Media Over QUIC
Internet-Draft
Intended status: Informational
Expires: 4 December 2026

T. Einarsson
Eyevinn Technology
H. Bjrs
KTH

2 June 2026

Low Overhead CMAF for Media over QUIC (LOCMAF)
draft-einarsson-moq-locmaf-00

Abstract

This document specifies LOCMAF (Low Overhead CMAF for Media over QUIC), a compact wire format for streaming low-latency CMAF media over the MoQ Transport protocol (MOQT) with per-object overhead comparable to the Low Overhead Container (LOC). LOCMAF carries the CMAF chunk head metadata from a single moof (movie fragment) — as a small set of tagged fields inside one of two LOCMAF object kinds, while leaving the sample data (mdat) untouched. In addition, it can carry the optional styp (segment type), prft (producer reference time), any number of emsg (event message) boxes. The first object of each MOQT group carries a full reference; subsequent objects in the same group carry only the differences. The receiver reconstructs CMAF chunks that are semantically equivalent to the sender input, including encryption metadata required by CMAF DRM (Common Encryption) pipelines.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://Eyevinn.github.io/locmaf-id/draft-einarsson-moq-locmaf.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-einarsson-moq-locmaf/.

Discussion of this document takes place on the Media Over QUIC Working Group mailing list (mailto:moq@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/moq/. Subscribe at https://www.ietf.org/mailman/listinfo/moq/.

Source for this draft and an issue tracker can be found at https://github.com/Eyevinn/locmaf-id.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This Internet-Draft will expire on 4 December 2026.

Copyright Notice

Table of Contents

1. Introduction

CMAF [CMAF] chunk headers have a size starting at 100 bytes, while the codec frames they describe may be only a few hundred bytes at low latency and low-bitrate such as audio tracks. Streaming CMAF directly over MoQ Transport [MOQT] therefore incurs a per-object overhead that the Low Overhead Container (LOC) [LOC] avoids by carrying raw codec frames with a minimal set of metadata. LOC, however, cannot transport the per-sample CENC [CENC] metadata needed for browser EME / CDM decryption of DRM-protected live streams, nor the prft (Producer Reference Time) and emsg (DASH Event Message) boxes that CMAF may carry alongside the moof.

LOCMAF closes this gap. It exploits the observation that consecutive CMAF chunk heads within a single CMAF segment are nearly identical: the first chunk of a MOQT group is sent in full, subsequent chunks are sent as compact deltas against the previous chunk in the same group, and mdat payloads are passed through unchanged. The receiver reconstructs full CMAF chunks that are byte-compatible enough to feed unmodified MSE / EME pipelines.

This document specifies the LOCMAF object framing, the full and delta chunk encodings, the CMSF [CMSF] catalog signalling, the receiver reconstruction algorithm, and the DRM box round-trip.

1.1. Relationship to prior work

A reason that CMAF headers are big is that they have a history in the multi-sample MP4 file format. Furthermore, each individual box starts with an 8-byte header using a fixed 4-byte size and a 4-byte identifier. This is in contrast to MOQT and QUIC that use varint.

A general MP4/CMAF box can be compressed by reducing the header size by using varints and shorter ids as proposed in the Compressed MP4 draft [COMPRESSED-MP4]. LOCMAF takes a more specific approach:

* LOCMAF does *not* compress the CMAF Header (initialisation segment). The CMAF Header is carried verbatim in the catalog (see Section 5). Init compression is a one-time-per-track cost; LOCMAF’s wire-byte target is the per-chunk overhead, which an init codec cannot reduce. Carrying the CMAF Header verbatim has a second, deployment-driven benefit: a locmaf packaging track uses the *same MSF initialisation-data mechanism* as a cmaf packaging track, and when both wrap the same source they MAY refer to the same init entry (see Section 5), as being proposed for the next draft of [CMSF]. Publishers can therefore introduce LOCMAF as a more efficient wire format for clients that support it without duplicating the initialisation bytes for legacy CMAF clients —

both audiences consume the same init from the same catalog entry, and the publisher only adds the LOCMAF-encoded media track alongside the CMAF one.

* LOCMAF's goal for the per-chunk path is *functionally equivalent reconstruction*, not byte-exact reconstruction: the reconstructed CMAF chunk carries the same samples, sample metadata, and CENC metadata as the source, but byte-level details that do not affect a CMAF reader are not preserved (Section 15 lists what may differ). Given that the typical target is to feed an MSE/EME player instance, this is not a disadvantage.

A reference implementation is available [MOQLIVEMOCK]. Worked examples and diagrams are published at [LOCMAF-SITE].

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are used throughout this document:

CMAF chunk: One moof + one mdat pair, optionally preceded by at most one styp, at most one prft, and zero or more emsg boxes, as defined in [CMAF] §7.3.3.2. The smallest CMAF addressable media object.

CMAF fragment: One or more CMAF chunks whose first chunk starts at a Stream Access Point ([CMAF] §7.3.2.2). A fragment is logically a single MovieFragmentBox worth of samples; in "chunked" CMAF the samples are split across multiple smaller moof + mdat pairs.

CMAF segment: One or more CMAF fragments in decode order ([CMAF] §7.3.2.4). The segment is the typical unit of HTTP delivery in DASH and HLS-fMP4; in LOCMAF the segment corresponds to one MOQT group.

CMAF Header: The ftyp + moov pair that initialises a CMAF track. Also called an _initialisation segment_ in DASH parlance and carried as initData in MSF [MSF] / CMSF [CMSF] catalogs.

MOQT group, MOQT object: As defined in [MOQT].

LOCMAF object: A MOQT object whose payload begins with one of the top-level header IDs defined in Section 7.

Full LOCMAF chunk: A LOCMAF object whose top-level header ID is LocmafFullHeader. It carries an absolute encoding of the CMAF chunk head and serves as the in-group reference for subsequent delta objects. See Section 9.

Delta LOCMAF chunk: A LOCMAF object whose top-level header ID is LocmafDeltaHeader. It encodes differences against the most recently received full LOCMAF chunk in the same MOQT group. See Section 10.

BMDT: Abbreviation for tfdt.baseMediaDecodeTime ([ISOBMFF]).

3. MOQT Group / Object Mapping

LOCMAF assumes the following mapping from CMAF to MOQT:

* One MOQT group per CMAF segment. Group boundaries align with random access points.

* One MOQT object per CMAF chunk. Each MOQT object is a LOCMAF object carrying the (full or delta) chunk head followed by the unmodified mdat payload.

