Media Over QUIC
Internet-Draft
Intended status: Informational
Expires: 11 July 2026

L. Pardue
Cloudflare
M. Engelbart
Technical University of Munich
A. Sharma
Meta

7 January 2026

MoQ qlog event definitions
draft-pardue-moq-qlog-moq-events-04

Abstract

This document defines a qlog event schema containing concrete events for MoQ.

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://LPardue.github.io/draft-pardue-moq-qlog-moq-events/draft-pardue-moq-qlog-moq-events.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-pardue-moq-qlog-moq-events/.

Discussion of this document takes place on the Media Over QUIC 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/LPardue/draft-pardue-moq-qlog-moq-events.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a qlog event schema (Section 8 of [QLOG-MAIN]) containing concrete events for Media over QUIC Transport [MOQT].

The event namespace with identifier moqt is defined; see Section 3. In this namespace multiple events derive from the qlog abstract Event class (Section 7 of [QLOG-MAIN]), each extending the "data" field and defining their "name" field values and semantics.

Table 1 summarizes the name value of each event type that is defined in this specification. Some event data fields use complex data types. These are represented as enums or re-usable definitions, which are grouped together on the bottom of this document for clarity.

Name value	Importance	Definition
moqt:control_message_created	Core	Section 4.1
moqt:control_message_parsed	Core	Section 4.2
moqt:stream_type_set	Core	Section 4.3
moqt:object_datagram_created	Core	Section 4.4
moqt:object_datagram_parsed	Core	Section 4.5
moqt:subgroup_header_created	Core	Section 4.6
moqt:subgroup_header_parsed	Core	Section 4.7
moqt:subgroup_object_created	Core	Section 4.8
moqt:subgroup_object_parsed	Core	Section 4.9
moqt:fetch_header_created	Core	Section 4.10
moqt:fetch_header_parsed	Core	Section 4.11
moqt:fetch_object_created	Core	Section 4.12
moqt:fetch_object_parsed	Core	Section 4.13

Table 1: MOQT Events

1.1. Usage with QUIC

The events described in this document can be used with or without logging the related QUIC events defined in [QLOG-QUIC]. If used with QUIC events, the QUIC document takes precedence in terms of recommended filenames and trace separation setups.

If used without QUIC events, it is recommended that the implementation assign a globally unique identifier to each MOQT session. This ID can then be used as the value of the qlog "group_id" field, as well as the qlog filename or file identifier, potentially suffixed by the vantagepoint type (For example, abcd1234_server.qlog would contain the server-side trace of the connection with GUID abcd1234).

2. Conventions and Definitions

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

The event and data structure definitions in ths document are expressed in the Concise Data Definition Language [CDDL] and its extensions described in [QLOG-MAIN].

The following fields from [QLOG-MAIN] are imported and used: name, namespace, type, data, group_id, RawInfo, and time-related fields.

Events are defined with an importance level as described in Section 8.3 of [QLOG-MAIN].

As is the case for [QLOG-MAIN], the qlog schema definitions in this document are intentionally agnostic to serialization formats. The choice of format is an implementation decision.

3. Event Schema Definition

This document describes how the MOQT protocol is expressed in qlog with an event schema. Per the requirements in Section 8 of [QLOG-MAIN], this document registers the moqt namespace. The event schema URI is urn:ietf:params:qlog:events:moqt.

3.1. Draft Event Schema Identification

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

Only implementations of the final, published RFC can use the events belonging to the event schema with the URI urn:ietf:params:qlog:events:moqt. Until such an RFC exists, implementations MUST NOT identify themselves using this URI.

Implementations of draft versions of the event schema MUST append the string "-" and the corresponding draft number to the URI. For example, draft 99 of this document is identified using the URI urn:ietf:params:qlog:events:moqt-99.

The namespace identifier itself is not affected by this requirement.

