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

W. Law
Akamai
S. Nandakumar
Cisco

2 June 2026

MOQT Streaming Format
draft-ietf-moq-msf-01

Abstract

This document specifies the MOQT Streaming Format, designed to operate on Media Over QUIC Transport.

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://moq-wg.github.io/msf/ draft-ietf-moq-msf.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-moq-msf/.

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/moq-wg/msf.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 4 December 2026.

Table of Contents

1. Introduction

MOQT Streaming Format (MSF) is a media format designed to deliver LOC [LOC] compliant media content over Media Over QUIC Transport (MOQT) [MoQTransport]. MSF works by fragmenting the bitstream into objects that can be independently transmitted. MSF leverages a catalog format to describe the output of the original publisher. MSF specifies how content should be packaged and signaled, defines how the catalog communicates the content, specifies prioritization strategies for real-time and workflows for beginning and terminating broadcasts. MSF also details how end-subscribers may perform adaptive bitrate switching. MSF is targeted at real-time and interactive levels of live latency, as well as VOD content.

This document describes version 1 of the streaming format.

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.

This document uses the conventions detailed in Section 1.3 of [RFC9000] when describing the binary encoding.

3. Scope

The purpose of MSF is to provide an interoperable media streaming format operating over [MoQTransport]. Interoperability implies that:

* An original publisher can package incoming media content into tracks, prepare a catalog and announce the availability of the content to an MOQT relay. Media content refers to audio and video data, as well as ancillary data such as captions, subtitles, accessibility and other timed-text data.

* An MOQT relay can process the announcement as well as cache and propagate the tracks, both to other relays or to the final subscriber.

* A final subscriber can parse the catalog, request tracks, decode and render the received media data.

MSF is intended to provide a format for delivering commercial media content. To that end, the following features are within scope:

* Video codecs - all codecs supported by [LOC]

* Audio codecs - all audio codecs supported by [LOC]

* Catalog track - describes the availability and characteristics of content produced by the original publisher.

* Timeline track - describes the relationship between MOQT Group and Object IDs to media time.

* Token-based authorization and access control

* Captions + Subtitles - support for [WEBVTT] and [IMSC1] transmission

* Latency support across multiple regimes (thresholds are informative only and describe the delay between the original publisher placing the content on the wire and the final subscriber rendering it)

* Real-time - less than 500ms

* Interactive - between 500ms and 2500ms

* Standard - above 2500ms

* VOD latency - content that was previously produced, is no longer live and is available indefinitely.

* Content encryption

* ABR between time-synced tracks - subscribers may switch between tracks at different quality levels in order to maximize visual or audio quality under conditions of throughput variability.

* Capable of delivering interstitial advertising.

* Logs and analytics management - support for the reporting of client-side QoE and relay delivery actions via publish tracks using [MOQLOG] and [MOQMETRICS].

Initial versions of MSF will prioritize basic features necessary to exercise interoperability across delivery systems. Later versions will add commercially necessary features.

4. Media packaging

MSF delivers LOC [LOC] packaged media bitstreams.

4.1. LOC packaging

This specification references Low Overhead Container (LOC) [LOC] to define how audio and video content is packaged. With this packaging mode, each EncodedAudioChunk or EncodedVideoChunk sample is placed in a separate MOQT Object. Samples that belong to the same Group of Pictures (GOP) MUST be placed within the same MOQT Group.

When LOC packaging is used for a track, the catalog packaging attribute (Section 5.2.4) MUST be present and it MUST be populated with a value of "loc".

4.2. Time-alignment

MSF Tracks MAY be time-aligned. Those that are, are subject to the following requirements:

* Tracks advertised in the catalog as belonging to a common alternate group MUST be time-aligned.

* The render duration of the first media object of each equally numbered MOQT Group, after decoding, SHOULD have overlapping presentation time.

A consequence of this restriction is that an MSF receiver SHOULD be able to cleanly switch between time-aligned media tracks at group boundaries.

If time-aligned media tracks do not have overlapping presentation time at equally-numbered group boundaries, then an alternate mechanism, not defined by this specification, must be provided to the client to enable it to switch smoothly between time-aligned, but numerically dissimilar, Group IDs.

4.3. Content protection and encryption

MSF supports end-to-end encryption of media content using MoQ Secure Objects [SecureObjects]. When encryption is enabled, the payload of LOC-packaged media objects is encrypted and authenticated, while relays can still route content based on unencrypted header information.

4.3.1. Encryption scheme signaling

The encryption scheme and parameters are signaled in the catalog using the following track-level fields:

* encryptionScheme Section 5.2.38 - identifies the encryption mechanism
* cipherSuite Section 5.2.39 - specifies the AEAD algorithm
* keyId Section 5.2.40 - identifies the key material for decryption
* trackBaseKey Section 5.2.41 - the base key material for this track

When the encryptionScheme field is present in a track definition, subscribers MUST decrypt the object payload using the specified scheme before processing.

4.3.2. Key management

The keyId and trackBaseKey values are obtained from an external key management system and the mechanism for obtaining these values is out of scope for this specification. Examples of key management systems include MLS-based key distribution [E2EE-MLS] or other out-of-band key exchange mechanisms.

Depending on the key management mechanism in use, a keyId MAY be scoped to:

* A single track

* A single MoQ Session

* Multiple tracks across one or more MoQ sessions

Publishers and subscribers MUST use the same key management system and agree on the keyId scope semantics for interoperable operation.

4.3.3. Recommended encryption scheme

The RECOMMENDED encryption scheme for MSF is "moq-secure-objects". Implementations supporting content encryption MUST implement the "moq-secure-objects" scheme as defined in [SecureObjects].

When using the "moq-secure-objects" scheme:

* The cipherSuite field MUST be present and set to a supported cipher suite value

* The keyId field MUST be present to identify the key material

* The trackBaseKey field MUST be present to provide the base key material

4.3.4. Encrypted object structure

For LOC-packaged tracks with encryption enabled (see [SecureObjects], Section 4):

* The immutable header extensions (including Group ID and Object ID) remain in plaintext and are authenticated

* The object payload is encrypted and authenticated using the specified cipher

* Private header extensions (type 0xA) are encrypted alongside the payload

5. Catalog

A Catalog is an MOQT Track that provides information about the other tracks being produced by a MSF publisher. A Catalog is used by MSF publishers for advertising their output and for subscribers in consuming that output. The payload of the Catalog object is opaque to Relays and can be end-to-end encrypted. The Catalog provides the names and namespaces of the tracks being produced, along with the relationship between tracks, properties of the tracks that consumers may use for selection and any relevant initialization data.

The catalog track MUST have a case-sensitive Track Name of "catalog".

A catalog object MAY be independent of other catalog objects or it MAY represent a delta update of a prior catalog object. The first catalog object published within a new group MUST be independent and MUST provide a complete catalog that does not require any prior catalog object for interpretation. Any catalog updates that precede the first Object of the latest Group MUST be ignored.

A catalog object SHOULD be published only when the availability of tracks changes, or after a period of time has passed such that the catalog object might fall out of cache in a delivery network.

Each catalog update MUST be mapped to an MOQT Object. All catalog updates, both independent and delta, MUST be mapped to MOQT sub-group 0. The first Object (with Object ID 0) in any Group in a catalog track MUST hold an independent copy of the catalog. All subsequent Objects within that Group (i.e Objects IDs >= 1) MUST hold a delta update. As soon as an independent update is produced, it MUST be placed at the start of a new Group.

Subscribers accessing the catalog MUST use SUBSCRIBE with a Joining FETCH (offset = 0) in order to obtain the latest complete catalog along with all subsequent catalog objects, including delta updates, that follow.

A catalog is a JSON [JSON] document, comprised of a series of mandatory and optional fields. At a minimum, a catalog MUST provide all mandatory fields. Some fields are conditional depending on the type of content carried. A producer MAY add additional fields to the ones described in this draft. Custom field names MUST NOT collide with field names described in this draft.

A parser MUST ignore fields it does not understand.

5.1. Root Catalog Fields

Table 1 lists the fields defined at the root of the catalog JSON object.

Field	Name	Definition
MSF version	version	Section 5.1.1
Generated at	generatedAt	Section 5.1.2
Is Complete	isComplete	Section 5.1.3
Tracks	tracks	Section 5.1.4
Publish tracks	publishTracks	Section 5.1.5
Delta update	deltaUpdate	Section 5.1.6
Initialization Data List	initDataList	Section 5.1.7

Table 1

5.1.1. MSF version

Required: Yes JSON Type: String Location: Root Catalog

Specifies the version of MSF referenced by this catalog. There is no guarantee that future catalog versions are backwards compatible and field definitions and interpretation may change between versions. A subscriber MUST NOT attempt to parse a catalog version which it does not understand.

For usage against IETF Internet-Draft releases, follow the convention of specifying the version as "draft-XX". For example "draft-03" refers to the -03 draft release.

5.1.2. Generated at

Required: Optional JSON Type: Number Location: Root Catalog

The wallclock time at which this catalog instance was generated, expressed as the number of milliseconds that have elapsed since January 1, 1970 (midnight UTC/GMT). This field SHOULD NOT be included if the isLive field is false.

5.1.3. Is Complete

Required: Optional JSON Type: Boolean Location: Root Catalog

A catalog-level indication that the broadcast is complete. This is a commitment that all tracks are complete, no new tracks will be added to the catalog, and no new content will be published on any track. Note that even if all individual tracks have isLive Section 5.2.7 set to FALSE, new tracks could still be added to the catalog until isComplete is set to TRUE. This field MUST NOT be included if it is FALSE. This field MUST NOT be removed from a catalog once it has been added.

5.1.4. Tracks

Required: Yes JSON Type: Array Location: Root Catalog

An array of track objects Section 5.2.1.

5.1.5. Publish tracks

Location: R Required: Optional JSON Type: Array

An array of publish track objects. Publish tracks define tracks to which the subscriber can publish data, such as logs, metrics, or other QoE data. This enables bi-directional communication where the subscriber acts as a publisher for specific tracks. Each publish track object follows the same structure as a regular track object Section 5.2.1 but is used for the reverse direction of data flow.

5.1.6. Delta update

Required: Optional JSON Type: Array Location: Root Catalog

An ordered Array of operation objects that specify changes to apply to the catalog. If this field is present, the catalog represents a delta (or partial) update with a restricted set of fields and special processing rules - see Section 5.3. If this field is absent, the catalog is independent.

Operations are applied sequentially in the order they appear in the array. Each operation object MUST contain an "op" field indicating the operation type, and a "tracks" field containing an Array of track objects Section 5.2.1. The following operation types are defined:

* "add" - Add new tracks that have not previously been declared. The value of the "tracks" field is an Array of track objects Section 5.2.1.