* Audio MOQT groups typically have the same duration as the video MOQT groups with which they will be muxed, to enable joint tune-in.

* Sparse tracks, such as subtitle, or event/metadata tracks, are more likely to have groups that are not aligned with video.

Per [MOQT], objects within a MOQT group are delivered in order and groups are independently decodable. LOCMAF relies on both properties: delta chunks reference the preceding chunk in the same group (see Section 10), and each group MUST begin with a LocmafFullHeader so a subscriber tuning in at a group boundary has a complete reference (see Section 7.5).

4. CMSF Catalog Signalling

A track that carries LOCMAF-encoded chunks MUST advertise:

* packaging equal to "locmaf".

* locmafVersion equal to "0.2" for this version of the specification.

This document extends the allowed packaging values defined in [MSF] to include one new entry, in the same manner that [CMSF] adds "cmaf":

Name	Value	Reference
LOCMAF	locmaf	this document

Table 1

locmafVersion is a track-level catalog field added by this document, analogous to the track-level fields [CMSF] adds for CMAF. It is present only when packaging == "locmaf" and is omitted otherwise. Its value is a string identifying the LOCMAF wire-format version; "0.2" denotes the format specified here. Receivers MUST compare it against their highest supported version and SHOULD refuse the track when the encoder advertises a version they do not implement. The set of valid locmafVersion values is governed by this document and its successors, not by an IANA registry (see Section 17).

The CMAF Header for a LOCMAF track is carried by the same MSF [MSF] mechanism that carries the CMAF Header for a plain cmaf packaging track. LOCMAF does not define its own init-carriage shape, nor add locmaf-specific catalog fields beyond locmafVersion. Whatever mechanism MSF specifies for cmaf init data — inline initData today, or whichever indirection form MSF adopts (e.g. a root-level init-data list referenced from the track entry) — applies unchanged to locmaf tracks.

A consequence: when a cmaf packaging track and a locmaf packaging track wrap the same source, they MAY refer to the same MSF init-data entry. The wrapped media is identical at the sample level and the CMAF Header bytes are identical at the byte level (see Section 5); only the per-chunk wire encoding differs.

It should be noted that cmaf and locmaf track may be mixed, e.g. video tracks using cmaf packaging while audio uses locmaf‘ dito.

5. CMAF Header Delivery

The CMAF Header for a LOCMAF track is byte-identical to the CMAF Header a plain cmaf packaging track of the same source would carry — ftyp followed by moov, followed by any optional supplemental boxes (pssh, mvex with trex, etc.). It is delivered uncompressed via the catalog using the same MSF mechanism as for cmaf packaging. There is no LOCMAF-specific CMAF Header carrier.

The moov in the CMAF Header MUST contain exactly one trak box (see Section 6).

A LOCMAF receiver:

1. Resolves the track’s MSF init-data reference (whichever form the catalog uses) to the base64-encoded CMAF Header bytes.

2. Base64-decodes the CMAF Header bytes.

3. Feeds the bytes to its MSE / decoder pipeline exactly as it would for a plain CMAF track.

4. Begins receiving LOCMAF-encoded media objects on the subscribed track and reconstructs each CMAF chunk from the LOCMAF payload.

5. Extracts the parameters required to regenerate CMAF chunks — track_ID, media timescale, trex defaults, and any track-encryption information (tenc defaults, default KID, default IV, scheme type, pattern parameters) — from the decoded CMAF Header. These values seed the reconstruction state used in step 4.

Compression of the catalog itself is out of scope for LOCMAF and handled at the MOQT / MSF [MSF] layer.

6. Scope and Publisher Requirements

LOCMAF v0.2 targets the low-latency CMAF case: short CMAF fragments composed of small CMAF chunks (often one sample per chunk), optionally carrying CENC encryption metadata. To keep the wire format minimal, the following constraints apply.

6.1. Mandatory preconditions

A LOCMAF publisher MUST ensure that:

1. *Single trak per moov.* The CMAF Header contains exactly one trak box. Multi-track ISO BMFF files MUST be demuxed before LOCMAF encoding.

2. *No key ID (KID) change within a CMAF chunk.* Key-identifier transitions MUST align with fragment (and therefore chunk) boundaries. This removes the need for sgpd / sbgp boxes in the wire format.

3. *Restricted sample_flags populations.* Per-sample, default, and first-sample sample_flags MUST populate only is_leading, sample_depends_on, sample_is_depended_on, and sample_is_non_sync_sample; the fields sample_has_redundancy, sample_padding_value, and sample_degradation_priority MUST be zero in the source. See Section 11.

4. *emsg version 1 only.* Any emsg boxes in the source MUST be version 1 per CMAF §7.4.5. See Section 12.

If a source violates any of these, the publisher MUST NOT use LOCMAF packaging for that track and MUST instead use plain CMAF or an alternative packaging (e.g. an MSF eventtimeline companion track for events). LOCMAF and plain CMAF tracks MAY coexist in the same catalog under the same namespace.

6.2. Recommended source properties

The following are recommendations whose violation costs wire bytes but does not break LOCMAF:

1. *Commensurate media timescales.* Choose a timescale so every frame has an exact integer duration (e.g. 48 000 for 48 kHz AAC, 60 000 for 60000/1001 fps video).

2. *Stable trex defaults.* Keeping trex consistent across the stream maximises what can be omitted from each chunk header.

6.3. tfdt.baseMediaDecodeTime contiguity

CMAF (§7.5.18) requires that each fragment’s BMDT equal the previous fragment’s BMDT plus the sum of its sample durations. The delta-chunk BMDT derivation defined in Section 10 relies on this property. Re-anchoring is signalled in-band by emitting an absolute BMDT override (see Section 10).

6.4. Optional encoder modes

A LOCMAF encoder MAY operate in *strict cmf2 mode*, in which it always emits the four tfhd defaults (sample duration, size, flags, sample-description index) in the full chunk header even when they match trex. This costs ~6 B per group but produces reconstructed CMAF chunks that satisfy CMAF §7.7.3 fragment self- decodability (each chunk is a single-chunk fragment in the LOCMAF mapping). It does not need to be signaled since it does not affect wire compatibility between encoders and decoders.