4. MOQT Events

MOQT events extend the $ProtocolEventData extension point defined in [QLOG-MAIN]. Additionally, they allow for direct extensibility by their use of per-event extension points via the $$ CDDL "group socket" syntax, as also described in [QLOG-MAIN].

```
MOQTEventData = MOQTControlMessageCreated /
                MOQTControlMessageParsed /
                MOQTStreamTypeSet /
                MOQTObjectDatagramCreated /
                MOQTObjectDatagramParsed /
                MOQTSubgroupHeaderCreated /
                MOQTSubgroupHeaderParsed /
                MOQTSubgroupObjectCreated /
                MOQTSubgroupObjectParsed /
                MOQTFetchHeaderCreated /
                MOQTFetchHeaderParsed /
                MOQTFetchObjectCreated /
                MOQTFetchObjectParsed

$ProtocolEventData /= MOQTEventData
```

Figure 1: MOQTEventData definition and ProtocolEventData extension

MOQT events are logged when a certain condition happens at the application layer, and there isn't always a one to one mapping between HTTP and QUIC events. The exchange of data between the HTTP and QUIC layer is logged via the "stream_data_moved" and "datagram_data_moved" events in [QLOG-QUIC].

The concrete MOQT event types are further defined below, their type identifier is the heading name.

Some MOQT messages include a reason phrase that can provide additional information in the format of a byte sequences. However, these sequences are not guaranteed to be presentable as UTF-8. In order to accomodate various encodings, where the wire image of a message includes a reason phrase, the MOQT qlog event type, includes two option fields: reason (for UTF-8) and reason_bytes (a hex-encoded string representing raw bytes). Implementations SHOULD log at least one format, but MAY log both or none.

4.1. control_message_created

The control_message_created event is emitted when a control message is created. It has Core importance level.

The definition of control message content is in Section 5.6.

```
MOQTControlMessageCreated = {
    stream_id: uint64
    ? length: uint64
    message: $MOQTControlMessage
    ? raw: RawInfo

    * $$moqt-controlmessagecreated-extension
}
```

Figure 2: MOQTControlMessageCreated definition

4.2. control_message_parsed

The control_message_parsed event is emitted when a control message is parsed. It has Core importance level.

The definition of control message content is in Section 5.6.

```
MOQTControlMessageParsed = {
    stream_id: uint64
    ? length: uint64
    message: $MOQTControlMessage
    ? raw: RawInfo

    * $$moqt-controlmessageparsed-extension
}
```

Figure 3: MOQTControlMessageParsed definition

4.3. stream_type_set

The stream_type_set event conveys when a MOQT stream type becomes known. It has Base importance level.

```
MOQTStreamTypeSet = {
    ? owner: Owner
    stream_id: uint64
    stream_type: $MOQTStreamType

    * $$moqt-streamtypeset-extension
}

$MOQTStreamType /=  "control" /
                    "subgroup_header" /
                    "fetch_header"
```

Figure 4: MOQTStreamTypeSet definition

4.4. object_datagram_created

The object_datagram_created event is emitted when the OBJECT_DATAGRAM message is created. It has Core importance level.

```
MOQTObjectDatagramCreated = {
    track_alias: uint64
    group_id: uint64
    ? object_id: uint64
    publisher_priority: uint8
    ? extension_headers_length: uint64
    ? extension_headers: [* MOQTExtensionHeader]
    ? object_status: uint64
    ? object_payload: RawInfo
    end_of_group: bool

    * $$moqt-objectdatagramcreated-extension
}
```

Figure 5: MOQTObjectDatagramCreated definition

4.5. object_datagram_parsed

The object_datagram_parsed event is emitted when the OBJECT_DATAGRAM message is parsed. It has Core importance level.

```
MOQTObjectDatagramParsed = {
    track_alias: uint64
    group_id: uint64
    ? object_id: uint64
    publisher_priority: uint8
    ? extension_headers_length: uint64
    ? extension_headers: [* MOQTExtensionHeader]
    ? object_status: uint64
    ? object_payload: RawInfo
    end_of_group: bool

    * $$moqt-objectdatagramparsed-extension
}
```

Figure 6: MOQTObjectDatagramParsed definition

4.6. subgroup_header_created

The subgroup_header_created event is emitted when a stream begins and a SUBGROUP_HEADER is created. It has Core importance level; see Section 9.2 of [QLOG-MAIN].

The SUBGROUP_HEADER object in MoQT uses 12 type values to encode various properties. The subgroup_header_created event conveys these as explicit fields, such as contains_end_of_group.