* "remove" - Remove tracks that have been previously declared. The value of the "tracks" field is an Array of track objects Section 5.2.1. Each track object MUST include a Track Name Section 5.2.3 field, MAY include a Track Namespace Section 5.2.2 field, and MUST NOT hold any other fields.

* "clone" - Clone new tracks from previously declared tracks. The value of the "tracks" field is an Array of track objects Section 5.2.1. Each track object MUST include a Parent Name Section 5.2.33 field and MAY include a Parent namespace Section 5.2.34 field. The cloned track inherits all attributes from the parent except the Track Name which MUST be new. Attributes redefined in the track object override inherited values.

5.1.7. Initialization Data List

Required: Optional JSON Type: Array Location: Root Catalog

An array of initialization reference objects. Each initialization reference object has the following fields:

* id : a string defining a reference to this initialization data which is unique within the scope of the catalog.

* type: as string defining the type of reference. This version of the specification defines a single allowed type, per the table below

Type	Data field definition
inline	Base64 [BASE64] encoded initialization data

Table 2

* data: a string holding the init payload as defined by the type.

The Initialization Data List, if present, MUST be located after the tracks array in the root of the JSON catalog. The purpose of this is to improve the human readability of the catalog tracks by moving the verbose init data towards the end of the document.

5.2. Track Object Fields

Table 2 lists the fields defined within each track object.

Field	Name	Definition
Track namespace	namespace	Section 5.2.2
Track name	name	Section 5.2.3
Packaging	packaging	Section 5.2.4
Event timeline type	eventType	Section 5.2.5
Is Live	isLive	Section 5.2.7
Target latency	targetLatency	Section 5.2.8
Buffers	buffers	Section 5.2.9
Track role	role	Section 5.2.6
Track label	label	Section 5.2.10
Render group	renderGroup	Section 5.2.11
Alternate group	altGroup	Section 5.2.12
Initialization ref	initRef	Section 5.2.13
Dependencies	depends	Section 5.2.14
Temporal ID	temporalId	Section 5.2.16
Spatial ID	spatialId	Section 5.2.17
Codec	codec	Section 5.2.18
Mime type	mimeType	Section 5.2.19
Framerate	framerate	Section 5.2.20
Timescale	timescale	Section 5.2.21
Maximum Bitrate	bitrate	Section 5.2.22
Average Bitrate	avgBitrate	Section 5.2.23
Maximum GOP Duration	maxGopDuration	Section 5.2.24
Maximum Group Duration	maxGroupDuration	Section 5.2.25

Width	width	Section 5.2.26
Height	height	Section 5.2.27
Audio sample rate	samplerate	Section 5.2.28
Channel configuration	channelConfig	Section 5.2.29
Display width	displayWidth	Section 5.2.30
Display height	displayHeight	Section 5.2.31
Language	lang	Section 5.2.32
Parent name	parentName	Section 5.2.33
Parent namespace	parentNamespace	Section 5.2.34
Track duration	trackDuration	Section 5.2.35
Authorization Info	authInfo	Section 5.2.42
Accessibility	accessibility	Section 5.2.44

Table 3

5.2.1. Tracks object

A track object is a JSON Object containing a collection of fields whose location is specified in Table 2.

5.2.2. Track namespace

Required: Optional JSON Type: String Location: Track Object

The name space under which the track name is defined. See section 2.3 of [MoQTransport]. The track namespace is optional. If it is not declared within a track, then each track MUST inherit the namespace of the catalog track. A namespace declared in a track object overrides any inherited name space.

5.2.3. Track name

Required: Yes JSON Type: String Location: Track Object

A string defining the name of the track. See section 2.3 of [MoQTransport]. Within the catalog, track names MUST be unique per namespace.

5.2.4. Packaging

Required: Yes JSON Type: String Location: Track Object

A string defining the type of payload encapsulation. Allowed values are strings as defined in Table 3.

Name	Value	Reference
LOC	loc	See RFC XXXX
Media Timeline	mediatimeline	See Section 7
Event Timeline	eventtimeline	See Section 8
MoQ Log	moqlog	See [MOQLOG]
MoQ Metrics	moqmetrics	See [MOQMETRICS]

Table 4

Table 3: Allowed packaging values

5.2.5. Event timeline type

Required: Optional JSON Type: String Location: Track Object

A String defining the type & structure of the data contained within the data field of the Event timeline track. Types are defined by the application provider and are not centrally registered. Implementers are encouraged to use a unique naming scheme, such as Reverse Domain Name Notation, where domain name components are listed in reverse order (e.g., "com.example.myeventtype"), to avoid naming collisions. This field is required if the Section 5.2.4 value is "eventtimeline". This field MUST NOT be used if the packaging value is not "eventtimeline".

5.2.6. Track role

Required: Optional JSON Type: String Location: Track Object

A string defining the role of content carried by the track. Specified roles are described in Table 4. These role values are case-sensitive.

This role field MAY be used in conjunction with the Mimetype Section 5.2.19 to fully describe the content of the track.

Table 4: Reserved track roles

Role	Description
audiodescription	An audio description for visually impaired users
video	Visual content
audio	Audio content
mediatimeline	An MSF media timeline Section 7
eventtimeline	An MSF event timeline Section 8
caption	A textual representation of the audio track
subtitle	A transcription of the spoken dialogue
signlanguage	A visual track for hearing impaired users.
log	A log publishing track per [MOQLOG].
metrics	A metrics publishing track per [MOQMETRICS].

Table 5

Custom roles MAY be used as long as they do not collide with the specified roles.

5.2.7. Is Live

Required: Yes JSON Type: Boolean Location: Track Object

A track-level indication of whether new Objects will be added to this specific track. True if new Objects will be added to the track. False if no new Objects will be added to the track. A False value is sent under two possible conditions: * the publisher of a previously live track has ended the track. * the track is Video-On-Demand (VOD) and was never live. A True value MUST never follow a False value.

5.2.8. Target latency

Required: Optional JSON Type: Number Location: Track Object

The target latency in milliseconds. Target latency is defined as the offset in wallclock time between when content was encoded and when it is displayed to the end user. For example, if a frame of video is encoded at 10:08:32.638 UTC and the target latency is 5000, then that frame should be rendered to the end-user at 10:08:37.638 UTC. If isLive is FALSE, this field MUST be ignored. All tracks belonging to the same render group MUST have identical target latencies. All tracks belonging to the same alternate group MUST have identical target latencies. If this field is absent from the track definition, and isLive is TRUE, then the player MAY choose the latency with which it renders the content.

This property MUST NOT be present if the buffers Section 5.2.9 property is present within a track definition.

5.2.9. Buffers

Required: Optional JSON Type: Object Location: Track Object

An object defining a set of target buffers. Buffer is defined as the duration of media data that MUST be buffered before decoding commences. This is typically known as a forward or jitter-buffer in a media player.Players with identical buffer lengths are likely to be synchronized. The target buffer object has these keys:

* target : defines the target buffer in integer milliseconds. Players SHOULD attempt to stabilize playback at this value.

* min : defines the minimum buffer in milliseconds. Players SHOULD NOT operate below this value.

* max : defines the maximum buffer in milliseconds. Players SHOULD NOT operate above this value.

Keys are optional. Unknown keys in the target buffer object MUST be ignored.

If isLive is FALSE, this target buffer property MUST be ignored. All tracks belonging to the same render group MUST have identical target buffers. All tracks belonging to the same alternate group MUST have identical target buffers. If this field is absent from the track definition, and isLive is TRUE, then the player MAY choose the buffers with which it conducts playback.

This property MUST NOT be present if the target latency Section 5.2.8 property is present within a track definition.

5.2.10. Track label

Required: Optional JSON Type: String Location: Track Object

A string defining a human-readable label for the track. Examples might be "Overhead camera view" or "Deutscher Kommentar". Note that the [JSON] spec requires UTF-8 support by decoders.

5.2.11. Render group

Required: Optional JSON Type: Number Location: Track Object

An integer specifying a group of tracks which are designed to be rendered together. Tracks with the same group number SHOULD be rendered simultaneously and are designed to accompany one another. A common example would be tying together audio and video tracks.

5.2.12. Alternate group

Required: Optional JSON Type: Number Location: Track Object

An integer specifying a group of tracks which are alternate versions of one-another. Alternate tracks represent the same media content, but differ in their selection properties. Alternate tracks MUST have matching media time sequences. A subscriber typically subscribes to one track from a set of tracks specifying the same alternate group number. A common example would be a set video tracks of the same content offered in alternate bitrates.

5.2.13. Initialization reference

Required: Optional JSON Type: String Location: Track Object

A string pointing at the id field of an entry in the Initialization Data List Section 5.1.7.

5.2.14. Dependencies

Required: Optional JSON Type: Array Location: Track Object

Certain tracks may depend on other tracks for decoding. Dependencies holds an array of track names Section 5.2.3 on which the current track is dependent. Since only the track name is signaled, the namespace of the dependencies is assumed to match that of the track declaring the dependencies.

5.2.15. Template

Required: Optional JSON Type: Array Location: Track Object

A media timeline template for tracks with fixed-duration segments. It specifies the relationship between media time, MOQT Location, and wallclock time through starting points and intervals. See Section 7.4 for the complete format specification, field definitions, and computation formulas.

Tracks that include a template field SHOULD NOT also have a separate media timeline track, as the template provides equivalent functionality. Different tracks (e.g., audio and video) MAY have independent template values to accommodate different group durations.

5.2.16. Temporal ID

Required: Optional JSON Type: Number Location: Track Object

A number identifying the temporal layer/sub-layer encoding of the track, starting with 0 for the base layer, and increasing by 1 for the next higher temporal fidelity.

5.2.17. Spatial ID

Required: Optional JSON Type: Number Location: Track Object

A number identifying the spatial layer encoding of the track, starting with 0 for the base layer, and increasing by 1 for the next higher fidelity.

5.2.18. Codec

Required: Conditional JSON Type: String Location: Track Object

A string defining the codec used to encode the track. For LOC packaged content, the string codec registrations are defined in Sect 3 and Section 4 of [WEBCODECS-CODEC-REGISTRY]. This property MUST be

specified for tracks which have an inherent codec associated with them (e.g., audio and video tracks). It is not required for raw data tracks or event streams.

5.2.19. Mimetype

Required: Optional JSON Type: String Location: Track Object

A string defining the mime type [MIME] of the track.