7. Object Framing

7.1. Top-level header IDs

LOCMAF defines two top-level header IDs:

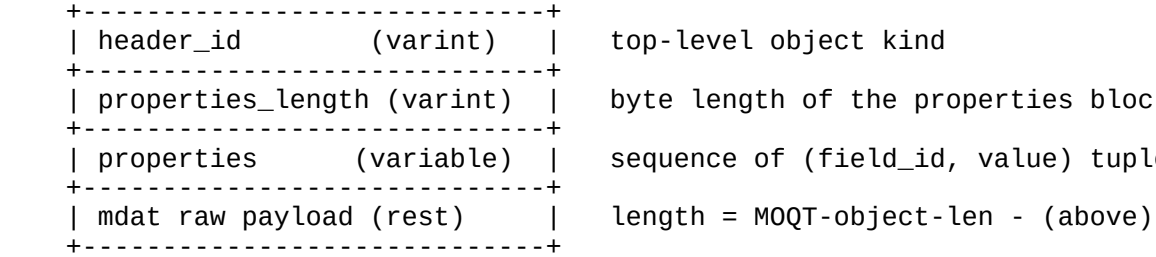
ID	Symbol	Object kind
23	LocmafFullHeader	Full LOCMAF chunk (see Section 9)
25	LocmafDeltaHeader	Delta LOCMAF chunk (see Section 10)

Table 2

Receivers MUST skip (and SHOULD log) unrecognised header_id values rather than abort. The MOQT object length terminates the unknown object cleanly.

Future extensions adding new top-level object kinds use any unassigned ID, allocated via the IANA registry defined in Section 17.

7.2. Object layout



header_id and properties_length are variable-length integers using the encoding defined by MoQ Transport [MOQT] for the session's MOQT version.

The mdat payload is the contents of the CMAF mdat box — the sample data, without the surrounding 8-byte size + 'mdat' box header. The receiver reconstructs a standard mdat box by wrapping these bytes in an 8-byte ISO BMFF header.

For event-only tracks (see Section 14) the mdat payload MAY be zero bytes; the receiver reconstructs an empty mdat box (8-byte header only).

7.3. Property encoding (parity rule)

The properties block is a flat sequence of (field_id, value) tuples. Field IDs are MOQT varints. The value encoding is determined by the parity of the ID:

* *Even ID:* scalar varint. The value is a single MOQT varint. No length prefix. In delta chunks, the encoded value is a zigzag varint (see Section 7.4) of the signed delta against the in-group reference; in full chunks it is an absolute unsigned MOQT varint.

* *Odd ID:* length-prefixed bytes. The tuple is field_id | value_length | value_bytes. The interpretation of the bytes is per-field; varint-list fields concatenate elements (each element is a zigzag varint (see Section 7.4) in delta context, an absolute MOQT varint in full context), raw-bytes fields carry opaque content. The one exception is the _signed list_ trunSampleCompositionTimeOffsets (ID 5): its elements are zigzag varints (see Section 7.4) in BOTH full and delta context, because composition time offsets are signed in trun version 1 (see Section 8.1).

Field IDs MAY appear in any order; receivers MUST tolerate any ordering. Encoders SHOULD emit IDs in ascending order to produce deterministic wire bytes.

7.4. Zigzag varint encoding

A _zigzag varint_ is a signed integer encoded as an unsigned MOQT varint by interleaving non-negative and negative values so that small-magnitude values of either sign occupy small unsigned values, and thus the shortest varint forms.

For a signed 64-bit integer n, the mapping to its unsigned zigzag representation z is:

```
encode:  z = (n << 1) ^ (n >> 63)  ; arithmetic right shift
                                   ; equivalently:
                                   ;   n >= 0:  z = 2 * n
                                   ;   n <  0:  z = -2 * n - 1

decode:  n = (z >> 1) ^ -(z & 1)   ; equivalently:
                                   ;   z even:  n =  z / 2
                                   ;   z odd:   n = -(z + 1) / 2
```

The first few mappings: 00, -11, 12, -23, 24, -35, 36, ….

The encoded z is then serialised as an unsigned MOQT varint ([MOQT]); decoders read the MOQT varint and apply the decode rule above.

This zigzag mapping is widely used in compact binary serialisation formats; the description is included here for self-containment of the LOCMAF wire format.

LOCMAF uses zigzag varints wherever a signed delta against the in-group reference is written, namely in scalar even-ID fields (see above) and per-element in varint-list odd-ID fields. Absolute values in LocmafFullHeader are encoded as plain unsigned MOQT varints, not zigzag — with one exception: the signed list trunSampleCompositionTimeOffsets (ID 5) carries zigzag varints even in a full chunk, because composition time offsets are signed in trun version 1 (the common CMAF case: B-frames make the composition/decode-time relation non-monotonic).

7.5. Full vs delta dispatch

The full-vs-delta distinction is signalled exclusively by the top-level header_id, never by the MOQT object position within a group.

1. The first object of every MOQT group MUST be a LocmafFullHeader.

2. The encoder MAY emit a LocmafFullHeader at any object position within a group, not only at object index 0. A mid-group full chunk re-anchors the in-group reference for subsequent delta chunks.

3. After receiving a LocmafFullHeader, the decoder MUST discard its in-group delta state and treat the new full chunk as the reference for any following LocmafDeltaHeader objects in the group.

4. The receiver MUST dispatch on header_id alone. It MUST NOT infer "full" from object index 0 or "delta" from object index > 0.

8. Field Reference

Field IDs are organised in blocks by source box:

Range	Block
116	fields from moof child boxes
18, 20, 22, 24	prft fields
23	styp field
25	emsg list
27	delta deletion marker

Table 3

8.1. Fields from moof child boxes

Fields drawn from boxes inside moof.traf (i.e. from trun, tfhd, tfdt, or senc). The symbol prefix names the containing box, and the field IDs are the same across both Full and Delta chunks.

ID	Symbol	Kind
1	trunSampleSizes	list
2	tfhdSampleDescriptionIndex	scalar
3	trunSampleDurations	list
4	tfhdDefaultSampleDuration	scalar
5	trunSampleCompositionTimeOffsets	signed list ‡
6	tfhdDefaultSampleSize	scalar
7	trunSampleFlags	list †
8	tfhdDefaultSampleFlags	scalar †
9	sencInitializationVector	raw bytes
10	tfdtBaseMediaDecodeTime	scalar
11	sencSubsampleCount	list
12	trunFirstSampleFlags	scalar †
13	sencBytesOfClearData	list
14	trunSampleCount	scalar
15	sencBytesOfProtectedData	list
16	sencPerSampleIVSize	scalar

Table 4

† Sample-flag fields (IDs 7, 8, 12) carry the 5-bit packed encoding defined in Section 11.

‡ The _signed list_ (ID 5) carries zigzag varints (see Section 7.4) per element in BOTH full and delta chunks, because composition time offsets are signed in trun version 1. This is the sole odd-ID list whose full-chunk elements are not plain unsigned varints.