If the subgroup_id is the object_id of the first object, the subgroup_id is omitted. Otherwise, it is included with the relevant value.

```
MOQTSubgroupHeaderCreated = {
    stream_id: uint64
    track_alias: uint64
    group_id: uint64
    ? subgroup_id: uint64
    publisher_priority: uint8
    contains_end_of_group: bool
    extensions_present: bool

    * $$moqt-subgroupheadercreated-extension
}
```

Figure 7: MOQTSubgroupHeaderCreated definition

4.7. subgroup_header_parsed

The subgroup_header_parsed event is emitted when the SUBGROUP_HEADER is parsed. It has Core importance level.

The SUBGROUP_HEADER object in MoQT uses 12 type values to encode various properties. The event conveys these as explicit fields, such as contains_end_of_group.

If the subgroup_id is the object_id of the first object, the subgroup_id is omitted. Otherwise, it is included with the relevant value.

```
MOQTSubgroupHeaderParsed = {
    stream_id: uint64
    track_alias: uint64
    group_id: uint64
    ? subgroup_id: uint64
    publisher_priority: uint8
    contains_end_of_group: bool
    extensions_present: bool

    * $$moqt-subgroupheaderparsed-extension
}
```

Figure 8: MOQTSubgroupHeaderParsed definition

4.8. subgroup_object_created

The subgroup_object_created event is emitted when a subgroup object is created. It has Core importance level.

```
MOQTSubgroupObjectCreated = {
    stream_id: uint64
    ? group_id: uint64
    ? subgroup_id: uint64
    object_id: uint64
    extension_headers_length: uint64
    ? extension_headers: [* MOQTExtensionHeader]
    object_payload_length: uint64
    ? object_status: uint64
    ? object_payload: RawInfo

    * $$moqt-subgroupobjectcreated-extension
}
```

Figure 9: MOQTSubgroupObjectCreated definition

4.9. subgroup_object_parsed

The subgroup_object_parsed event is emitted when a subgroup object is parsed. It has Core importance level.

```
MOQTSubgroupObjectParsed = {
    stream_id: uint64
    ? group_id: uint64
    ? subgroup_id: uint64
    object_id: uint64
    extension_headers_length: uint64
    ? extension_headers: [* MOQTExtensionHeader]
    object_payload_length: uint64
    ? object_status: uint64
    ? object_payload: RawInfo

    * $$moqt-subgroupobjectparsed-extension
}
```

Figure 10: MOQTSubgroupObjectParsed definition

4.10. fetch_header_created

The fetch_header_created event is emitted when a stream begins and a FETCH_HEADER is created. It has Core importance level.

```
MOQTFetchHeaderCreated = {
    stream_id: uint64
    request_id: uint64

    * $$moqt-fetchheadercreated-extension
}
```

Figure 11: MOQTFetchHeaderCreated definition

4.11. fetch_header_parsed

The fetch_header_parsed event is emitted when the SUBGROUP_HEADER is parsed. It has Core importance level.

```
MOQTFetchHeaderParsed = {
    stream_id: uint64
    request_id: uint64

    * $$moqt-fetchheaderparsed-extension
}
```

Figure 12: MOQTFetchHeaderParsed definition

4.12. fetch_object_created

The fetch_object_created event is emitted when a fetch object is created. It has Core importance level.

```
MOQTFetchObjectCreated = {
    stream_id: uint64
    group_id: uint64
    subgroup_id: uint64
    object_id: uint64
    publisher_priority: uint8
    extension_headers_length: uint64
    ? extension_headers: [* MOQTExtensionHeader]
    object_payload_length: uint64
    ? object_status: uint64
    ? object_payload: RawInfo

    * $$moqt-fetchobjectcreated-extension
}
```

Figure 13: MOQTFetchObjectCreated definition

4.13. fetch_object_parsed

The fetch_object_parsed event is emitted when a fetch object is parsed. It has Core importance level.

```
MOQTFetchObjectParsed = {
    stream_id: uint64
    group_id: uint64
    subgroup_id: uint64
    object_id: uint64
    publisher_priority: uint8
    extension_headers_length: uint64
    ? extension_headers: [* MOQTExtensionHeader]
    object_payload_length: uint64
    ? object_status: uint64
    ? object_payload: RawInfo

    * $$moqt-fetchobjectparsed-extension
}
```

Figure 14: MOQTFetchObjectParsed definition

5. MOQT Data Type Definitions

The following data type definitions can be used in MOQT events.