5.2.20. Framerate

Required: Optional JSON Type: Number Location: Track Object

A number defining the framerate of the track, expressed as frames per second. This property SHOULD only accompany video or other frame-based content.

5.2.21. Timescale

Required: Optional JSON Type: Number Location: Track Object

The number of time units that pass per second.

5.2.22. Maximum Bitrate

Required: Conditional JSON Type: Number Location: Track Object

A number defining the maximum bitrate of the track, expressed in bits per second. This property MUST be specified for audio and video tracks.

5.2.23. Average Bitrate

Required: Optional JSON Type: Number Location: Track Object

A number defining the average bitrate of the track, over the lifetime of the track, expressed in bits per second.

5.2.24. Maximum GOP Duration

Required: Optional JSON Type: Number Location: Track Object

A number defining the maximum duration, expressed in milliseconds, between successive independently decodable points (random access points) in the media track. This property SHOULD only accompany video tracks.

5.2.25. Maximum Group Duration

Required: Optional JSON Type: Number Location: Track Object

A number defining the maximum duration, expressed in milliseconds, of any MOQT Group in this track. This value helps subscribers estimate buffer requirements for the track.

5.2.26. Width

Required: Optional JSON Type: Number Location: Track Object

A number expressing the maximum encoded width of the video frames in pixels. This property SHOULD accompany tracks which have a visual representation.

5.2.27. Height

Required: Optional JSON Type: Number Location: Track Object

A number expressing the maximum encoded height of the video frames in pixels. This property SHOULD accompany tracks which have a visual representation.

5.2.28. Audio sample rate

Required: Conditional JSON Type: Number Location: Track Object

The number of audio frame samples per second. This property MUST accompany tracks for which audio codecs are specified.

5.2.29. Channel configuration

Required: Conditional JSON Type: String Location: Track Object

A string specifying the audio channel configuration. A string is used in order to provide the flexibility to describe complex channel configurations for multi-channel and Next Generation Audio schemas. This property MUST accompany tracks for which audio codecs are specified.

5.2.30. Display width

Required: Optional JSON Type: Number Location: Track Object

A number expressing the intended display width of the track content in pixels. This property SHOULD only accompany tracks which have a visual representation.

5.2.31. Display height

Required: Optional JSON Type: Number Location: Track Object

A number expressing the intended display height of the track content in pixels. This property SHOULD only accompany tracks which have a visual representation.

5.2.32. Language

Required: Optional JSON Type: String Location: Track Object

A string defining the dominant language of the track. The string MUST be one of the standard Tags for Identifying Languages as defined by [LANG].

5.2.33. Parent name

Required: Optional JSON Type: String Location: Track Object

A string defining the parent track name Section 5.2.3 to be cloned. This field MUST only be included inside a clone operation in a delta update Section 5.1.6.

5.2.34. Parent namespace

Required: Optional JSON Type: String Location: Track Object

A string defining the parent track namespace Section 5.2.2 to be cloned. This field MUST only be included inside a clone operation in a delta update Section 5.1.6. If this field is missing from a clone operation, then the namespace of the catalog is assumed.

5.2.35. Track duration

Required: Optional JSON Type: Number Location: Track Object

The duration of the track expressed in integer milliseconds. This field MUST NOT be included if the isLive Section 5.2.7 field value is true.

5.2.36. Connection URI

Required: Optional JSON Type: String Location: Track Object

A string containing the MOQT connection endpoint URI for the publish track. When specified, the subscriber MUST establish a new MOQT connection to this URI for publishing the track data. If this field is absent, the subscriber SHOULD reuse the existing MOQT connection that was used to receive the catalog.

The URI MUST be a valid MOQT endpoint URI as defined by [MoQTransport] (Sect 3.1.1). Examples include "moqt://logs.example.com:4443", "moqt://metrics.example.com:8443", or "https://logs.example.com/moqt".

5.2.37. Token

Required: Optional JSON Type: String Location: Track Object

A string containing an authentication token or credential for the track. For publish tracks, this token authorizes the subscriber to publish data to the specified track. The format and validation of the token is application-specific.

5.2.38. Encryption scheme

Required: Optional JSON Type: String Location: Track Object

A string identifying the encryption scheme used to protect the track content. The default and RECOMMENDED value is "moq-secure-objects" as defined in [SecureObjects]. If this field is absent, the track content is unencrypted.

Table 5: Registered encryption schemes

Name	Value	Reference
MoQ Secure Objects	moq-secure-objects	[SecureObjects]

Table 6

Custom encryption schemes MAY be used. Custom scheme names SHOULD use Reverse Domain Name Notation to avoid collisions (e.g., "com.example.custom-encryption").

5.2.39. Cipher suite

Required: Optional JSON Type: String Location: Track Object

A string identifying the AEAD cipher suite used for encryption. This field MUST be present when encryptionScheme is specified. For the "moq-secure-objects" scheme, the following cipher suites are defined:

Table 6: Cipher suites for moq-secure-objects

Name	Value	Tag Size
AES-128-GCM-SHA256	aes-128-gcm-sha256	128 bits
AES-256-GCM-SHA512	aes-256-gcm-sha512	128 bits
AES-128-CTR-HMAC-SHA256-80	aes-128-ctr-hmac-sha256-80	80 bits

Table 7

Implementations MUST support "aes-128-gcm-sha256". Implementations SHOULD support "aes-128-ctr-hmac-sha256-80" for scenarios requiring smaller authentication tags.

5.2.40. Key ID

Required: Optional JSON Type: String Location: Track Object

A string identifying the key material used for encryption. This value is transmitted in the Secure Object KID extension header as defined in ([SecureObjects], Section 4.2) of each encrypted object.

The keyId and associated trackBaseKey are obtained from an external key management system. The mechanism for obtaining these values is out of scope for this specification. Examples include MLS-based key distribution [E2EE-MLS] or other out-of-band key exchange mechanisms.

The scope of a keyId is determined by the key management system in use. A keyId MAY be scoped to a single track, a single MSF session, or multiple tracks and sessions. When multiple tracks share the same Key ID, they MAY share the same base key material, though per-track keys are derived using the track name as defined in ([SecureObjects], Section 5).

5.2.41. Track Base Key

Required: Optional JSON Type: String Location: Track Object

A base64-encoded [BASE64] string containing the base key material for this track, as defined in ([SecureObjects], Section 5). This field works in conjunction with keyId to provide the cryptographic material needed for decryption. The trackBaseKey is obtained from the same key management system that provides the keyId.

When present, this field contains the raw key material that, together with the track name and other parameters defined in ([SecureObjects], Section 5), is used to derive the actual encryption keys. Publishers and subscribers MUST use matching trackBaseKey values for successful decryption.

5.2.42. Authorization Info

Required: Optional JSON Type: Object Location: Track Object

An object indicating that authorization is required to access this track. The presence of this field signals to subscribers that they must obtain and present valid authorization tokens when subscribing to this track.

The keys of this object are authorization scheme identifiers. Registered schemes are defined in Table 7. The values are scheme-specific configuration objects defined by the referenced specifications.

Table 7: Registered Authorization Schemes

Scheme	Value	Reference
Privacy Pass	privacy-pass	[PrivacyPassAuth]
CAT	cat	[C4M]

Table 8

Custom authorization schemes MAY be used. Custom scheme names MUST use a unique naming convention, such as Reverse Domain Name Notation (e.g., "com.example.custom-auth"), to avoid naming collisions.

Subscribers inspect the authInfo field to determine which authorization scheme to use and then obtain tokens through out-of-band mechanisms. The specific token format, acquisition process, and presentation method are defined by the authorization scheme specification.

5.2.43. Token Delivery via URI

Tokens can be delivered to subscribers through the catalog request URI using variable substitution. Fragment parameters (following the # character) are processed client-side and can be substituted into catalog fields using the variable substitution mechanism defined in Section 5.4.

For example, given a URI:

moqt://relay.example.com/live#namespace--name&token=XYZ789

A catalog with:

```
"authInfo": {
  "cat": "%token%"
}
```

Would be resolved by the subscriber to include "cat": "XYZ789", which is then presented in control messages as specified by the authorization scheme.

5.2.44. Accessibility

Required: Optional JSON Type: Array Location: Track Object

An array of accessibility descriptors indicating accessibility features embedded within the track. Each descriptor is a JSON Object containing:

* A required ’scheme’ field (String) identifying the accessibility scheme.

* A required ’value’ field (String) specifying the accessibility channels or features available.

Table 6: Registered accessibility schemes

Scheme	Description
urn:scte:dash:cc:cea-608:2015	CEA-608 closed captions
urn:scte:dash:cc:cea-708:2015	CEA-708 closed captions

Table 9

The ’value’ field for CEA-608/708 schemes uses the format defined by [SCTE214-1], where caption service channels are specified as semicolon-separated pairs of channel identifier and language code (e.g., "CC1=eng;CC3=spa").

A subscriber MAY use this information to determine caption availability and configure an appropriate caption decoder.

5.3. Delta updates

A catalog update might contain incremental changes. This is a useful property if many tracks may be initially declared but then there are small changes to a subset of tracks. The producer can issue a delta update to describe these changes. Changes are described incrementally, meaning that a delta update can itself modify a prior delta update.

A restricted set of operations are allowed with each delta update: * Add a new track that has not previously been declared. * Add a new track by cloning a previously declared track. * Remove a track that has been previously declared.

The following rules are to be followed in constructing and processing delta updates:

* A delta update MUST include the Delta Update Section 5.1.6 field with at least one operation. It MUST NOT contain an instance of a Tracks Section 5.1.4 field or an MSF version Section 5.1.1 field.

* Operations are applied sequentially in the order they appear in the deltaUpdate array. Each operation is applied to the target document; the resulting document becomes the target of the next operation. Evaluation continues until all operations are successfully applied.

* The tuple of Track Namespace and Track Name defines a fixed set of Track attributes which MUST NOT be modified after being declared. To modify any attribute, a new track with a different Namespace|Name tuple is created by Adding or Cloning and then the old track is removed.

* Producers that publish frequent delta updates SHOULD periodically publish a new independent catalog in a new MOQT Group in order to bound the amount of delta processing required for joining subscribers.

5.4. Variable Substitution

Catalog field values MAY contain variables that are substituted at delivery time. This mechanism enables a single cached catalog to be customized for each viewer, supporting use cases such as personalized advertising, A/B watermarking, QoE reporting endpoints, and logging identifiers.

5.4.1. Variable Syntax

Variables are denoted by enclosing the variable name in percent characters (%). Variable names MUST consist of alphanumeric characters, hyphens, and underscores. Variable names are case-sensitive.

