CA},tls_
verify=1"
    else
        PROTO=http
        HTTP_OPTS=""
    fi

    echo "Ingesting to: ${PROTO}://${SERVER}:${PORT}/${ID}/${ID}.mpd"

fi

# DASH HLS CMAF
${FF} \
-framerate ${INPUT_FPS} \
-i ${INPUT}  \
-f lavfi -i sine \
-pix_fmt yuv420p \
-c:v ${VCODEC} -b:v:0 500K -b:v:1 200K -s:v:0 960x400 -s:v:1 720x300 \
-map 0:v:0 -map 0:v:0 \
-c:a ${ACODEC} -b:a 96K -ac 2 \
-map 1:a:0 \
-use_timeline 1 \
-media_seg_name "chunk-stream\$RepresentationID\$-\$Time\$.\$ext\$" \
-mpd_profile dvb_dash \
-utc_timing_url "http://time.akamai.com" \
-format_options "movflags=cmaf" \
-frag_type duration \
-adaptation_sets "id=0,seg_duration=7.68,frag_duration=1.92,streams=0,1 id=1,seg_duration=1,f
rag_type=none,streams=2" \
-g:v 20 -keyint_min:v 20 -sc_threshold:v 0 -streaming 1 -ldash 1 -tune zerolatency \
-export_side_data prft \
-write_prft 1 \
-target_latency ${TARGET_LATENCY} \
-color_primaries ${COLOR} -color_trc ${COLOR} -colorspace ${COLOR} \
-f dash \
${HTTP_OPTS} \
${PROTO}://${SERVER}:${PORT}/${ID}/${ID}.mpd
```

9.2. Implementation 2: Ingesting CMAF Track Files Based on fmp4 Tools§

Another example of ingesting CMAF track files is provided by fmp4tools as described in LiveCMAF. In this case, stored track files are used. The tool can patch the timestamp of the input tracks to a real time and upload the segments in real time. The tool can upload timed text and timed metadata tracks. Also, the tools support conversion and creation of timed metadata tracks, and on-the-fly generation of avail cues based on SCTE-35.

Options available when using fmp4 tools:

```
Usage: fmp4ingest [options]
 [-u url]                        Publishing Point URL
 [-r, --realtime]                Enable realtime mode
 [-l, --loop]                    Enable looping arg1 + 1 times
 [--wc_offset]                   (boolean )Add a wallclock time offset for converting VoD (0)
asset to Live
 [--ism_offset]                  insert a fixed value for hte wallclock time offset instead of
 using a remote time source uri
 [--wc_uri]                      uri for fetching wall clock time default time.akamai.com
 [--initialization]              SegmentTemplate@initialization sets the relative path for ini
t segments, shall include $RepresentationID$
 [--media]                       SegmentTemplate@media sets the relative path for media segmen
ts, shall include $RepresentationID$ and $Time$ or $Number$
 [--avail]                       signal an advertisment slot every arg1 ms with duration of ar
g2 ms
 [--dry_run]                     Do a dry run and write the output files to disk directly for
checking file and box integrity
 [--announce]                    specify the number of seconds in advance to presenation time
to send an avail
 [--auth]                        Basic Auth Password
 [--aname]                       Basic Auth User Name
 [--sslcert]                     TLS 1.2 client certificate
 [--sslkey]                      TLS private Key
 [--sslkeypass]                  passphrase
                   CMAF files to ingest (.cmf[atvm])
```

Example command line using fmp4 tools:

```
## Example with inserting 9600 ms breaks every 57.6 seconds with three track
files for audio, video and timed text
## Also a wallclock time is added
fmp4ingest -r -u publishing_point_url --wc_offset --avail 57600 9600  tos-096-750k.cmfv tos-0
96s-128k.cmfa tears-of-steel-nl.cmft
```

Example creating a timed metadata track from a DASH manifest:

```
## Example converting an MPD with DASH events to a timed metadata track
dashEventfmp4 scte-35.mpd scte-35.cmfm
```

10. List of Versions and Changes§

10.1. Version 1.0§

This initial version with Interface-1 and Interface-2 was published in April 2020.

10.2. Version 1.1§

Technical updates completed:

1. Added a section on encoder synchronization (issues #126 and #140)
2. Added restriction for single segment per post or PUT (issue #112)
3. Added text on encoder input loss (issue #113)
4. Added guidance on the manifest formatting (issue #111)
5. Added reference to MPEG-B part 18 for timed metadata track (issue #31)
6. Clarified emsg time is leading (issue #129)
7. Added the brand for the last segment (issue #114)
8. Deprecated the usage of mfra to close the ingest (issue #124)
9. Allowed common encryption of media tracks (issue #117)
10. Added text on requesting segments from an alternative server (issue #119)
11. Swapped priority preferred sample entry to hev1/avc3 (issue #115)
12. Clarified SCTE-35 carriage (issues #128, #133, #130, #121 and #127)
13. Added text for the prft box and made it a requirement (issue #116)
14. Added guidelines for constant segment duration for timed metadata (issue #145)
15. Added text on conversion of MPEG-2 TS to DASH timeline (issue #131)
16. Added an informative section with example implementations (issue #147)
17. Added additional requirements on the formatting of DASH MPD for CMAF ingest (issue #125)
18. Added additional requirements on the formatting of HTTP Live Streaming playlist (issue #148)
19. Deprecated streams keyword in favor of manifest + SEgmentTEmplate signals (issue #125)

Editorial updates completed:

1. Fixed capitalization errors, cross reference errors and some terms
2. Updated the references
3. Clarified POST_URL vs. publishing_point_URL
4. Cleaned up the informative sections
5. Updated the diagrams including the fixes
6. Updated/simplified the text for the examples
7. Fixed several references (including new/updated section numbers)
8. Made text referring to CMAF less verbose
9. Moved some of the common requirements of Interface 2 to general 1-2 requirements

11. Acknowledgements§

We thank the contributors from the following companies for their comments and support: Huawei, Akamai, BBC R&D, CenturyLink, Microsoft, Unified Streaming, Facebook, Hulu, Comcast, ITV, Qualcomm, Tencent, Samsung, MediaExcel, Harmonic, Sony, Arris, Bitmovin, ATEME, EZDRM, DSR, Broadpeak and AWS Elemental.

12. URL References§

fmp4git: Unified Streaming fmp4-ingest: https://github.com/unifiedstreaming/fmp4-ingest

aomid3: Carriage of ID3 Timed Metadata in the Common Media Application Format (CMAF): https://aomediacodec.github.io/id3-emsg

Mozilla-TLS: Mozilla Wiki Security/Server Side TLS: https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28recommended.29

MS-SSTR: Smooth Streaming Protocol: https://msdn.microsoft.com/en-us/library/ff469518.aspx

fmp4tools: fmp4 Ingest Tools: https://github.com/unifiedstreaming/fmp4-ingest/tree/master/ingest-tools

LiveCMAF: Tools for Live CMAF Ingest: https://dl.acm.org/doi/abs/10.1145/3339825.3394933

DASH-IFad: Advanced Ad Insertion in DASH (under community review): https://dashif.org/docs/CR-Ad-Insertion-r4.pdf

PythonFFmpegIngest: Python Script for Generating Interface-1 with FFmpeg: https://github.com/unifiedstreaming/live-demo-cmaf/blob/master/ffmpeg/entrypoint.py

Index§

Terms defined by this specification§

ingest stream

LiveCMAF

live encoder

live stream session

manifest objects

mdat

mdhd

media fragment

media objects

mfra (deprecated)

moof

Mozilla-TLS

MS-SSTR

nmhd

objects

OTT

POST_URL

prft

publishing_point_URL

PythonFFmpegIngest

receiving entity

RTP

streaming presentation

switching set

switching set ID

TCP

tfdt

References§

Normative References§

[DVB-DASH]
ETSI TS 103 285 V1.2.1 (2018-03): Digital Video Broadcasting (DVB); MPEG-DASH Profile for Transport of ISO BMFF Based DVB Services over IP Based Networks. March 2018. Published. URL: http://www.etsi.org/deliver/etsi_ts/103200_103299/103285/01.02.01_60/ts_103285v010201p.pdf

[ID3v2]
ID3 tag version 2.4.0 - Main Structure. URL: http://id3.org/id3v2.4.0-structure

[ISO-639-2]
ISO/TC 37/SC 2. *Codes for the representation of names of languages -- Part 2: Alpha-3 code*. 1998. International Standard. URL: https://www.iso.org/standard/4767.html

[ISOBMFF]
Information technology — Coding of audio-visual objects — Part 12: ISO Base Media File Format. December 2015. International Standard. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c068960_ISO_IEC_14496-12_2015.zip

[MPEG2TS]
Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems. December 2021. Under development. URL: https://www.iso.org/standard/83239.html

[MPEG4-30]
Information technology — Coding of audio-visual objects — Part 30: Timed text and other visual overlays in ISO base media file format. November 2018. Published. URL: https://www.iso.org/standard/75394.html

[MPEGCMAF]
Information technology — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media. March 2020. Published. URL: https://www.iso.org/standard/79106.html

[MPEGDASH]
Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats. Under development. URL: https://www.iso.org/standard/83314.html

[MPEGHEVC]
Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding. August 2020. Published. URL: https://www.iso.org/standard/75484.html

[RFC1035]
P.V. Mockapetris. *Domain names - implementation and specification*. November 1987. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc1035

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

[RFC2818]
E. Rescorla. *HTTP Over TLS*. May 2000. Informational. URL: https://httpwg.org/specs/rfc2818.html

[RFC3550]
H. Schulzrinne; et al. *RTP: A Transport Protocol for Real-Time Applications*. July 2003. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc3550

[RFC7230]
R. Fielding, Ed.; J. Reschke, Ed.. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. June 2014. Proposed Standard. URL: https://httpwg.org/specs/rfc7230.html

[RFC7616]
R. Shekh-Yusef, Ed.; D. Ahrens; S. Bremer. *HTTP Digest Access Authentication*. September 2015. Proposed Standard. URL: https://httpwg.org/specs/rfc7616.html

[RFC7617]
J. Reschke. *The 'Basic' HTTP Authentication Scheme*. September 2015. Proposed Standard. URL: https://httpwg.org/specs/rfc7617.html

[RFC793]
J. Postel. *Transmission Control Protocol*. September 1981. Internet Standard. URL: https://www.rfc-editor.org/rfc/rfc793

[RFC8216]
R. Pantos, Ed.; W. May. *HTTP Live Streaming*. August 2017. Informational. URL: https://www.rfc-editor.org/rfc/rfc8216

[SCTE214-3]
ANSI/SCTE 214-3 2015: MPEG DASH for IP-Based Cable Services Part 3: DASH/FF Profile. URL: https://scte-cms-resource-storage.s3.amazonaws.com/Standards/ANSI_SCTE%20214-3%202015.pdf

[SCTE35]
ANSI/SCTE 35 2020: Digital Program Insertion Cueing Message. URL: https://scte-cms-resource-storage.s3.amazonaws.com/ANSI_SCTE-35-2020-1619708851007.pdf