5.1. Owner

```
Owner = "local" /
        "remote"
```

Figure 15: Owner definition

5.2. MOQTSetupParameter

The generic $MOQTSetupParameter is defined here as a CDDL "type socket" extension point. It can be extended to support additional MOQT Setup Parameters.

```
; The MOQTSetupParameter is any key-value map (e.g., JSON object)
$MOQTSetupParameter /= {
    * text => any
}
```

Figure 16: MOQTSetupParameter type socket definition

```
MOQTBaseSetupParameters /=  MOQTAuthoritySetupParameter /
                            MOQTAuthorizationTokenSetupParameter /
                            MOQTPathSetupParameter /
                            MOQTMaxRequestIdSetupParameter /
                            MOQTMaxAuthTokenCacheSizeSetupParameter /
                            MOQTImplementationSetupParameter /
                            MOQTUnknownSetupParameter

$MOQTSetupParameter /= MOQTBaseSetupParameters
```

Figure 17: MOQTBaseSetupParameters definition

5.2.1. MOQTAuthoritySetupParameter

```
MOQTAuthoritySetupParameter = {
  name: "authority"
  value: text
}
```

Figure 18: MOQTAuthoritySetupParameter definition

5.2.2. MOQTAuthorizationTokenSetupParameter

```
MOQTAuthorizationTokenSetupParameter = {
  name: "authorization_token"
  alias_type: $MOQTAliasType
  ? token_alias: uint64
  ? token_type: uint64
  ? token_value: RawInfo
}

$MOQTAliasType /=  "delete" /
                   "register" /
                   "use_alias" /
                   "use_value"
```

Figure 19: MOQTAuthorizationTokenSetupParameter definition

5.2.3. MOQTPathSetupParameter

```
MOQTPathSetupParameter = {
  name: "path"
  value: text
}
```

Figure 20: MOQTPathSetupParameter definition

5.2.4. MOQTMaxRequestIdSetupParameter

```
MOQTMaxRequestIdSetupParameter = {
  name: "max_request_id"
  value: uint64
}
```

Figure 21: MOQTMaxRequestIdSetupParameter definition

5.2.5. MOQTMaxAuthTokenCacheSizeSetupParameter

```
MOQTMaxAuthTokenCacheSizeSetupParameter = {
  name: "max_auth_token_cache_size"
  value: uint64
}
```

Figure 22: MOQTMaxAuthTokenCacheSizeSetupParameter definition

5.2.6. MOQTImplementationSetupParameter

```
MOQTImplementationSetupParameter = {
  name: "implementation"
  value: text
}
```

Figure 23: MOQTImplementationSetupParameter definition

5.2.7. MOQTUnknownSetupParameter

```
MOQTUnknownSetupParameter = {
  name:"unknown"
  name_bytes: uint64
  ? length: uint64
  ? value: uint64
  ? value_bytes: RawInfo
}
```

Figure 24: MOQTUnknownSetupParameter definition

5.3. MOQTParameter

The generic $MOQTParameter is defined here as a CDDL "type socket" extension point. It can be extended to support additional MOQT Parameters.

```
; The MOQTParameter is any key-value map (e.g., JSON object)
$MOQTParameter /= {
    * text => any
}
```

Figure 25: MOQTParameter type socket definition

```
MOQTBaseParameters /= MOQTAuthorizationTokenParameter /
                      MOQTDeliveryTimeoutParameter /
                      MOQTMaxCacheDurationParameter /
                      MOQTUnknownParameter

$MOQTParameter /= MOQTBaseParameters
```

Figure 26: MOQTBaseParameters definition

5.3.1. MOQTAuthorizationTokenParameter

```
MOQTAuthorizationTokenParameter = {
  name: "authorization_token"
  alias_type: uint64
  ? token_alias: uint64
  ? token_type: uint64
  ? token_value: RawInfo
}
```

Figure 27: MOQTAuthorizationTokenParameter definition

5.3.2. MOQTDeliveryTimeoutParameter

```
MOQTDeliveryTimeoutParameter = {
  name: "delivery_timeout"
  value: uint64
}
```

Figure 28: MOQTDeliveryTimeoutParameter definition

5.3.3. MOQTMaxCacheDurationParameter

```
MOQTMaxCacheDurationParameter = {
  name: "max_cache_duration"
  value: uint64
}
```

Figure 29: MOQTMaxCacheDurationParameter definition

5.3.4. MOQTUnknownParameter

```
MOQTUnknownParameter = {
  name:"unknown"
  name_bytes: uint64
  ? length: uint64
  ? value: uint64
  ? value_bytes: RawInfo
}
```

Figure 30: MOQTUnknownParameter definition

5.4. MOQTByteString

The MOQTByteString type allows representing MOQT bytestrings, such as the value of a Track or Track Namespace tuple field, using two different encodings. The value field can be used for bytestrings that can be encoded in UTF-8. The value_bytes field can be used for bytestrings of any type by using the hexstring encoding.