The percent character (%) MUST NOT appear in catalog field values except as part of a variable reference. Literal percent characters are not permitted.

Variable values MUST consist only of alphanumeric characters, hyphens, underscores, and the at sign (@). Special characters including commas, semicolons, quotes, ampersands, and other punctuation MUST NOT appear in variable values. This restriction prevents injection attacks and ensures safe substitution into catalog field values.

5.4.2. Variable Resolution

Variables are resolved from the fragment identifier of the URI used to access the catalog. The fragment identifier is the portion following the # character and is processed entirely client-side, making it suitable for per-viewer customization without affecting server-side caching.

Query parameters (following the ? character) are reserved for server-side processing and MUST NOT be used for variable substitution.

When a subscriber requests a catalog using a URI containing a fragment identifier, the fragment is parsed as key-value pairs (using & as delimiter and = as separator). Each key becomes available as a variable name, and the variable is replaced with the corresponding value.

5.5. Catalog Compression

Catalogs can contain significant redundancy, particularly when initialization data is included. To reduce payload size, catalog objects MAY be compressed using the MSF_COMPRESSION property (Section 12.1). If all catalog objects are compressed, the track property (Section 12.1.1) is used. If only some catalog objects are compressed (for example, the complete catalog is compressed while delta updates are not), the object property (Section 12.1.2) is used on each compressed object.

5.6. Catalog Examples

The following section provides non-normative JSON examples of various catalogs compliant with this draft.

5.6.1. Time-aligned Audio/Video Tracks with single quality

This example shows a catalog for a media producer capable of sending LOC packaged, time-aligned audio and video tracks.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "1080p-video",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "video",
      "renderGroup": 1,
      "codec":"av01.0.08M.10.0.110.09",
      "width":1920,
      "height":1080,
      "framerate":30,
      "bitrate":1500000
    },
    {
      "name": "audio",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "audio",
      "renderGroup": 1,
      "codec":"opus",
      "samplerate":48000,
      "channelConfig":"2",
      "bitrate":32000
    }
  ]
}
```

5.6.2. Simulcast video tracks - 3 alternate qualities along with audio

This example shows a catalog for a media producer capable of sending 3 time-aligned video tracks for high definition, low definition and medium definition video qualities, along with an audio track. In this example the namespace is absent, which infers that each track must inherit the namespace of the catalog.

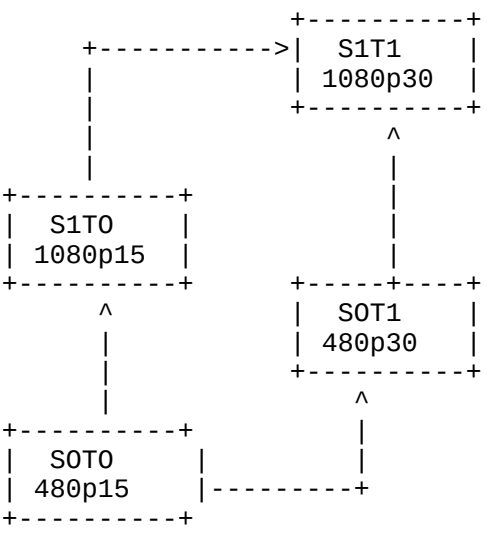
```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks":[
    {
      "name": "hd",
```

```
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "targetLatency": 1500,
        "role": "video",
        "codec":"av01",
        "width":1920,
        "height":1080,
        "bitrate":5000000,
        "framerate":30,
        "altGroup":1
      },
      {
        "name": "md",
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "targetLatency": 1500,
        "role": "video",
        "codec":"av01",
        "width":720,
        "height":640,
        "bitrate":3000000,
        "framerate":30,
        "altGroup":1
      },
      {
        "name": "sd",
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "targetLatency": 1500,
        "role": "video",
        "codec":"av01",
        "width":192,
        "height":144,
        "bitrate":500000,
        "framerate":30,
        "altGroup":1
      },
      {
        "name": "audio",
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "targetLatency": 1500,
        "role": "audio",
        "codec":"opus",
```

```
        "samplerate":48000,
        "channelConfig":"2",
        "bitrate":32000
      }
    ]
  }
```

5.6.3. SVC video tracks with 2 spatial and 2 temporal qualities

This example shows a catalog for a media producer capable of sending scalable video codec with 2 spatial and 2 temporal layers with a dependency relation as shown below:



The corresponding catalog uses "depends" attribute to express the track relationships.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks":[
    {
      "name": "480p15",
      "namespace": "conference.example.com/conference123/alice",
      "renderGroup": 1,
      "packaging": "loc",
      "isLive": true,
      "buffers": {"target":2000},
      "role": "video",
      "codec":"av01.0.01M.10.0.110.09",
```

```
        "width":640,
        "height":480,
        "bitrate":3000000,
        "framerate":15
      },
      {
        "name": "480p30",
        "namespace": "conference.example.com/conference123/alice",
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "buffers": {"target":2000},
        "role": "video",
        "codec":"av01.0.04M.10.0.110.09",
        "width":640,
        "height":480,
        "bitrate":3000000,
        "framerate":30,
        "depends": ["480p15"]
      },
      {
        "name": "1080p15",
        "namespace": "conference.example.com/conference123/alice",
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "buffers": {"target":2000},
        "role": "video",
        "codec":"av01.0.05M.10.0.110.09",
        "width":1920,
        "height":1080,
        "bitrate":3000000,
        "framerate":15,
        "depends":["480p15"]
      },

      {
        "name": "1080p30",
        "namespace": "conference.example.com/conference123/alice",
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "buffers": {"target":2000},
        "role": "video",
        "codec":"av01.0.08M.10.0.110.09",
        "width":1920,
        "height":1080,
        "bitrate":5000000,
```

```
        "framerate":30,
        "depends": ["480p30", "1080p15"]
      },
      {
        "name": "audio",
        "namespace": "conference.example.com/conference123/alice",
        "renderGroup": 1,
        "packaging": "loc",
        "isLive": true,
        "buffers": {"target":2000},
        "role": "audio",
        "codec":"opus",
        "samplerate":48000,
        "channelConfig":"2",
        "bitrate":32000
      }
    ]
  }
```

5.6.4. Delta update - adding two tracks

This example shows the catalog delta update for a media producer adding two tracks to an established video conference. One track is newly declared, the other is cloned from a previous track.

```
{
  "generatedAt": 1746104606044,
  "deltaUpdate": [
    {
      "op": "add",
      "tracks": [
        {
          "name": "slides",
          "isLive": true,
          "role": "video",
          "codec": "av01.0.08M.10.0.110.09",
          "width": 1920,
          "height": 1080,
          "framerate": 15,
          "bitrate": 750000,
          "renderGroup": 1
        }
      ]
    },
    {
      "op": "clone",
      "tracks": [
        {
          "parentName": "video-1080",
          "parentNamespace": "example.com/custom",
          "name": "video-720",
          "width": 1280,
          "height": 720,
          "bitrate": 600000
        }
      ]
    }
  ]
}
```

5.6.5. Delta update removing tracks

This example shows a delta update for a media producer removing two tracks from an established video conference.

```
{
  "generatedAt": 1746104606044,
  "deltaUpdate": [
    {
      "op": "remove",
      "tracks": [{"name": "video"}, {"name": "slides"}]
    }
  ]
}
```

5.6.6. Time-aligned Audio/Video Tracks with custom field values

This example shows a catalog for a media producer capable of sending LOC packaged, time-aligned audio and video tracks along with custom fields in each track description.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "1080p-video",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "buffers": {"target":2000, "min": 1500, "max": 5000},
      "role": "video",
      "renderGroup": 1,
      "codec":"av01.0.08M.10.0.110.09",
      "width":1920,
      "height":1080,
      "framerate":30,
      "bitrate":1500000,
      "com.example-billing-code": 3201,
      "com.example-tier": "premium",
      "com.example-debug": "h349835bfkjfg82394d945034jsdfn349fns"
    },
    {
      "name": "audio",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "buffers": {"target":2000, "min": 1500, "max": 5000},
      "role": "audio",
      "renderGroup": 1,
      "codec":"opus",
      "samplerate":48000,
      "channelConfig":"2",
      "bitrate":32000
    }
  ]
}
```

5.6.7. Time-aligned VOD Audio/Video Tracks

This example shows a catalog for a media producer offering VOD (video on-demand) non-live content. The content is LOC packaged, and includes time-aligned audio and video tracks.

```
{
  "version": "1",
  "tracks": [
    {
      "name": "video",
      "namespace": "movies.example.com/assets/boy-meets-girl-season3/episode5",
      "packaging": "loc",
      "isLive": false,
      "trackDuration": 8072340,
      "renderGroup": 1,
      "codec":"av01.0.08M.10.0.110.09",
      "width":1920,
      "height":1080,
      "framerate":30,
      "bitrate":1500000
    },
    {
      "name": "audio",
      "namespace": "movies.example.com/assets/boy-meets-girl-season3/episode5",
      "packaging": "loc",
      "isLive": false,
      "trackDuration": 8072340,
      "renderGroup": 1,
      "codec":"opus",
      "samplerate":48000,
      "channelConfig":"2",
      "bitrate":32000
    }
  ]
}
```

5.6.8. Encrypted Audio/Video Tracks

This example shows a catalog for encrypted LOC-packaged audio and video tracks using MoQ Secure Objects with AES-128-GCM.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "1080p-video",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "video",
      "renderGroup": 1,
      "codec": "av01.0.08M.10.0.110.09",
      "width": 1920,
      "height": 1080,
      "framerate": 30,
      "bitrate": 1500000,
      "encryptionScheme": "moq-secure-objects",
      "cipherSuite": "aes-128-gcm-sha256",
      "keyId": "key-2024-q1-premium",
      "trackBaseKey": "dGhpc2lzYXNhbXBsZWJhc2VrZXk="
    },
    {
      "name": "audio",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "audio",
      "renderGroup": 1,
      "codec": "opus",
      "samplerate": 48000,
      "channelConfig": "2",
      "bitrate": 32000,
      "encryptionScheme": "moq-secure-objects",
      "cipherSuite": "aes-128-gcm-sha256",
      "keyId": "key-2024-q1-premium",
      "trackBaseKey": "dGhpc2lzYXNhbXBsZWJhc2VrZXk="
    }
  ]
}
```