The remaining name components map field-for-field onto the source box (e.g. tfhdDefaultSampleDuration tfhd.default_sample_duration, trunFirstSampleFlags trun.first_sample_flags). Indexing rules: per-sample lists (IDs 1, 3, 5, 7, 11) carry the samples[i].* values from their box; the per-subsample lists (IDs 13, 15) carry senc.samples[i].subsamples[j].* flattened in chunk order; and sencInitializationVector (9) is the concatenation of per-sample IVs, each sencPerSampleIVSize bytes long.

The ID space is structurally aligned with the parity rule: every default/scalar field has an even ID and every per-sample list field has an odd ID, with sencInitializationVector (9) as the documented exception (raw bytes rather than a list).

8.2. prft fields

The ProducerReferenceTimeBox (prft, [CMAF] §6.6.8, §7.3.2.4) carries an NTP-style wall-clock anchor tied to a media time. In CMAF it MAY precede any moof inside a CMAF chunk and applies to the addressable media object whose moof it precedes. LOCMAF carries it per-chunk through the following fields:

ID	Symbol	Source field	Kind
18	prftNtpTimestamp	prft.ntp_timestamp (NTP64)	scalar (absolute in full; zigzag delta in delta)
20	prftMediaTime	prft.media_time	scalar (absolute in full; zigzag delta in delta)
22	prftVersion	prft.version	scalar (default 1)
24	prftFlags	prft.flags (24-bit FullBox flags)	scalar (default 0)

Table 5

prftNtpTimestamp (ID 18) and prftMediaTime (ID 20) have no default value, so a full LOCMAF chunk that carries prft MUST include both; prftVersion (ID 22) and prftFlags (ID 24) default as given above and MAY be omitted when they match their defaults. The receiver reconstructs a prft box in the output chunk iff IDs 18 and 20 are present.

In a delta chunk, the scalar fields are zigzag-encoded deltas against the most recent full chunk in the same group that itself carried prft fields; a field left absent is unchanged from that reference. Deltas are signed because both quantities can decrease in valid CMAF streams: producer NTP clocks can be corrected backward, and composition-time reordering with B-frames can place a chunk's presentation anchor before the previous chunk's. If the previous full chunk had no prft, an encoder that begins emitting prft mid-group MUST use absolute encodings (i.e. re-anchor).

This presence-signalling supports three producer patterns with no further wire-format support:

1. *None:* no prft field is ever emitted.

2. *Per-group:* absolute prft fields on the LocmafFullHeader only, absent from subsequent LocmafDeltaHeader objects in the group.

3. *Per-chunk:* absolute prft fields on the LocmafFullHeader, delta prft fields on subsequent LocmafDeltaHeader objects.

8.2.1. prftNtpTimestamp

The value is the full 64-bit NTP timestamp defined by ISO BMFF (32-bit seconds since 1900-01-01 + 32-bit fraction) carried as a varint scalar — absolute in a full chunk, zigzag delta in a delta chunk. Full source precision is preserved so that downstream consumers can measure producer-vs-receiver clock drift from the sub-millisecond jitter around the mean inter-chunk period (a coarser representation would round away the drift signal).

NTP64 is layout-equivalent to a Q32.32 fixed-point seconds value (integer seconds in the upper 32 bits, binary fraction of a second in the lower 32 bits). Encoders and receivers MUST treat the field as a single 64-bit unsigned integer for the purposes of delta computation:

```
encoder: delta_i64 = (int64)(current_ntp64 - previous_ntp64)
         wire      = MOQT-varint(zigzag(delta_i64))

receiver: delta_i64 = unzigzag(MOQT-varint-decode(wire))
          current_ntp64 = previous_ntp64 + (uint64)delta_i64
```

The carry from fraction into seconds at a second boundary is absorbed by the 64-bit add naturally; there is no separate handling. The receiver splits the resulting 64-bit value back into the prft.ntp_timestamp seconds (upper 32) and fraction (lower 32) fields.

The steady-state delta for common frame periods lands in the 4-byte varint band: ~85.9 M units for a 20 ms (50 fps) gap, ~71.6 M for 60 fps, ~171.8 M for 25 fps. The 4-byte cost is the dominant per-chunk overhead of the prft path; encoders that do not need drift-detection precision MAY choose the per-group emission pattern instead of per-chunk (see the producer patterns above).

8.2.2. prftMediaTime

prftMediaTime carries the v1 prft.media_time field (an integer in the track's mdhd.timescale ticks) directly. It is not re-scaled. Steady-state deltas at the track timescale are small (e.g. 1024 ticks for an AAC frame at 48 kHz, 1001 ticks for a 60000/1001 fps video frame at timescale 60000) and fit in 12 varint bytes.

8.2.3. prftVersion

Defaults to 1 (the v1 prft form with a 64-bit media_time). Encoders SHOULD omit the field; receivers that find the field absent reconstruct version 1.

8.2.4. prftFlags

Carries the 24-bit FullBox flags field defined in ISO/IEC 14496-12 for ProducerReferenceTimeBox. Known values include 0 (wall-clock anchor at the encoder, the common case), 1, 2, 4, 8, and 24 (combinations of the lower bits identifying inband-event semantics and producer scope). Encoders MAY omit the field when its value is 0; receivers that find the field absent reconstruct flags = 0. The on-wire encoding is a varint scalar; LOCMAF does not constrain the value beyond what [ISOBMFF] defines.

8.2.5. prft.reference_track_ID

Not carried on the wire. The receiver reconstructs it as the track_ID of the single trak in the CMAF Header's moov (see Section 6).

8.3. styp fields

Fields whose source is the FileTypeBox payload of a styp box ([ISOBMFF], §4.3) at the top of a CMAF chunk:

ID	Symbol	Kind	Notes
23	stypBrandList	raw bytes (odd, length-prefixed)	Concatenation of 4-byte FourCC codes: major_brand followed by each compatible_brand. Length MUST be a positive multiple of 4.

Table 6

LOCMAF does not carry styp.minor_version on the wire. Per [CMAF] § 7.2, minor_version is 0 for any structural CMAF brand used as major_brand, so the reconstructed styp.minor_version is always 0.

styp fields MAY appear only in LocmafFullHeader; encoders MUST NOT emit them in LocmafDeltaHeader. Per [CMAF] § 7.3.3.1, a styp inside an addressable media object is ignored by players, so the delta path has no use for it.