Implementations SHOULD populate one of either the value or value_bytes field. Populating both fields is redundant.

```
MOQTByteString = {
  ? value: text
  ? value_bytes: hexstring
}
```

Figure 31: MOQTByteString definition

5.5. MOQTLocation

A Location, as defined in Section 1.3.1 of [MOQT]

```
MOQTLocation = {
  group: uint64
  object: uint64
}
```

Figure 32

5.6. MOQTControlMessage

The generic $MOQTControlMessage is defined here as a CDDL "type socket" extension point. It can be extended to support additional MOQT control message types.

```
; The MOQTControlMessage is any key-value map (e.g., JSON object)
$MOQTControlMessage /= {
    * text => any
}
```

Figure 33: MOQTControlMessage type socket definition

The MOQT control message types defined in this document are as follows:

```
MOQTBaseControlMessages = MOQTClientSetupMessage /
                          MOQTServerSetupMessage /
                          MOQTGoaway /
                          MOQTMaxRequestId /
                          MOQTRequestsBlocked /
                          MOQTSubscribe /
                          MOQTSubscribeOk /
                          MOQTSubscribeError /
                          MOQTSubscribeUpdate /
                          MOQTUnsubscribe /
                          MOQTPublishDone /
                          MOQTPublish /
                          MOQTPublishOk /
                          MOQTPublishError /
                          MOQTFetch /
                          MOQTFetchOk /
                          MOQTFetchError /
                          MOQTFetchCancel /
                          MOQTTrackStatus /
                          MOQTTrackStatusOk /
                          MOQTTrackStatusError /
                          MOQTPublishNamespace /
                          MOQTPublishNamespaceOk /
                          MOQTPublishNamespaceError /
                          MOQTPublishNamespaceDone /
                          MOQTPublishNamespaceCancel /
                          MOQTSubscribeNamespace /
                          MOQTSubscribeNamespaceOk /
                          MOQTSubscribeNamespaceError /
                          MOQTUnsubscribeNamespace

$MOQTControlMessage /= MOQTBaseControlMessages
```

Figure 34: MOQTBaseControlMessages definition

5.6.1. MOQTClientSetupMessage

```
MOQTClientSetupMessage = {
  type: "client_setup"
  number_of_supported_versions: uint64
  supported_versions: [* uint64]
  number_of_parameters: uint64
  ? setup_parameters: [* $MOQTSetupParameter]
}
```

Figure 35: MOQTClientSetupMessage definition

5.6.2. MOQTServerSetupMessage

```
MOQTServerSetupMessage = {
  type: "server_setup"
  selected_version: uint64
  number_of_parameters: uint64
  ? setup_parameters: [* $MOQTSetupParameter]
}
```

Figure 36: MOQTServerSetupMessage definition

5.6.3. MOQTGoaway

```
MOQTGoaway = {
  type: "goaway"
  ? length: uint64
  new_session_uri: RawInfo
}
```

Figure 37: MOQTGoaway definition

5.6.4. MOQTMaxRequestId

```
MOQTMaxRequestId = {
  type: "max_request_id"
  request_id: uint64
}
```

Figure 38: MOQTMaxRequestId definition

5.6.5. MOQTRequestsBlocked

```
MOQTRequestsBlocked = {
  type: "requests_blocked"
  maximum_request_id: uint64
}
```

Figure 39: MOQTRequestsBlocked definition

5.6.6. MOQTSubscribe

```
MOQTSubscribe = {
  type: "subscribe"
  request_id: uint64
  track_namespace: [ *MOQTByteString]
  track_name: MOQTByteString
  subscriber_priority: uint8
  group_order: uint8
  forward: uint8
  filter_type: uint64
  ? start_location: MOQTLocation
  ? end_group: uint64
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 40: MOQTSubscribe definition

5.6.7. MOQTSubscribeOk