5.6.9. Media timeline and Event timeline

This example shows a catalog for a media producer capable of sending LOC packaged, time-aligned audio and video tracks, along with a Media Timeline which describes the history of those tracks and an Event Timeline providing synchronized data.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "history",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "mediatimeline",
      "mimetype": "application/json",
      "depends": ["1080p-video","audio"]
    },
    {
      "name": "identified-objects",
      "namespace": "another-provider/time-synchronized-data",
      "packaging": "eventtimeline",
      "eventType": "com.ai-extraction/appID/v3",
      "mimetype": "application/json",
      "depends": ["1080p-video"]
    },
    {
      "name": "1080p-video",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "video",
      "renderGroup": 1,
      "codec":"av01.0.08M.10.0.110.09",
      "width":1920,
      "height":1080,
      "framerate":30,
      "bitrate":1500000
    },
    {
      "name": "audio",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "audio",
      "renderGroup": 1,
      "codec":"opus",
      "samplerate":48000,
      "channelConfig":"2",
      "bitrate":32000
    }
  ]
}
```

5.6.10. Media timeline template

This example shows a catalog using inline media timeline templates instead of an explicit media timeline track. The Section 5.2.15 attribute on each track defines a regular pattern where each segment is 2002ms long. Clients can compute any entry using the template parameters. Note that different tracks MAY have different template values to accommodate different group durations.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "1080p-video",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "video",
      "renderGroup": 1,
      "template": [0, 2002, [0, 0], [1, 0], 1759924158381, 2002],
      "codec":"av01.0.08M.10.0.110.09",
      "width":1920,
      "height":1080,
      "framerate":30,
      "bitrate":1500000
    },
    {
      "name": "audio",
      "namespace": "conference.example.com/conference123/alice",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "audio",
      "renderGroup": 1,
      "template": [0, 2002, [0, 0], [1, 0], 1759924158381, 2002],
      "codec":"opus",
      "samplerate":48000,
      "channelConfig":"2",
      "bitrate":32000
    }
  ]
}
```

5.6.11. Video track with embedded captions and SCTE-35 events

This example shows a live broadcast with CEA-608 closed captions embedded in the video track and a separate SCTE-35 event timeline for ad insertion.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "video",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 4000,
      "role": "video",
      "renderGroup": 1,
      "codec": "avc1.4d401f",
      "width": 1920,
      "height": 1080,
      "framerate": 30,
      "bitrate": 5000000,
      "accessibility": [
        {
          "scheme": "urn:scte:dash:cc:cea-608:2015",
          "value": "CC1=eng;CC3=spa"
        }
      ]
    },
    {
      "name": "audio",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 4000,
      "role": "audio",
      "renderGroup": 1,
      "codec": "opus",
      "samplerate": 48000,
      "channelConfig": "2",
      "bitrate": 128000
    },
    {
      "name": "scte35",
      "packaging": "eventtimeline",
      "eventType": "urn:scte:scte35:2013:bin",
      "mimeType": "application/json",
      "isLive": true,
      "role": "eventtimeline",
      "depends": ["video"]
    }
  ]
}
```

5.6.12. Video track with CEA-708 captions

This example shows a live broadcast with CEA-708 closed captions embedded in the video track, demonstrating multiple caption services.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "video",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 4000,
      "role": "video",
      "renderGroup": 1,
      "codec": "hev1.1.6.L93.B0",
      "width": 1920,
      "height": 1080,
      "framerate": 30,
      "bitrate": 4000000,
      "accessibility": [
        {
          "scheme": "urn:scte:dash:cc:cea-708:2015",
          "value": "1=lang:eng;2=lang:spa;3=lang:fra"
        }
      ]
    },
    {
      "name": "audio",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 4000,
      "role": "audio",
      "renderGroup": 1,
      "codec": "mp4a.40.2",
      "samplerate": 48000,
      "channelConfig": "2",
      "bitrate": 128000
    }
  ]
}
```

5.6.13. Terminating a live broadcast

This example shows a catalog for a media producer terminating a previously live broadcast containing a video and an audio track.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "isComplete": true,
  "tracks": []
}
```

5.6.14. Variable Substitution for personalized delivery

This example shows a catalog using variable substitution to enable personalized advertising and reporting while maintaining cacheability. Given a catalog request URI of:

```
moqt://relay.example.com/sports/catalog?a=1#token=1234&id=bob&event=xyz
```

The fragment parameters (token, id, event) are available for variable substitution. The following catalog template:

```
{
  "version": "1",
  "tracks": [
    {
      "name": "video",
      "packaging": "loc",
      "isLive": true,
      "role": "video",
      "renderGroup": 1
    },
    {
      "name": "cmcdv2-%id%",
      "namespace": "advertising-decisions/live-sports/%event%",
      "packaging": "eventtimeline",
      "eventType": "com.example.iab.vast",
      "c4m": "%token%"
    }
  ]
}
```

Would be resolved by the subscriber as:

```
{
  "version": "1",
  "tracks": [
    {
      "name": "video",
      "packaging": "loc",
      "isLive": true,
      "role": "video",
      "renderGroup": 1
    },
    {
      "name": "cmcdv2-bob",
      "namespace": "advertising-decisions/live-sports/xyz",
      "packaging": "eventtimeline",
      "eventType": "com.example.iab.vast",
      "c4m": "1234"
    }
  ]
}
```

5.6.15. Time-aligned Audio/Video Tracks with Authorization

This example shows a catalog for a media producer requiring authorization for premium content. The premium 4K track uses CAT authorization with a token resolved via variable substitution from %cat-token%. The standard 720p track uses Privacy Pass with a token resolved from %pp-token%. Token presentation timing is determined by the authorization scheme specification.

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "premium-4k-video",
      "namespace": "streaming.example.com/live/sports",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "video",
      "renderGroup": 1,
      "codec": "av01.0.12M.10.0.110.09",
      "width": 3840,
      "height": 2160,
      "framerate": 60,
      "bitrate": 15000000,
      "authInfo": {
        "cat": "%cat-token%"
```

```
        }
      },
      {
        "name": "standard-720p-video",
        "namespace": "streaming.example.com/live/sports",
        "packaging": "loc",
        "isLive": true,
        "targetLatency": 2000,
        "role": "video",
        "renderGroup": 2,
        "codec": "av01.0.05M.10.0.110.09",
        "width": 1280,
        "height": 720,
        "framerate": 30,
        "bitrate": 2500000,
        "authInfo": {
          "privacy-pass": "%pp-token%"
        }
      },
      {
        "name": "audio",
        "namespace": "streaming.example.com/live/sports",
        "packaging": "loc",
        "isLive": true,
        "targetLatency": 2000,
        "role": "audio",
        "renderGroup": 1,
        "codec": "opus",
        "samplerate": 48000,
        "channelConfig": "2",
        "bitrate": 128000
      }
    ]
  }
```

5.6.16. Publish tracks for logs and metrics

This example shows a catalog that includes publish tracks for client-side logging and metrics collection. The subscriber can publish QoE data and logs back to the delivery system using these track definitions. The namespace and track name formats follow the conventions defined in [MOQLOG] and [MOQMETRICS].

```
{
  "version": "1",
  "generatedAt": 1746104606044,
  "tracks": [
    {
      "name": "video",
      "namespace": "broadcast.example.com/live/stream1",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "video",
      "renderGroup": 1,
      "codec": "av01.0.08M.10.0.110.09",
      "width": 1920,
      "height": 1080,
      "framerate": 30,
      "bitrate": 1500000
    },
    {
      "name": "audio",
      "namespace": "broadcast.example.com/live/stream1",
      "packaging": "loc",
      "isLive": true,
      "targetLatency": 2000,
      "role": "audio",
      "renderGroup": 1,
      "codec": "opus",
      "samplerate": 48000,
      "channelConfig": "2",
      "bitrate": 32000
    }
  ],
  "publishTracks": [
    {
      "namespace": "moq://metrics.moq.arpa/v1/%resourceId%",
      "name": "4",
      "packaging": "moqmetrics",
      "role": "metrics",
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
    },
    {
      "namespace": "moq://moq-syslog.arpa/logs-v1/%resourceId%",
      "name": "6",
      "packaging": "moqlog",
      "role": "log",
      "connectionUri": "moqt://logs.example.com:4443",
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
    }
```

```
  ]
}
```

In this example:

* The metrics track uses the namespace format defined in [MOQMETRICS] with %resourceId% as a placeholder for the subscriber's unique resource identifier. The track name "4" indicates Warning level granularity.

* The log track uses the namespace format defined in [MOQLOG] with %resourceId% as a placeholder. The track name "6" indicates Informational level priority. This track establishes a separate connection to logs.example.com.

* Both tracks include authentication tokens for publishing authorization.

* The subscriber MUST replace %resourceId% with their actual resource identifier as defined in the respective specifications.

6. Media transmission

The MOQT Groups and MOQT Objects need to be mapped to MOQT Streams. Irrespective of the Section 4 in place, each MOQT Object MUST be mapped to a new MOQT Stream.

6.1. Group numbering

Group IDs for a track MUST be unique and MUST increase monotonically. Within a continuous publishing session, each subsequent Group ID SHOULD increase by 1.

When a publisher restarts (e.g., after connectivity loss or encoder restart), it MUST ensure the new starting Group ID is greater than any previously published Group ID for that track. One approach is to use the current wall clock time in milliseconds since the Unix epoch as the starting Group ID.

If a publisher maintains state across a restart and knows the previous Group ID, it SHOULD signal the gap using the MOQT Prior Group ID Gap Extension header.

6.2. Object Numbering

Object ID MUST be zero for the first Object within a Group and then MUST increase monotonically by one within that Group.

7. Media Timeline track

The media timeline track provides data about the previously published Groups and their relationship to wallclock time and media time. Media timeline tracks allow players to seek to precise points behind the live head in a live broadcast, or for random access in a VOD asset. Media timeline tracks are optional. Multiple media timeline tracks can exist inside a catalog.

7.1. Media Timeline track payload

A media timeline track is a JSON [JSON] document. This document MAY be compressed using the MSF_COMPRESSION property (Section 12.1). The document supports two formats: an explicit entry format and a template format. Publishers MAY combine both formats in a single document.

7.1.1. Explicit entry format

The explicit format contains an array of records. Each record consists of an array of three required items, whose ordinal position defines their type:

* The first item holds the media presentation timestamp, expressed as a JSON Number. This value MUST match the media presentation timestamp, expressed as the floor in integral milliseconds, of the first media sample in the referenced Object. Implementers who require increased time precision can parse the retrieved media object itself.

* The second item holds the MOQT Location of the entry, defined as a tuple of the MOQT Group ID and MOQT Object ID, and expressed as a JSON Array of Numbers, where the first number is the Group ID and the second number is the Object ID.

* The third item holds the wallclock time at which the media was encoded, defined as the number of milliseconds that have elapsed since January 1, 1970 (midnight UTC/GMT) and expressed as a JSON Number. For VOD assets, or if the wallclock time is not known, the value SHOULD be 0.