If a LocmafFullHeader carries no stypBrandList, the receiver emits no styp box in the reconstructed CMAF chunk. CMAF ([CMAF] § 7.3.3.1) does not require a styp for decoding or playback; players that need brand information consult the CMAF Header’s ftyp. Per [CMAF] § 7.2, when an encoder _does_ emit stypBrandList, the reconstructed styp.minor_version is 0.

8.4. emsg field

ID	Symbol	Kind	Notes
25	emsgList	raw bytes (odd, length-prefixed)	A self-delimited concatenation of v1 emsg records in CMAF order. See Section 12 for record format.

Table 7

The presence of emsgList is independent of LocmafFullHeader vs LocmafDeltaHeader. Both kinds carry the full list when present; there is no delta encoding for emsg.

8.5. Delta deletion marker

ID	Symbol	Kind	Notes
27	deltaDeletedLocmafIDs	list (odd, length-prefixed)	List of field IDs removed since the previous moof in the same group. Used only in LocmafDeltaHeader.

Table 8

9. Full LOCMAF Chunk Encoding

A LocmafFullHeader carries an absolute encoding of one CMAF chunk's head: at most one optional styp, at most one optional prft, zero or more emsg boxes that preceded the moof in the source, and the moof itself. The mdat payload follows the property block, unchanged.

9.1. Emission rules for moof child-box fields

The encoder walks the source moof (paired with the catalog's moov) and emits each moof field only when the value cannot be derived from the moov's trex defaults:

tfhdSampleDescriptionIndex
: tfhd.HasSampleDescriptionIndex() AND value ≠ trex.default_sample_description_index

tfhdDefaultSampleDuration
: tfhd.HasDefaultSampleDuration() AND value ≠ trex.default_sample_duration

tfhdDefaultSampleSize
: all samples in the chunk have the same size AND that size ≠ trex.default_sample_size AND sample_count > 1

tfhdDefaultSampleFlags
: tfhd.HasDefaultSampleFlags() AND value ≠ trex.default_sample_flags

tfdtBaseMediaDecodeTime
: always

trunSampleCount
: always

trunFirstSampleFlags
trun.HasFirstSampleFlags()

trunSampleSizes
trun.HasSampleSize() AND sample sizes are not all equal AND sample_count > 1; the list carries sample_count − 1 values (first n−1 in chunk order)

trunSampleDurations
trun.HasSampleDuration()

trunSampleCompositionTimeOffsets
trun.HasSampleCompositionTimeOffset()

trunSampleFlags
trun.HasSampleFlags()

sencPerSampleIVSize
senc present AND per_sample_iv_size ≠ tenc.default_per_sample_iv_size

sencInitializationVector
senc present AND per_sample_iv_size > 0 AND samples carry IVs (see also Section 13.4)

sencSubsampleCount
senc present AND samples carry subsample maps

sencBytesOfClearData
same as sencSubsampleCount

sencBytesOfProtectedData
same as sencSubsampleCount

In strict cmf2 mode (see Section 6), tfhdDefaultSampleDuration, tfhdDefaultSampleSize, tfhdDefaultSampleFlags, and tfhdSampleDescriptionIndex are emitted unconditionally on the full chunk, even when they match trex.

9.1.1. Sample-size derivation

Let n = trunSampleCount and let P be the chunk's mdat-payload length (MOQT object length minus the framing already consumed). The receiver MUST derive sample sizes as follows:

* If trunSampleSizes (ID 1) is present, it carries exactly n − 1 values. sample_size[i] = listed[i] for i in [0, n−1), and sample_size[n−1] = P − sum(listed). The receiver MUST NOT consult tfhdDefaultSampleSize, trex.default_sample_size, or any other source for sample sizes in this chunk.

* Else if tfhdDefaultSampleSize (ID 6) is present, all n samples have that size.

* Else if trex.default_sample_size is non-zero, all n samples have that size.

* Else, when n == 1, the lone sample's size is P. When n > 1 and no size information is available the chunk is malformed and the receiver MUST reject it.

Correspondingly, when sample_count == 1 both trunSampleSizes and tfhdDefaultSampleSize MUST be omitted — the single sample's size is always P. When sample_count > 1 with uniform sizes the encoder MUST emit tfhdDefaultSampleSize (subject to the trex.default_sample_size equality rule) and MUST NOT emit trunSampleSizes; when sizes vary the encoder MUST emit trunSampleSizes with exactly n − 1 entries and MUST NOT emit tfhdDefaultSampleSize. Omitting the last sample size shaves one varint per chunk; using the default for uniform-size tracks (common for fixed-bitrate audio, e.g., AC-3) collapses n − 1 varints to one.

9.2. Emission rules for prft / styp / emsg fields

stypBrandList is emitted iff the source CMAF chunk preceded its moof with a styp box. When omitted, the receiver produces no styp in the reconstructed chunk.

prft fields (18, 20, 22, 24) are emitted iff the source CMAF chunk preceded its moof with a prft box, subject to the per-field defaults described in Section 8.2 (encoders MAY omit prftVersion when it is 1 and prftFlags when it is 0). All emitted values are absolute.

emsgList is emitted iff the source CMAF chunk preceded its moof with one or more v1 emsg boxes. See Section 12 for the record format.

10. Delta LOCMAF Chunk Encoding

A LocmafDeltaHeader carries only the differences between the current CMAF chunk's head and the most recently received full chunk in the same MOQT group.

10.1. Field value encoding

Each emitted field’s value is interpreted relative to its kind:

Kind	Wire encoding	Reconstruction
scalar (even ID)	zigzag varint (see Section 7.4) of current_value − previous_value	current = previous + delta
list (odd ID)	zigzag varint (see Section 7.4) per element, concatenated; element delta = current[i] − previous[i]	element-wise sum with the previous list
raw bytes (odd ID)	full new bytes verbatim	overwrite previous bytes

Table 9

The "previous value" for each field is the effective value used in the reconstruction of the previous LOCMAF chunk in the same group (or, after a mid-group LocmafFullHeader, the previous chunk starting from that re-anchor).

10.1.1. List length changes

The length of a per-sample list field in the current chunk is trunSampleCount, which is always emitted (see Section 9). This covers trunSampleDurations (ID 3), trunSampleCompositionTimeOffsets (ID 5), trunSampleFlags (ID 7), and sencSubsampleCount (ID 11). trunSampleSizes (ID 1) is the documented exception: it carries trunSampleCount − 1 entries (the last sample size is derived from the mdat-payload length per Section 9.1.1). The per-subsample list fields (IDs 13, 15) have total length equal to sum(sencSubsampleCount[i]) over the new sample count. Consequently the receiver knows len(current) for every list field before parsing the field’s payload bytes.