```
MOQTSubscribeOk = {
  type: "subscribe_ok"
  request_id: uint64
  track_alias: uint64
  expires: uint64
  group_order: uint8
  content_exists: uint8
  ? largest_location: MOQTLocation
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 41: MOQTSubscribeOk definition

5.6.8. MOQTSubscribeError

```
MOQTSubscribeError = {
  type: "subscribe_error"
  request_id: uint64
  error_code: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 42: MOQTSubscribeError definition

5.6.9. MOQTSubscribeUpdate

```
MOQTSubscribeUpdate = {
  type: "subscribe_update"
  request_id: uint64
  subscription_request_id: uint64
  start_location: MOQTLocation
  end_group: uint64
  subscriber_priority: uint8
  forward: uint8
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 43: MOQTSubscribeUpdate definition

5.6.10. MOQTUnsubscribe

```
MOQTUnsubscribe = {
  type: "unsubscribe"
  request_id: uint64
}
```

Figure 44: MOQTUnsubscribe definition

5.6.11. MOQTPublishDone

```
MOQTPublishDone = {
  type: "publish_done"
  request_id: uint64
  status_code: uint64
  stream_count: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 45: MOQTPublishDone definition

5.6.12. MOQTPublish

```
MOQTPublish = {
  type: "publish"
  request_id: uint64
  track_namespace: [ *MOQTByteString]
  track_name: MOQTByteString
  track_alias: uint64
  group_order: uint8
  content_exists: uint8
  ? largest: MOQTLocation
  forward: uint8
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 46: MOQTPublish definition

5.6.13. MOQTPublishOk

```
MOQTPublishOk = {
  type: "publish_ok"
  request_id: uint64
  forward: uint8
  subscriber_priority: uint8
  group_order: uint8
  filter_type: uint64
  ? start: MOQTLocation
  ? end_group: uint64
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 47: MOQTPublishOk definition

5.6.14. MOQTPublishError

```
MOQTPublishError = {
  type: "publish_error"
  request_id: uint64
  error_code: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 48: MOQTPublishError definition

5.6.15. MOQTFetch

```
MOQTFetch = {
  type: "fetch"
  request_id: uint64
  subscriber_priority: uint8
  group_order: uint8
  fetch_type: $MOQTFetchType
  ? standalone_fetch: $MOQTStandaloneFetch
  ? joining_fetch: $MOQTJoiningFetch
  ? parameters: [* $MOQTParameter]
}

$MOQTStandaloneFetch = {
  track_namespace: [ *MOQTByteString]
  track_name: MOQTByteString
  start_location: MOQTLocation
  end_location: MOQTLocation
}

$MOQTJoiningFetch = {
  joining_request_id: uint64
  joining_start: uint64
}

$MOQTFetchType /=  "standalone" /
                   "absolute_joining" /
                   "relative_joining"
```

Figure 49: MOQTFetch definition

5.6.16. MOQTFetchOk

```
MOQTFetchOk = {
  type: "fetch_ok"
  request_id: uint64
  group_order: uint8
  end_of_track: uint8
  end_location: MOQTLocation
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 50: MOQTFetchOk definition

5.6.17. MOQTFetchError

```
MOQTFetchError = {
  type: "fetch_error"
  request_id: uint64
  error_code: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 51: MOQTFetchError definition

5.6.18. MOQTFetchCancel

```
MOQTFetchCancel = {
  type: "fetch_cancel"
  request_id: uint64
}
```

Figure 52: MOQTFetchCancel definition

5.6.19. MOQTTrackStatus

```
MOQTTrackStatus = {
  type: "track_status"
  request_id: uint64
  track_namespace: [ *MOQTByteString]
  track_name: MOQTByteString
  subscriber_priority: uint8
  group_order: uint8
  forward: uint8
  filter_type: uint64
  ? start_location: MOQTLocation
  ? end_group: uint64
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 53: MOQTTrackStatus definition

5.6.20. MOQTTrackStatusOk

```
MOQTTrackStatusOk = {
  type: "track_status_ok"
  request_id: uint64
  track_alias: uint64
  expires: uint64
  group_order: uint8
  content_exists: uint8
  ? largest_location: MOQTLocation
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 54: MOQTTrackStatusOk definition

5.6.21. MOQTTrackStatusError

```
MOQTTrackStatusError = {
  type: "track_status_error"
  request_id: uint64
  error_code: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 55: MOQTTrackStatusError definition

5.6.22. MOQTPublishNamespace