An example media timeline is shown below:

```
[
  [0, [0,0], 1759924158381],
  [2002, [1,0], 1759924160383],
  [4004, [2,0], 1759924162385],
  [6006, [3,0], 1759924164387],
  [8008, [4,0], 1759924166389]
]
```

7.2. Media Timeline Catalog requirements

A media timeline track MUST carry a 'type' identifier in the Catalog with a value of "mediatimeline". A media timeline track MUST carry a 'depends' attribute which contains an array of all track names to which the media timeline track applies. The mime-type of a media timeline track MUST be specified as "application/json".

7.3. Media Timeline track updating

The publisher MUST publish an independent media timeline in the first MOQT Object of each MOQT Group of a media timeline track. An independent media timeline object MUST contain all media timeline records accumulated and accessible up to that point, allowing a subscriber joining at any group boundary to receive the accessible timeline history. The publisher MAY publish incremental updates in the second and subsequent Objects within each Group. Incremental updates contain only new media timeline records since the previous media timeline Object in that Group.

7.4. Media Timeline Template

When the relationship between media time, MOQT Location and wallclock time follows a regular, predictable pattern, a media timeline template MAY be used instead of an explicit media timeline track. The template approach is best suited for content with fixed-duration segments, such as VOD assets or live broadcasts with constant segment durations. For content with variable segment durations, the explicit media timeline track (Section 7.1) MUST be used instead.

7.4.1. Template Format

A media timeline template is expressed as an inline track attribute using the Section 5.2.15 field. The template is a JSON Array containing six mandatory values in the following fixed order:

1. startMediaTime - The media presentation timestamp of the first entry, as defined for media timeline entries in Section 7.1.1.

2. deltaMediaTime - The constant interval between media presentation timestamps, expressed as a JSON Number in milliseconds.

3. startLocation - The MOQT Location of the first entry, as defined for media timeline entries in Section 7.1.1.

4. deltaLocation - The constant interval between MOQT Locations, expressed as a JSON Array of two Numbers where the first number is the Group ID delta and the second number is the Object ID delta.

5. startWallclock - The wallclock time of the first entry, as defined for media timeline entries in Section 7.1.1. For VOD assets, or if the wallclock time is not known, the value SHOULD be 0.

6. deltaWallclock - The constant interval between wallclock times, expressed as a JSON Number in milliseconds. For VOD assets, or if the wallclock time is not known, the value SHOULD be 0.

All six values are mandatory and MUST appear in the specified order.

Clients compute entry values using the following formulas, where n is the zero-based entry index:

```
mediaTime[n] = startMediaTime + (n * deltaMediaTime)
location[n] = [startLocation[0] + (n * deltaLocation[0]),
               startLocation[1] + (n * deltaLocation[1])]
wallclock[n] = startWallclock + (n * deltaWallclock)
```

An example template value is shown below:

```
[0, 2002, [0, 0], [1, 0], 1759924158381, 2002]
```

This template indicates that the first entry has media time 0ms, location [0,0], and wallclock time 1759924158381. Each subsequent entry increments media time by 2002ms, location by [1,0] (next group, same object), and wallclock by 2002ms.

7.4.2. Template Immutability

Unlike the explicit media timeline track, the media timeline template is intended to be immutable once publishing starts. Publishers MUST NOT change the template values for a track after the first Object has been published.

8. Event Timeline track

The event timeline track provides a mechanism to associate ad-hoc event metadata with the broadcast. Use-case examples include live sports score data, GPS coordinates of race cars, SAP-types for media segments or active speaker notifications in web conferences.

To allow the client to bind this event metadata with the broadcast content described by the media timeline track, each event record MUST contain a reference to one of Media PTS, wallclock time or MOQT Location.

Event timeline tracks are optional. Multiple event timeline tracks can exist inside a catalog. The type & structure of the data contained within each event timeline track is declared in the catalog, to facilitate client selection and parsing.

8.1. Event Timeline data format

An event timeline track is a JSON [JSON] document. This document MAY be compressed using the MSF_COMPRESSION property (Section 12.1). The document contains an array of records. Each record consists of a JSON Object containing the following required fields:

* An index reference, which MUST be either ’t’ for wallclock time, ’l’ for Location or ’m’ for Media PTS. Only one of these index values may be used within each record. Event timelines SHOULD use the same index reference type for each record. The definitions for wallclock time, Location and Media PTS are identical to those defined for media timeline payload Section 7.1. Wallclock time and media PTS values are JSON Number, while Location value is an Array of Numbers, where the first item represents the MOQT GroupID and the second item the MOQT Object ID.

* A ’data’ Object, whose structure is defined by the Section 5.2.5 value declared for this track in the Catalog.

8.2. Event Timeline Catalog requirements

An event timeline track MUST carry:

* a Section 5.2.4 attribute with a value of "eventtimeline".

* a Section 5.2.14 attribute which contains an array of all track names to which the event timeline track applies.

* a Section 5.2.19 attribute with a value of "application/json".

* an Section 5.2.5 attribute declaring the type & structure of data contained in the event timeline track.

8.3. Event Timeline track updating

The publisher MUST publish an independent event timeline in the first MOQT Object of each MOQT Group of an event timeline track. An independent event timeline object MUST contain all event timeline records accumulated and accessible up to that point, allowing a subscriber joining at any group boundary to receive the accessible event history. The publisher MAY publish incremental updates in the second and subsequent Objects within each Group. Incremental updates contain only new event timeline records since the previous event timeline Object in that Group.

8.4. Event timeline track examples

8.4.1. Event timeline track with wallclock time indexing

This example shows how sports scores and game information might be defined in a live sports broadcast.

```
[
    {
        "t": 1756885678361,
        "data": {
            "status": "in_progress",
            "period": 1,
            "clock": "12:00",
            "homeScore": 0,
            "awayScore": 0,
            "lastPlay": "Game Start"
        }
    },
    {
        "t": 1756885981542,
        "data": {
            "status": "in_progress",
            "period": 1,
            "clock": "09:25",
            "homeScore": 2,
            "awayScore": 0,
            "lastPlay": "Team A: #23 makes 2-pt jump shot"
        }
    }
]
```

8.4.2. Event timeline track with MOQT Location indexing

This example shows drone GPS coordinates synched with the start of each Group.

```
[
    {
        "l": [0,0],
        "data": [47.1812,8.4592]
    },
    {
        "l": [1,0],
        "data": [47.1662,8.5155]
    }
]
```

9. Log track

Log tracks provide a mechanism for subscribers to publish diagnostic and operational log data back to the delivery system. This enables QoE monitoring, debugging, and analytics collection. Log tracks are defined in the catalog's publishTracks array with a packaging value of "moqlog".

9.1. Log track payload

The log track payload format is defined in [MOQLOG]. Each MOQT Object contains a single JSON-formatted log entry. The log entry structure includes optional fields such as severity, timestamp, hostname, application name, process ID, message ID, and the log message itself. Additional structured data fields support distributed tracing integration with TraceID, SpanID, and InstrumentationScope aligned with OpenTelemetry specifications. Implementations MUST follow the object payload format specified in Section 4 of [MOQLOG].

9.2. Log track namespace and name

TODO: Finalize on track naming

Log tracks MUST use the namespace and track name format defined in Section 3 of [MOQLOG]. The Track Namespace consists of the tuples: (moq://moq-syslog.arpa/logs-v1/),(resourceID) where resourceID is a unique identifier for the publishing resource.

The Track Name is a single byte containing the log priority level in binary. Priority levels range from 0 (Emergency) to 7 (Debug), following syslog severity conventions.

9.3. Log track Group ID and Object ID

The Group ID represents the timestamp at which the log entry was captured, truncated to a 62-bit binary integer representing microseconds since the Unix epoch. This allows log entries to be naturally ordered by time within the MOQT object model.

The Object ID is set to zero for a log entry unless multiple log messages occur within the same microsecond, in which case the Object ID increments to distinguish between messages captured at the same timestamp.

9.4. Log track catalog requirements

A log track MUST be declared in the publishTracks array of the catalog with:

* a Section 5.2.4 attribute with a value of "moqlog".

* a Section 5.2.6 attribute with a value of "log".

A log track MAY include:

* a Section 5.2.36 attribute if logs should be published to a different endpoint.

* a Section 5.2.37 attribute for publish authorization.

10. Metrics track

Metrics tracks provide a mechanism for subscribers to publish quantitative measurements back to the delivery system. This enables QoE analytics, performance monitoring, and operational dashboards. Metrics tracks are defined in the catalog’s publishTracks array with a packaging value of "moqmetrics".

10.1. Metrics track payload

The metrics track payload format is defined in [MOQMETRICS]. The data model consists of Resources (systems generating metrics identified by ResourceID), optional Attributes (dimensional key-value pairs), and Metrics (named measurements with timeseries values).

[MOQMETRICS] defines two metric value types - Gauge and Counter. Both value types support either 64-bit floating point or integer representation. Implementations MUST follow the object payload format specified in Section 3 of [MOQMETRICS].

10.2. Metrics track namespace and name

TODO: Finalize the track naming.

Metrics tracks MUST use the namespace and track name format defined in Section 3 of [MOQMETRICS]. The Track Namespace consists of the tuples: (moq://metrics.moq.arpa/v1/),(resourceID) where resourceID is a unique identifier for the publishing resource.

The Track Name identifies the granularity level for the metrics being published, specified as a single tuple (<granularity-level>) where granularity-level is a value from 0 (Emergency) to 7 (Debug). Higher priority levels (lower numbers) indicate more critical metrics that should always be reported.

10.3. Metrics track Group ID and Object ID

The Group ID represents the capture time as the number of milliseconds since January 1, 1970 (Unix epoch). This allows metrics to be naturally grouped and ordered by their capture timestamp within the MOQT object model.

Within each Group, Object IDs are assigned as follows:

* *Object ID 0*: Contains the capture timestamp as Unix epoch time in nanoseconds, along with any optional attributes for the metrics in this group.

* *Object ID 1 and above*: Each subsequent Object contains an individual metric as a name-value pair.

This structure allows a single capture event to include multiple metrics while maintaining efficient MOQT caching and distribution.

10.4. Metrics track catalog requirements

A metrics track MUST be declared in the publishTracks array of the catalog with:

* a Section 5.2.4 attribute with a value of "moqmetrics".

* a Section 5.2.6 attribute with a value of "metrics".

A metrics track MAY include:

* a Section 5.2.36 attribute if metrics should be published to a different endpoint.