When len(current) ≠ len(previous) the delta rule extends as follows:

* For indices i in [0, min(len(current), len(previous))): the wire carries zigzag(current[i] − previous[i]) and the receiver reconstructs current[i] = previous[i] + delta[i].

* For indices i in [len(previous), len(current)) (current longer than previous): the wire carries zigzag(current[i]), i.e. the absolute value, equivalent to treating the missing previous entry as 0. The receiver reconstructs current[i] = delta[i].

* For indices i in [len(current), len(previous)) (current shorter than previous): no bytes are emitted for these positions. The receiver simply truncates to len(current).

The deletion list deltaDeletedLocmafIDs (ID 27) is an exception: it carries the set of field IDs deleted in _this_ chunk, encoded as plain unsigned varints (not zigzag, not deltas against a "previous deletion list"). Its length is determined by the field's byte-length prefix; the receiver reads unsigned varints until the prefix is exhausted.

10.2. tfdtBaseMediaDecodeTime is normally derived

The receiver derives the new BMDT as previous_bmdt + sum(previous_sample_durations). This is safe because CMAF requires the decode timeline to be contiguous — each fragment's baseMediaDecodeTime equals the previous fragment's plus the sum of its sample durations (see Section 6.3, which carries the CMAF reference and its normative keyword). When the source BMDT nevertheless diverges from this derivation (audio pre-roll, splicing, stream re-anchor), the encoder MUST emit tfdtBaseMediaDecodeTime (ID 10) in the delta chunk as an absolute unsigned varint (i.e. the same encoding as in a full chunk, not a zigzag delta). The receiver checks for the field first and uses its value when present.

10.3. Deletions

Delta encoding in LOCMAF is additive: a field absent from a LocmafDeltaHeader is treated as unchanged from the previous chunk. This compresses the common case (most moof fields stay stable from chunk to chunk) but on its own gives an encoder no way to signal that a field which was present in the previous chunk is genuinely gone in the current one. The deletion marker provides that signal.

The motivating case is trunFirstSampleFlags (ID 12). A SAP-1 random-access chunk emits this field to flag its first sample as a sync sample; the immediately following non-sync chunk must say "this override no longer applies" so the receiver falls back to trex.default_sample_flags for the first sample.

The deltaDeletedLocmafIDs field (ID 27) carries a varint list of field IDs that were present in the previous chunk but are no longer present. The decoder applies deletions before applying deltas.

Example: when the first chunk of a group carried trunFirstSampleFlags (a SAP-1 sync sample) and the second chunk does not, the second chunk emits ID 27 with a one-element list containing ID 12. The typical cost is two bytes — one length-prefixed list with one field ID — versus the tens to hundreds of bytes of re-anchoring.

10.4. Empty delta

An empty delta payload (properties_length == 0) is valid and means "no field changed since the previous chunk." This is the steady-state case for sample-level fragmented streams. The on-wire LOCMAF object reduces to LocmafDeltaHeader | properties_length=0 | mdat, which is two bytes plus the mdat.

10.5. prft and emsg in delta chunks

prft fields use the encoding above (scalar values become zigzag deltas against the in-group reference). emsgList carries the full new event list, with no delta encoding — see Section 12.

stypBrandList (ID 23) MUST NOT appear in a LocmafDeltaHeader.

11. Compact sample_flags Encoding

ISO BMFF sample_flags ([ISOBMFF] §8.8.3.1) is a 32-bit bit- packed field, but the bits that vary in CMAF content occupy only five of the 32. LOCMAF encodes the five varying bits in a 5-bit transport value to fit them in a single 6-bit-payload MOQT varint (leaving room for the zigzag sign bit in delta context).

11.1. Wire encoding

The 5-bit packed value (LSB first):

bit	source field
0	sample_is_non_sync_sample
12	sample_depends_on
34	sample_is_depended_on

Table 10

trunSampleFlags (ID 7) carries this 5-bit value (range 031) per sample. tfhdDefaultSampleFlags (ID 8) and trunFirstSampleFlags (ID 12) carry the same 5-bit transport.

In a full chunk the field is an unsigned varint scalar (or list). In a delta chunk it is a signed zigzag varint (or list of zigzag deltas).

11.2. Reconstruction

The receiver expands the 5-bit transport into a 32-bit sample_flags:

```
sample_flags = (is_depended_on << 22)
             | (depends_on     << 24)
             | (non_sync       << 16)
```

is_leading, sample_has_redundancy, sample_padding_value, and sample_degradation_priority are reconstructed as zero.

11.3. Encoder constraint

The encoder MUST validate that the source's sample_flags populates only the five bits listed above (see Section 6). Source content that uses other bits MUST be carried via plain CMAF packaging instead.

12. emsg Round-Trip

The DASH event message box ([DASH] § 5.10.3) carries application-defined timed metadata. CMAF (§ 7.4.5) mandates version 1 emsg boxes for in-band CMAF event messages.

LOCMAF carries emsg as a length-prefixed list of records inside the chunk header at field ID 25. Encoders MUST emit only v1 records.

12.1. Relationship to MSF eventtimeline

New MOQT deployments SHOULD use a companion MSF eventtimeline track [MSF] for event metadata rather than inline emsg. LOCMAF's emsg support exists primarily to preserve inline events in sources transcoded from DASH or CMAF Ingest [DASH-IF-INGEST].

12.2. Record format

Each record inside emsgList is a compact encoding of a v1 emsg payload:

```
record = scheme_id_uri          (varint length + UTF-8 bytes)
       | value                  (varint length + UTF-8 bytes)
       | timescale              (varint; 0 = "use track mdhd.timescale")
       | presentation_time      (encoding depends on timescale; below)
       | event_duration         (varint, ‘timescale‘ ticks; 0 = unknown)
       | id                     (varint)
       | message_data           (varint length + opaque bytes)
```

reference_track_id and version are implicit (the track this chunk belongs to, and 1 respectively).

12.2.1. timescale default

timescale == 0 means "use the track’s mdhd.timescale." Receivers write the track’s mdhd.timescale into the reconstructed emsg.timescale field when the record carried 0. Encoders MAY emit a non-zero timescale only when the source’s emsg.timescale actually differs from the track timescale.

12.2.2. presentation_time encoding

The encoding of presentation_time depends on the record’s timescale field:

* *timescale == 0 (track-timescale, delta encoding):* the field is encoded as a signed zigzag varint delta against the chunk’s BMDT (in track-timescale ticks). The reconstructed value is chunk_bmdt + delta.