```
MOQTPublishNamespace = {
  type: "publish_namespace"
  request_id: uint64
  track_namespace: [ *MOQTByteString]
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 56: MOQTPublishNamespace definition

5.6.23. MOQTPublishNamespaceOk

```
MOQTPublishNamespaceOk = {
  type: "publish_namespace_ok"
  request_id: uint64
}
```

Figure 57: MOQTPublishNamespaceOk definition

5.6.24. MOQTPublishNamespaceError

```
MOQTPublishNamespaceError = {
  type: "publish_namespace_error"
  request_id: uint64
  error_code: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 58: MOQTPublishNamespaceError definition

5.6.25. MOQTPublishNamespaceDone

```
MOQTPublishNamespaceDone = {
  type: "publish_namespace_done"
  track_namespace: [ *MOQTByteString]
}
```

Figure 59: MOQTPublishNamespaceDone definition

5.6.26. MOQTPublishNamespaceCancel

```
MOQTPublishNamespaceCancel = {
  type: "publish_namespace_cancel"
  track_namespace: [ *MOQTByteString]
  error_code: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 60: MOQTPublishNamespaceCancel definition

5.6.27. MOQTSubscribeNamespace

```
MOQTSubscribeNamespace = {
  type: "subscribe_namespace"
  request_id: uint64
  track_namespace_prefix: [ *MOQTByteString]
  number_of_parameters: uint64
  ? parameters: [* $MOQTParameter]
}
```

Figure 61: MOQTSubscribeNamespace definition

5.6.28. MOQTSubscribeNamespaceOk

```
MOQTSubscribeNamespaceOk = {
  type: "subscribe_namespace_ok"
  request_id: uint64
}
```

Figure 62: MOQTSubscribeNamespaceOk definition

5.6.29. MOQTSubscribeNamespaceError

```
MOQTSubscribeNamespaceError = {
  type: "subscribe_namespace_error"
  request_id: uint64
  error_code: uint64
  ? reason: text
  ? reason_bytes: hexstring
}
```

Figure 63: MOQTSubscribeNamespaceError definition

5.6.30. MOQTUnsubscribeNamespace

```
MOQTUnsubscribeNamespace = {
  type: "unsubscribe_namespace"
  track_namespace_prefix: [ *MOQTByteString]
}
```

Figure 64: MOQTUnsubscribeNamespace definition

5.7. MOQTExtensionHeader

```
MOQTExtensionHeader = {
  header_type: uint64
  ? header_value: uint64
  ? header_length: uint64
  ? payload: RawInfo
}
```

Figure 65: Extension Header definition

6. Security Considerations

The security and privacy considerations discussed in [QLOG-MAIN] apply to this document as well.

7. IANA Considerations

This document registers a new entry in the "qlog event schema URIs" registry (created in Section 15 of [QLOG-MAIN]).

Event schema URI: urn:ietf:params:qlog:events:moqt

Namespace moqt

Event Types control_message_created, control_message_parsed, stream_type_set, object_datagram_created, object_datagram_parsed, subgroup_header_created, subgroup_header_parsed, subgroup_object_created, subgroup_object_parsed, fetch_header_created, fetch_header_parsed, fetch_object_created, fetch_object_parsed

Description: Event definitions related to the MOQT protocol.

Reference: This Document

8. Normative References

[CDDL] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[MOQT] Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-15, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-15>.

[QLOG-MAIN] Marx, R., Niccolini, L., Seemann, M., and L. Pardue, "qlog: Structured Logging for Network Protocols", Work in Progress, Internet-Draft, draft-ietf-quic-qlog-main-schema-13, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-main-schema-13>.

[QLOG-QUIC] Marx, R., Niccolini, L., Seemann, M., and L. Pardue, "QUIC event definitions for qlog", Work in Progress, Internet-Draft, draft-ietf-quic-qlog-quic-events-12, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-quic-events-12>.

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

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

Acknowledgments

Thanks to Lorenzo Miniero, Sujay Patel, and Aman Sharm for feedback and contributions to this document.

Authors' Addresses

Lucas Pardue
Cloudflare
Email: lucas@lucaspardue.com

Mathis Engelbart
Technical University of Munich
Email: mathis.engelbart@gmail.com

Aman Sharma
Meta
Email: amsharma@meta.com