* a Section 5.2.37 attribute for publish authorization.

10.5. Well-known event timeline types

Event timelines can carry various types of broadcast metadata synchronized with media content. The "MSF Event Timeline Types" registry (Section 14.2) maintains a list of well-known event types. Publishers SHOULD use registered types when applicable to ensure interoperability.

Event timelines can carry data types including but not limited to:

* Ad insertion signaling (e.g., SCTE-35 splice points) - see [SCTE35-MSF]

* Out-of-band timed-text cues (WebVTT, IMSC1) - see [WebVTT-MSF] and [IMSC1-MSF]

* Sports scores and game state

* GPS coordinates and telemetry

* Active speaker notifications

* Custom application-specific metadata

The packaging format and data structure for each event type is defined by the specification referenced in the registry. Custom event types not in the registry SHOULD use Reverse Domain Name Notation (e.g., "com.example.myeventtype") to avoid naming collisions.

11. Workflow

11.1. URL construction and interpretation

An MSF URL identifies a MOQT session and an optional sub-resource within that session. It inherits the MOQT URI scheme defined in [MoQTransport] and extends it to add a fragment definition, which defines the type, the namespace and name of the track along with optional key-value attributes.

```
; MSF URI Definition
; The following rules are imported from RFC 3986:
;   authority    (Section 3.2)
;   path-abempty (Section 3.3)
;   query        (Section 3.4)

msf-uri = "moqt://" authority path-abempty [ "?" query ] "#" msf-fragment
```

The MOQT specification carries the normative definition of these components, along with processing instructions. They are repeated here for clarity. In the event of any conflict, the definition in [MoQTransport] is authoritative.

* Scheme: This case-insensitive scheme defines the underlying transport. The client may use either a WebTransport or native QUIC connection. A moqt URI can be converted to an https URI by replacing the scheme, so the path-abempty and query components use the same syntax as https URIs.

* Authority: Required. Contains the host and optional port (defaulting to 443). This information is used by the client to establish the transport session.

* Path: Optional. If present, it provides server-specific configuration or routing information used during connection initialization.

* Query: Optional. Contains key-value parameters separated by &. If the query is absent, the ? separator MUST be omitted. Query arguments are intended for the server and SHOULD be ignored by the client.

This specification registers a fragment type of "msf" in the "MOQT URI Fragment Types" registry established by [MoQTransport] (see Section 14). This type is required for any URL defining an MSF resource. The value of this type is the msf-fragment-value.

The msf-fragment is defined by the following ABNF:

```
; MSF Fragment Definition
; The following rules are imported from RFC 3986:
;   pchar        (Section 3.3)
;   unreserved   (Section 2.3)
;   pct-encoded  (Section 2.1)
;   sub-delims   (Section 2.2)

msf-fragment      = "msf:" msf-fragment-value

msf-fragment-value = track-identifier [ "&" parameter-list ]

track-identifier  = 1*( pchar-no-amp / "/" )
                    ; MSF namespace-name string
                    ; MUST NOT contain '&' or '?'
                    ; '?' MUST be percent-encoded as %3F within this component

parameter-list    = parameter *( "&" parameter )

parameter         = param-name "=" param-value

param-name        = 1*( pchar-no-amp / "/" )

param-value       = *( pchar-no-amp / "/" )

; Derived from RFC 3986 pchar (Section 3.3), excluding '&' and '?':
;   pchar = unreserved / pct-encoded / sub-delims / ":" / "@"
; '&' is excluded as it serves as the MSF fragment component delimiter.
; '?' is excluded to avoid ambiguity with the URI query component delimiter.
pchar-no-amp      = unreserved / pct-encoded / sub-delims-no-amp / ":" / "@"

; Derived from RFC 3986 sub-delims (Section 2.2), excluding '&':
;   sub-delims = "!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "="
sub-delims-no-amp = "!" / "$" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "="
```

The msf-fragment-value identifies a specific Track. The track-identifier MUST be formatted as an MSF namespace-name string (see Section 11.1.2). The client uses this identifier to initiate a SUBSCRIBE or FETCH command once the transport session is established. The namespace-name string MAY be followed by a series of key-value parameters, each separated from the preceding element by &. These key-value parameters are intended for processing by the client and, being part of the fragment, are not transferred to the server at connection time. Certain of the fragment key-value parameters have a reserved meaning, as defined by Section 11.1.1.

11.1.1. Reserved fragment parameters

Table 8 defines reserved key names for the parameter portion of the msf-fragment. Keynames are case-sensitive.

Name	Description
wallclock-range	A subclip defined by a wallclock time range
mediatime-range	A subclip defined by a media time range
location-range	A subclip defined by a MOQT Location range
c4m	A base64 encoded C4M token
connection	Mandates a connection type

Table 10

* wallclock-range - a range defined by start and end wallclock times, each expressed as milliseconds since Unix Epoch and separated by a "-" dash. The dash and end time MAY be omitted to indicate an open range. The range definition is inclusive.

* mediatime-range - a range defined by start and end media times, each expressed as milliseconds and separated by a "-" dash. The dash and end time MAY be omitted to indicate an open range. The range definition is inclusive.

* location-range - a range defined by start and end media MOQT Location separated by a "-" dash. Range definitions are inclusive. MOQT Location is expressed as Group ID and Object ID separated by a "." dot. End Location may be omitted to indicate an open range which continues to the end of the content. End Object ID may be omitted, indicating the whole end group is included in the range. The "." dot and "-" dash separators MUST be omitted when the second value is omitted.

* c4m - a base64 encoded token, as defined by [C4M].

* connection - mandates the client to use a particular connection type when connecting to the server. There are two allowed values - "q" or "wt". "q" indicates that a Native QUIC connection MUST be used. "wt" indicates that a WebTransport connection MUST be used.

If multiple ranges are specified within the same URL for the same parameter, the client MUST process the union of those ranges.

Example fragment parameters:

* connection=q

* connection=wt

* wallclock-range=1761759637565-1761759836189

* wallclock-range=1761751753894

* mediatime-range=0-13421

* mediatime-range=982

* location-range=34.0-2145.16

* location-range=16.24 (open range starting at Group ID 16 Object ID 24)

* location-range=16-24 (range from Group ID 16 through to and including all Objects in Group ID 24)

* c4m=gqhkYWxnIGVzaGFyqGR0eXBNhdZ9hdWQAY3VybGZlbWlzcwZleWV2aW5uZWlhdGVwQWNyZW5lY

11.1.2. MSF Namespace-Name String Encoding

To represent MoQ Tuples (which are sequences of byte strings) within the URL fragment, MSF uses a specific encoding convention, which is normatively defined by MOQT [MoQTransport] and repeated here for convenience:

* Hierarchy: Each element of the Namespace tuple is rendered in order, separated by a single hyphen (-).

* Delimiter: The Track Name is appended to the end, separated from the namespace by a double hyphen (--).

* Character Escaping:

  - Unreserved characters (a-z, A-Z, 0-9, _) are represented literally.

- All other byte values (including hyphens and periods used as data) MUST be percent-encoded using a period (.) followed by two lowercase hexadecimal digits (e.g., a literal hyphen in a name becomes .2d).

Note: This encoding ensures that the structural delimiters (- and --) remain unambiguous.

11.1.3. Example MSF URLs

* URL pointing at a catalog track (either WebTransport or native QUIC may be used): moqt://example.com/server/config?a=1&b=2#msf:customer-livestream-123--catalog
  - Session: example.com/server/config?a=1&b=2
  - Streaming format type: msf
  - Namespace: (’customer’, ’livestream’, ’123’)
  - Track Name: ’catalog’
* URL with a required raw QUIC connection pointing at a catalog: moqt://example.com/relay-app/relayID#msf:customerID-broadcastID--catalog&connection=q
* URL pointing at a non-catalog track, with a required WebTransport connection moqt://example.com/relay-app/relayID#msf:customerID-broadcastID--video&connection=wt
* URL pointing at a subclip: moqt://example.com/relay-app/relayID#msf:customerID-broadcastID--catalog&location-range=34-64
* URL pointing at a catalog and supplying a token for the client: (linebreaks for display only) moqt://example.com/relay-app/relayID#msf:customerID-broadcastID--catalog&c4m=gqhkYWxnIGVzaGFyqGR0eXBNhdZ9hdWQAY3VybGZlbWlzcwZleWV2aW5uZWlhdGVwQWNyZW5lYnJmcmVqMTIzNDU2NzgwMHZpc3VlZF9hdD0xNzMwNDM
* URL pointing at a catalog and supplying a token for the client along with a separate token for the server: (linebreaks for display only) moqt://example.com/relay-app/relayID?token=HTRCII74GHFT@JHBCVSW56HKKneH2Dbyq6NHBI2#msf:customerID-broadcastID--catalog&c4m=gqhkYWxnIGVzaGFyqGR0eXBNhdZ9hdWQAY3VybGZlbWlzcwZleWV2aW5uZWlhdGVwQWNyZW5lYnJmcmVqMTIzNDU2NzgwMHZpc3VlZF9hdD0xNzMwNDM

11.2. Initiating a broadcast

An MSF publisher MUST publish a catalog track object before publishing any media track objects.

11.3. Ending a live broadcast

After publishing a catalog and defining tracks carrying live content, an original publisher can deliver a deterministic signal to all subscribers that the broadcast is complete by taking the following steps:

* Send a SUBSCRIBE_DONE (See MOQT Sect 8.1.2) message for all active tracks using status code 0x2 Track Ended.

* If the live stream is being converted instantly to a VOD asset, then publish an independent (non-delta) catalog update which, for each track, sets isLive Section 5.2.7 to FALSE and adds a track duration Section 5.2.35 field.

* If the live stream is being terminated permanently without conversion to VOD, then publish an independent catalog update which signals isComplete Section 5.1.3 as TRUE and which contains an empty Tracks Section 5.1.4 field.

11.4. Authorization

MSF supports token-based authorization through pluggable authentication schemes. Both original publishers and end subscribers can independently acquire authorization tokens and present them to relays. Relays use these tokens to authorize whether an end subscriber is permitted to setup the connection, as well as to receive content and/or to send content.

11.4.1. Discovering Authorization Requirements

End subscribers discover authorization requirements by parsing the catalog. For each track of interest, examine the authInfo field. If present, the track requires authorization and the subscriber must obtain appropriate credentials for one of the schemes listed in the field.

Original publishers and end subscribers may obtain authorization tokens from multiple sources, including out-of-band provisioning or as a parameter in the connection URI. These tokens can be presented either in the connection setup message (CLIENT_SETUP) or in control messages (SUBSCRIBE, FETCH, PUBLISH, PUBLISH_NAMESPACE) as described in Section 11.4.3.