* *timescale != 0 (foreign-timescale, absolute encoding):* the field is encoded as an unsigned varint carrying the absolute emsg.presentation_time directly. No delta is applied because the BMDT and the event time would live on different axes.

The receiver discriminates by inspecting the record’s timescale field, which is encoded earlier in the record.

13. DRM Box Round-Trip

LOCMAF preserves the per-sample CENC [CENC] metadata needed for EME-based decryption.

13.1. Supported schemes

LOCMAF v0.2 supports the following CENC [CENC] protection schemes, identified by the tenc.default_isProtected = 1 track defaults and the four-character scheme_type in the surrounding schm box:

* cenc: AES-128-CTR full-sample encryption. Per-sample initialization vectors are big-endian counters advanced by the per-sample encrypted-byte total ([CENC], §10.1); LOCMAF carries these IVs via sencInitializationVector and permits omission under the counter rule of Section 13.4.

* cbcs: AES-128-CBC subsample pattern encryption with a constant initialization vector taken from tenc.default_constant_iv ([CENC], §10.4); no per-sample IV appears in senc, and the pattern (default_crypt_byte_block / default_skip_byte_block) is carried verbatim with the CMAF Header.

The cbc1 and cens schemes are out of scope; sources using them MUST fall back to plain CMAF packaging.

13.2. Supported boxes

Box	Where in CMAF	LOCMAF treatment
senc	inside traf	per-sample IVs and subsample maps carried via moof field IDs 9, 11, 13, 15, 16.
saio	inside traf	not carried on the wire; recomputed by the receiver to point at the reconstructed senc.
saiz	inside traf	not carried on the wire; reconstructed from per-sample IV size and subsample counts.
tenc	inside sinf in moov	carried verbatim inside the CMAF Header.

Table 11

13.3. Unsupported boxes

The following CMAF DRM boxes are not supported by LOCMAF v0.2. Sources that require them MUST use plain CMAF packaging instead:

Box	Reason for exclusion
sgpd/sbgp	Mid-fragment key rotation via seig sample groups is out of scope; KID changes MUST align with fragment boundaries (see Section 6).
pssh (per-fragment)	License-acquisition information is signalled via the CMSF contentProtections mechanism, per [CMAF] §7.4.3.
subs	Sub-sample information for image subtitle profiles (e.g. im1i) is out of scope.

Table 12

13.4. CENC IV counter prediction (optional)

For the cenc scheme, ISO/IEC 23001-7 §9.6 specifies that the per-sample initialization vector is a big-endian counter advanced sample-by-sample by exactly ceil(total_encrypted_bytes_in_sample / 16) AES blocks. Both endpoints already see the per-sample encrypted-byte totals (sencBytesOfProtectedData) and the IV anchor (carried on the first full chunk of the track), so the receiver can derive every subsequent per-sample IV deterministically.

LOCMAF v0.2 permits encoders to omit sencInitializationVector (ID 9) when the source follows this counter rule, and requires receivers to support derivation:

* An encoder MAY omit sencInitializationVector on full and delta chunks when every per-sample IV in the chunk matches the value derived by the CENC counter rule from the previous chunk’s IVs and sencBytesOfProtectedData totals.

* A receiver MUST be able to derive per-sample IVs from the counter rule. When sencInitializationVector is absent and the scheme is cenc, the receiver advances the running IV counter and uses the derived value.

* When the source diverges from the counter rule (random IVs, mid-track counter restart, or any non-conformant strategy), the encoder MUST emit sencInitializationVector absolutely on every affected sample.

The cbcs scheme uses a constant IV from tenc.default_constant_iv carried once via the CMAF Header. There is no per-sample IV in the moof in the first place, so counter prediction does not apply to cbcs.

14. Event-Only Tracks and CMAF Ingest Compatibility

DASH-IF Ingest [DASH-IF-INGEST] defines a CMAF-based push interface for live encoders. One of its track shapes is the sparse event-only track: a CMAF track that carries no media samples (or samples of zero size) and exists purely to deliver timed events via emsg boxes attached to its chunks.

LOCMAF supports event-only tracks without any wire-format extension. A LocmafFullHeader for an event-only group sets trunSampleCount = 0, carries tfdtBaseMediaDecodeTime and emsgList, and is followed by an empty mdat payload. Subsequent chunks in the same group use LocmafDeltaHeader with the absolute-BMDT override pattern (see Section 10) because the zero sample-count produces no derivation increment.

Two encoder strategies are valid:

1. *Absolute BMDT per chunk.* The delta chunk emits tfdtBaseMediaDecodeTime explicitly. Costs an extra varint per chunk; recommended for sparse event-only tracks.

2. *Synthetic per-chunk sample.* The encoder sets sample_count = 1 with a default_sample_duration equal to the intended per-chunk advancement and a zero-size sample. BMDT derivation works without override. Matches how DASH-IF Ingest commonly shapes sparse metadata tracks (urim, stpp).

For new MOQT deployments, MSF eventtimeline [MSF] is the preferred mechanism for event metadata. LOCMAF event-only tracks are intended for gateways that transit-relay CMAF Ingest content unchanged across MOQT.

15. Receiver Reconstruction

A receiver maintains, per subscribed track:

1. The track’s CMAF Header (from the catalog), parsed for the single trak’s track_ID, trex defaults, tenc defaults, and mdhd.timescale.

2. An in-group "previous chunk" state, populated from the most recent LocmafFullHeader and updated by each subsequent LocmafDeltaHeader. Discarded on group boundaries and on mid-group LocmafFullHeader re-anchors (see Section 7.5).

For each LOCMAF object, the receiver:

1. Reads header_id and dispatches per Section 7.5.

2. Reads properties_length and the property block.

3. Decodes property tuples per the parity rule and the per-field rules above.

4. Applies the decoded fields to the previous-chunk state to produce the absolute field values for the current chunk.

5. Reconstructs the CMAF chunk:

    1. Synthesises a styp box from stypBrandList when present; omits it otherwise.

    2. Synthesises a prft box from any prft fields present; omits it when no prft field is present.

    3. Synthesises one or more v1 emsg boxes from emsgList when present; omits them otherwise.

    4. Synthesises the moof box (mfhd, traf with tfhd, tfdt, trun, optionally senc/saio/saiz) from the moof fields and the CMAF Header’s trex / tenc defaults. The LOCMAF wire format carries neither the track_ID nor the tfhd flags, so the receiver supplies them from the init and from the CMAF rules:

        * The synthesised tfhd.track_ID MUST be set to the track_ID of the single trak in the CMAF Header’s moov (item 1).

