

Media Over QUIC
Internet-Draft
Intended status: Informational
Expires: 7 February 2026

L. Pardue
Cloudflare
M. Engelbart
Technical University of Munich
6 August 2025

MoQ qlog event definitions
draft-pardue-moq-qlog-moq-events-02

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

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 7 February 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Usage with QUIC	5
2. Conventions and Definitions	5
3. Event Schema Definition	6
3.1. Draft Event Schema Identification	6
4. MOQT Events	6
4.1. control_message_created	7
4.2. control_message_parsed	7
4.3. stream_type_set	8
4.4. object_datagram_created	8
4.5. object_datagram_parsed	9
4.6. object_datagram_status_created	9
4.7. object_datagram_status_parsed	9
4.8. subgroup_header_created	10
4.9. subgroup_header_parsed	10
4.10. subgroup_object_created	11
4.11. subgroup_object_parsed	11
4.12. fetch_header_created	11
4.13. fetch_header_parsed	12
4.14. fetch_object_created	12
4.15. fetch_object_parsed	13
5. MOQT Data Type Definitions	13
5.1. Owner	13
5.2. MOQTSetupParameter	13
5.2.1. MOQTPathSetupParameter	14
5.2.2. MOQTMaxRequestIdSetupParameter	14
5.2.3. MOQTUnknownSetupParameter	14
5.3. MOQTParameter	14
5.3.1. MOQTAuthorizationTokenParameter	15
5.3.2. MOQTDeliveryTimeoutParameter	15

5.3.3.	MOQTMaxCacheDurationParameter	15
5.3.4.	MOQTUnknownParameter	15
5.4.	MOQTByteString	16
5.5.	MOQTLocation	16
5.6.	MOQTControlMessage	16
5.6.1.	MOQTClientSetupMessage	17
5.6.2.	MOQTServerSetupMessage	18
5.6.3.	MOQTGoaway	18
5.6.4.	MOQTMaxRequestId	18
5.6.5.	MOQTRequestsBlocked	18
5.6.6.	MOQTSubscribe	19
5.6.7.	MOQTSubscribeOk	19
5.6.8.	MOQTSubscribeError	19
5.6.9.	MOQTSubscribeUpdate	20
5.6.10.	MOQTUnsubscribe	20
5.6.11.	MOQTSubscribeDone	20
5.6.12.	MOQTPublish	20
5.6.13.	MOQTPublishOk	21
5.6.14.	MOQTPublishError	21
5.6.15.	MOQTFetch	21
5.6.16.	MOQTFetchOk	22
5.6.17.	MOQTFetchError	22
5.6.18.	MOQTFetchCancel	23
5.6.19.	MOQTTrackStatusRequest	23
5.6.20.	MOQTTrackStatus	23
5.6.21.	MOQTAnnounce	23
5.6.22.	MOQTAnnounceOk	24
5.6.23.	MOQTAnnounceError	24
5.6.24.	MOQTUnannounce	24
5.6.25.	MOQTAnnounceCancel	24
5.6.26.	MOQTSubscribeAnnounces	24
5.6.27.	MOQTSubscribeAnnouncesOk	25
5.6.28.	MOQTSubscribeAnnouncesError	25
5.6.29.	MOQTUnsubscribeAnnounces	25
5.7.	MOQTExtensionHeader	25
6.	Security Considerations	26
7.	IANA Considerations	26
8.	Normative References	26
	Acknowledgments	27
	Authors' Addresses	27

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
<code>moqt:control_message_created</code>	Core	Section 4.1
<code>moqt:control_message_parsed</code>	Core	Section 4.2
<code>moqt:stream_type_set</code>	Core	Section 4.3
<code>moqt:object_datagram_created</code>	Core	Section 4.4
<code>moqt:object_datagram_parsed</code>	Core	Section 4.5
<code>moqt:object_datagram_status_created</code>	Core	Section 4.6
<code>moqt:object_datagram_status_parsed</code>	Core	Section 4.7
<code>moqt:subgroup_header_created</code>	Core	Section 4.8
<code>moqt:subgroup_header_parsed</code>	Core	Section 4.9
<code>moqt:subgroup_object_created</code>	Core	Section 4.10
<code>moqt:subgroup_object_parsed</code>	Core	Section 4.11
<code>moqt:fetch_header_created</code>	Core	Section 4.12
<code>moqt:fetch_header_parsed</code>	Core	Section 4.13
<code>moqt:fetch_object_created</code>	Core	Section 4.14
<code>moqt:fetch_object_parsed</code>	Core	Section 4.15

Table 1: MOQT Events

When any event from this document is included in a qlog trace, the "protocol_types" qlog array field MUST contain an entry with the value "MOQT":

```
$ProtocolType /= "MOQT"
```

Figure 1: ProtocolType extension for MOQT

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 this 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, protocol_types, 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 /  
                MOQTObjectDatagramStatusCreated /  
                MOQTObjectDatagramStatusParsed /  
                MOQTSubgroupHeaderCreated /  
                MOQTSubgroupHeaderParsed /  
                MOQTSubgroupObjectCreated /  
                MOQTSubgroupObjectParsed /  
                MOQTFetchHeaderCreated /  
                MOQTFetchHeaderParsed /  
                MOQTFetchObjectCreated /  
                MOQTFetchObjectParsed
```