11.4.2. Token Acquisition

Token acquisition is out of scope for this specification. Both original publishers and end subscribers independently obtain tokens through mechanisms such as:

* Direct authentication with a distribution service
* OAuth 2.0 or OpenID Connect flows
* Out-of-band provisioning

The token acquisition endpoint and flow depend on the authorization scheme and deployment configuration. Original publishers and end subscribers MAY use the same or different authorization services depending on the deployment.

Tokens MAY also be supplied as parameters in the connection URI. When a connection URI contains token parameters, the subscriber extracts the token value and includes it in the appropriate MOQT message. For example:

```
moqt://relay.example.com/moqt#namespace--name&token=eyJhbGciOiJFZERTQSJ9...
```

URI parameters provide a convenient mechanism for distributing pre-authorized playback links. The parameter name and format are deployment-specific.

11.4.3. Presenting Authorization

Once tokens are obtained, both original publishers and end subscribers present them to relays according to the scheme specification. Tokens MAY be presented in the SETUP message (CLIENT_SETUP or SERVER_SETUP) using the AUTHORIZATION TOKEN setup parameter, or in individual control messages using the AUTHORIZATION TOKEN message parameter.

When a token is associated with a track, it MUST be included in ALL control messages that accept the AUTHORIZATION TOKEN parameter and are associated with that track. For end subscribers, this includes SUBSCRIBE, SUBSCRIBE_NAMESPACE, FETCH, and REQUEST_UPDATE messages. For original publishers, this includes PUBLISH and PUBLISH_NAMESPACE messages. This requirement applies regardless of whether the token was also provided in SETUP.

11.4.4. Handling Authorization Failures

When a relay rejects an authorization token, subscribers SHOULD handle the failure gracefully:

* If a SUBSCRIBE or FETCH request is rejected due to invalid or expired authorization, the subscriber MAY attempt to obtain a fresh token and retry the request.

* If a token is rejected due to insufficient scope, the subscriber SHOULD NOT retry with the same token.

* For interactive applications, subscribers MAY prompt users to re-authenticate when authorization fails.

* Subscribers SHOULD implement exponential backoff when retrying failed authorization attempts to avoid overwhelming the relay or token issuer.

The specific error codes and retry semantics are defined by the authorization scheme specifications. See [PrivacyPassAuth] for Privacy Pass error handling and [C4M] for CAT error handling.

12. MSF Properties

MSF defines MOQT Track Properties and Object Properties (see [MoQTransport]) to signal metadata about MSF tracks and objects. These properties are carried in MOQT control messages and object headers, allowing endpoints to learn track and object characteristics before processing payload data.

12.1. Compression Signaling

MSF provides two mutually exclusive mechanisms to signal compression of track payloads, including catalogs (Section 5), media timeline tracks (Section 7), and event timeline tracks (Section 8). Publishers MUST use one of the following approaches:

* *Track Property* (Section 12.1.1): Signals that ALL objects in the track are compressed using the specified algorithm.

* *Object Property* (Section 12.1.2): Signals compression on individual objects, allowing a mixture of compressed and uncompressed objects within the same track.

A publisher MUST NOT use both mechanisms on the same track. If the track property is set, then every object in the track is compressed and the object property MUST NOT be present. If compression varies between objects, then the track property MUST NOT be set and the object property MUST be used on each compressed object.

The compression algorithm values used by both mechanisms are:

Value	Compression Algorithm	Reference
0	None (uncompressed)	RFC XXXX
1	GZIP	[GZIP]

Table 11

Table: MSF Compression Values

All MSF implementations MUST support both uncompressed payloads (value 0 or property absent) and GZIP compressed payloads (value 1).

12.1.1. MSF_COMPRESSION Track Property

The MSF_COMPRESSION track property signals that ALL objects in the track are compressed using the specified algorithm. This mechanism is appropriate when every object published on the track uses the same compression.

Publishers MUST include the MSF_COMPRESSION track property in the PUBLISH message (publisher-initiated flow) or SUBSCRIBE_OK (subscriber-initiated flow).

Subscribers MUST check for this property in the corresponding message before processing the track payload.

If the property is absent, and no per-object compression is signaled, the subscriber MUST treat the payload as uncompressed. If the property is present with a value the subscriber does not support, the subscriber MUST NOT attempt to process the payload and SHOULD unsubscribe from the track.

12.1.2. MSF_COMPRESSION Object Property

The MSF_COMPRESSION object property signals that an individual object is compressed. This mechanism is appropriate when a track contains a mixture of compressed and uncompressed objects. For example, a catalog track where the first object in a group (the complete catalog) is compressed, but subsequent delta update objects within the same group are uncompressed.

Publishers MUST include the MSF_COMPRESSION object property on each compressed object.

Subscribers MUST check for this property on each received object before processing its payload. If the property is absent on an object, the subscriber MUST treat that object's payload as uncompressed. If the property is present with a value the subscriber does not support, the subscriber MUST NOT attempt to process that object.

13. Security Considerations

ToDo

14. IANA Considerations

14.1. "MOQT URI Fragment Types" registry

This document creates a new entry in the "MOQT URI Fragment Types" registry (see [MoQTransport] Sect 14.3).

Fragment Type	Description	Specification
msf	MOQT Streaming Format	this

Table 12

14.2. "MSF Event Timeline Types" registry

This document establishes the "MSF Event Timeline Types" registry. This registry lists the event types that can be used with the eventType field Section 5.2.5 in MSF catalogs.

New entries in this registry are subject to Expert Review policy as defined in [RFC8126].

The initial contents of this registry are:

Event Type	Description	Specification
urn:scte:scte35:2013:bin	SCTE-35 binary splice_info_section	[SCTE35-MSF]
urn:scte:scte35:2013:xml	SCTE-35 XML representation	[SCTE35-MSF]
urn:msf:timedtext:webvtt	WebVTT timed text cues	[WebVTT-MSF]
urn:msf:timedtext:imsc1	IMSC1 timed text cues	[IMSC1-MSF]

Table 13

14.3. MSF_COMPRESSION Track Property

This document requests IANA to register the following entry in the "Track Properties" registry established by [MoQTransport] (Section 14.4):

Property Name	Property ID	Value Type	Reference
MSF_COMPRESSION	TBD	varint	RFC XXXX

Table 14

The MSF_COMPRESSION track property indicates that ALL objects on the track are compressed using the specified algorithm. See Section 12.1.1 for the full specification.

14.4. MSF_COMPRESSION Object Property

This document requests IANA to create a new "MSF Compression Algorithms" registry with the following initial values:

Value	Compression Algorithm	Reference
0	None (uncompressed)	RFC XXXX
1	GZIP	[GZIP]

Table 15

Values 2-127 are available for registration via Standards Action.
Values 128 and above are reserved for private use.

15. References

15.1. Normative References

[BASE64] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[C4M] Law, W., Lemmons, C., Simon, G., and S. Nandakumar, "Authentication scheme for MOQT using Common Access Tokens", Work in Progress, Internet-Draft, draft-ietf-moq-c4m-00, 19 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-c4m-00>.

[GZIP] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, DOI 10.17487/RFC1952, May 1996, <https://www.rfc-editor.org/rfc/rfc1952>.

[IMSC1] "W3C, TTML Profiles for Internet Media Subtitles and Captions 1.0 (IMSC1)", April 2016, <https://www.w3.org/TR/ttml-imsc1/>.

[JSON] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[LANG] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/rfc/rfc5646>.

[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>.

[MIME] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[MOQLOG] Jennings, C. F. and S. Nandakumar, "Logging over Media over QUIC Transport", Work in Progress, Internet-Draft, draft-jennings-moq-log-03, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-jennings-moq-log-03>.

[MOQMETRICS] Jennings, C. F. and S. Nandakumar, "Metrics over MOQT", Work in Progress, Internet-Draft, draft-jennings-moq-metrics-02, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-jennings-moq-metrics-02>.

[MoQTransport] 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>.

[PrivacyPassAuth] Nandakumar, S., Jennings, C. F., and T. Meunier, "Privacy Pass Authentication for Media over QUIC (MoQ)", Work in Progress, Internet-Draft, draft-ietf-moq-privacy-pass-auth-02, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-privacy-pass-auth-02>.

[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>.

[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>.

[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>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[SecureObjects] Jennings, C. F., Nandakumar, S., and R. Barnes, "End-to-End Secure Objects for Media over QUIC Transport", Work in Progress, Internet-Draft, draft-jennings-moq-secure-objects-04, 8 February 2026, <https://datatracker.ietf.org/doc/html/draft-jennings-moq-secure-objects-04>.

[WEBCODECS-CODEC-REGISTRY] "WebCodecs Codec Registry", September 2024, <https://www.w3.org/TR/webcodecs-codec-registry/>.

[WEBVTT] "World Wide Web Consortium (W3C), WebVTT: The Web Video Text Tracks Format", April 2019, <https://www.w3.org/TR/webvtt1/>.

15.2. Informative References

[E2EE-MLS] Jennings, C. F., Nandakumar, S., and R. Barnes, "End-to-end Security for Media over QUIC", Work in Progress, Internet-Draft, draft-jennings-moq-e2ee-mls-03, 30 June 2025, <https://datatracker.ietf.org/doc/html/draft-jennings-moq-e2ee-mls-03>.

[IMSC1-MSF] "IMSC1 Packaging for MOQT Streaming Format", 2026, <https://github.com/suhasHere/imsc1-msf>.

[SCTE214-1] "SCTE 214-1: MPEG DASH for IP-Based Cable Services Part 1 - MPD Constraints and Extensions", 2022, <https://www.scte.org/standards/library/catalog/scte-214-1-mpeg-dash-for-ip-based-cable-services-part-1-mpd-constraints-and-extensions/>.

[SCTE35-MSF] "SCTE-35 over MSF Event Timeline", 2026, <https://github.com/wilaw/SCTE35-over-MSF-Event-Timeline>.

[WebVTT-MSF]
"WebVTT Packaging for MOQT Streaming Format", 2026,
<https://github.com/suhasHere/webvtt-msf>.

Acknowledgments

* the MoQ Workgroup and mailing lists.

Contributors

The following persons where the co-authors of the individual draft (draft-law-moq-warpstreamingformat) this document is based on:

* Luke Curley

* Victor Vasiliev

* Suhas Nandakumar

* Kirill Pugin

* Will Law

Authors' Addresses

Will Law
Akamai
Email: wilaw@akamai.com

Suhas Nandakumar
Cisco
Email: snandaku@cisco.com