        * The synthesised tfhd MUST set the default-base-is-moof flag (tf_flags 0x020000) and MUST NOT set base-data-offset-present, as required by [CMAF]; sample data offsets are therefore relative to the start of the containing moof ([ISOBMFF]).

    5. Wraps the mdat payload bytes in an 8-byte mdat box header.

6. Feeds the reconstructed chunk to the local CMAF reader / MSE pipeline.

The reconstructed CMAF chunk is *functionally equivalent* to the source chunk: every sample has the same size, decode time, presentation time, flags, and CENC metadata, and the chunk feeds an MSE / EME pipeline identically to the source. Byte-level identity with the source moof is not preserved. Implementations MAY differ in:

* The exact ordering of saio / saiz / senc and other generated boxes inside the reconstructed traf, provided the ordering is legal CMAF.

* The trun.tr_flags packing chosen on reconstruction.

* Whether tfhd defaults that match trex appear in the reconstructed tfhd (they SHOULD when the encoder ran in strict cmf2 mode; they MAY otherwise).

A receiver MUST NOT depend on byte-level identity with the source CMAF stream. A downstream consumer that needs a specific CMAF byte layout MUST repackage the output of the LOCMAF receiver to produce the desired form.

16. Security Considerations

LOCMAF is a compression layer over CMAF media and does not introduce new authentication or confidentiality mechanisms. It is intended to be used over MOQT [MOQT], which inherits QUIC’s transport security. Per-sample encryption metadata defined by [CENC] is preserved through the LOCMAF round-trip; LOCMAF neither weakens nor strengthens the underlying DRM scheme.

A receiver MUST validate that reconstructed moof, prft, and emsg boxes are well-formed before passing them to a media pipeline. Malformed deltas could otherwise be used to construct ISO BMFF [ISOBMFF] boxes with inconsistent field lengths. Specifically:

* The receiver MUST bound the size of any reconstructed per-sample list against trunSampleCount.

* The receiver MUST verify that the sum of reconstructed sencBytesOfClearData and sencBytesOfProtectedData for each sample equals the sample’s size.

* The receiver MUST verify that the CENC-IV counter, if derived, does not advance past the per-sample-IV-size range.

The CENC IV counter-prediction optimisation (Section 13.4) does not disclose key material and produces the same IV stream a conformant encoder would have transmitted; it does not weaken CENC.

Replay considerations within a MOQT group are inherited from MOQT — LOCMAF adds no new replay attack surface.

17. IANA Considerations

This document does not register the "locmaf" packaging value or the locmafVersion catalog field with IANA (see Section 17.1); it does request a registry for the LOCMAF top-level header IDs (see Section 17.2).

17.1. Catalog packaging value and locmafVersion

Following the precedent of [CMSF] — whose IANA Considerations record no IANA actions for its "cmaf" packaging value or its added catalog fields — this document registers neither of the following with IANA:

* packaging: "locmaf" extends the [MSF] packaging-values table (see Section 4). [MSF] defines no IANA registry for packaging values; new values are introduced by the documents that define them.

* locmafVersion is a track-level catalog field defined by this document (see Section 4). [MSF] permits documents and producers to define additional catalog fields and maintains no IANA registry of them.

The set of valid locmafVersion values — "0.2" for this document — is governed by this specification and its successors; no IANA action is required. Should a future [MSF] revision introduce an IANA packaging-value registry, the "locmaf" value SHOULD be registered there, while locmafVersion and its values remain document-governed.

17.2. LOCMAF Top-Level Header IDs

This document defines a new registry for LOCMAF top-level header IDs.

ID	Symbol	Reference
23	LocmafFullHeader	this document
25	LocmafDeltaHeader	this document

Table 13

All other IDs in the unsigned varint range are available for assignment via Specification Required ([RFC8126]).

LOCMAF property field IDs are part of this document’s wire format and are not registered with IANA. New field IDs are introduced through revisions of this specification, signalled by a bump of the locmafVersion catalog value (see Section 4). The full field-ID assignment for this version is given in Section 8.

18. References

18.1. Normative References

[CMSF] Law, W., "CMSF- a CMAF compliant implementation of MOQT Streaming Format", Work in Progress, Internet-Draft, draft-ietf-moq-cmsf-00, 1 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-cmsf-00>.

[MOQT] Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-18, 12 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-18>.

[MSF] Law, W. and S. Nandakumar, "MOQT Streaming Format", Work in Progress, Internet-Draft, draft-ietf-moq-msf-01, 2 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-msf-01>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

18.2. Informative References

[CENC] "Information technology — MPEG systems technologies — Part 7: Common encryption in ISO base media file format files", ISO/IEC 23001-7:2023, 2023.

[CMAF] "Information technology — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media", ISO/IEC 23000-19:2024, 2024.

[COMPRESSED-MP4]
Curley, L., "Compressed MP4", Work in Progress, Internet-Draft, draft-lcurley-compressed-mp4-00, 16 March 2026, <https://datatracker.ietf.org/doc/html/draft-lcurley-compressed-mp4-00>.

[DASH] "Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats", ISO/IEC 23009-1:2026, 2026.

[DASH-IF-INGEST]
"DASH-IF Live Media Ingest Protocol", n.d., <https://dashif.org/Ingest/>.

[ISOBMFF] "Information technology — Coding of audio-visual objects — Part 12: ISO base media file format", ISO/IEC 14496-12:2026, 2026.

[LOC] Zanaty, M., Nandakumar, S., and P. Thatcher, "Low Overhead Media Container", Work in Progress, Internet-Draft, draft-ietf-moq-loc-02, 15 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-loc-02>.

[LOCMAF-SITE]
"LOCMAF — Low Overhead CMAF for MoQ", n.d., <https://locmaf.dev>.

[MOQLIVEMOCK]
"moqlivemock — Reference LOCMAF server and tooling", n.d., <https://github.com/Eyevinn/moqlivemock>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

Acknowledgments

The initial version of LOCMAF was developed as part of the Master Thesis work of Hugo Bjrs at Eyevinn Technology, supervised by Torbjrn Einarsson. The authors thank the Media over QUIC working group, in particular the authors and contributors to [MOQT] and [CMSF], for the prior art this work builds on.

Authors' Addresses

Torbjrn Einarsson
Eyevinn Technology
Email: torbjorn.einarsson@eyevinn.se

Hugo Bjrs
KTH
Email: hugobjoers@gmail.com