```
$ProtocolEventData /= MOQTEventData
```

Figure 2: 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 3: 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 4: 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 5: 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_payload: RawInfo

  * $$moqt-objectdatagramcreated-extension
}

```

Figure 6: 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: [* MOQTEExtensionHeader]  
    ? object_payload: RawInfo  
  
    * $$moqt-objectdatagramparsed-extension  
}
```

Figure 7: MOQTObjectDatagramParsed definition

4.6. object_datagram_status_created

The `object_datagram_status_created` event is emitted when the `OBJECT_DATAGRAM_STATUS` message is created. It has Core importance level; see Section 9.2 of [QLOG-MAIN].

```
MOQTObjectDatagramStatusCreated = {  
    track_alias: uint64  
    group_id: uint64  
    object_id: uint64  
    publisher_priority: uint8  
    extension_headers_length: uint64  
    ? extension_headers: [* MOQTEExtensionHeader]  
    object_status: uint64  
  
    * $$moqt-objectdatagramstatuscreated-extension  
}
```

Figure 8: MOQTObjectDatagramStatusCreated definition

4.7. object_datagram_status_parsed

The `object_datagram_status_parsed` event is emitted when the `OBJECT_DATAGRAM_STATUS` message is parsed. It has Core importance level; see Section 9.2 of [QLOG-MAIN].

```

MOQTObjectDatagramStatusParsed = {
    track_alias: uint64
    group_id: uint64
    object_id: uint64
    publisher_priority: uint8
    extension_headers_length: uint64
    ? extension_headers: [* MOQTEExtensionHeader]
    object_status: uint64

    * $$moqt-objectdatagramstatusparsed-extension
}

```

Figure 9: MOQTObjectDatagramStatusParsed definition

4.8. 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].

```

MOQTSubgroupHeaderCreated = {
    stream_id: uint64
    track_alias: uint64
    group_id: uint64
    ? subgroup_id: uint64
    publisher_priority: uint8

    * $$moqt-subgroupheadercreated-extension
}

```

Figure 10: MOQTSubgroupHeaderCreated definition

4.9. subgroup_header_parsed

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

```

MOQTSubgroupHeaderParsed = {
    stream_id: uint64
    track_alias: uint64
    group_id: uint64
    ? subgroup_id: uint64
    publisher_priority: uint8

    * $$moqt-subgroupheaderparsed-extension
}

```

Figure 11: MOQTSubgroupHeaderParsed definition

4.10. 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: [* MOQTEExtensionHeader]  
    object_payload_length: uint64  
    ? object_status: uint64  
    ? object_payload: RawInfo  
  
    * $$moqt-subgroupobjectcreated-extension  
}
```

Figure 12: MOQTSubgroupObjectCreated definition

4.11. 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: [* MOQTEExtensionHeader]  
    object_payload_length: uint64  
    ? object_status: uint64  
    ? object_payload: RawInfo  
  
    * $$moqt-subgroupobjectparsed-extension  
}
```

Figure 13: MOQTSubgroupObjectParsed definition

4.12. 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 14: MOQTFetchHeaderCreated definition

4.13. 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 15: MOQTFetchHeaderParsed definition

4.14. 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: [* MOQTEExtensionHeader]  
    object_payload_length: uint64  
    ? object_status: uint64  
    ? object_payload: RawInfo  
  
    * $$moqt-fetchobjectcreated-extension  
}
```

Figure 16: MOQTFetchObjectCreated definition

4.15. 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: [* MOQTEExtensionHeader]
  object_payload_length: uint64
  ? object_status: uint64
  ? object_payload: RawInfo

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

Figure 17: 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 18: 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 19: MOQTSetupParameter type socket definition

```

MOQTBaseSetupParameters /= MOQTPathSetupParameter /
                           MOQTMaxRequestIdSetupParameter /
                           MOQTUnknownSetupParameter

$MOQTSetupParameter /= MOQTBaseSetupParameters

```

Figure 20: MOQTBaseSetupParameters definition

5.2.1. MOQTPathSetupParameter

```

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

```

Figure 21: MOQTPathSetupParameter definition

5.2.2. MOQTMaxRequestIdSetupParameter

```

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

```

Figure 22: MOQTMaxRequestIdSetupParameter definition

5.2.3. MOQTUnknownSetupParameter

```

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

```

Figure 23: 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 24: MOQTParameter type socket definition

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

Figure 25: MOQTBaseParameters definition

5.3.1. MOQTAuthorizationTokenParameter

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

Figure 26: MOQTAuthorizationTokenParameter definition

5.3.2. MOQTDeliveryTimeoutParameter

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

Figure 27: MOQTDeliveryTimeoutParameter definition

5.3.3. MOQTMaxCacheDurationParameter

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

Figure 28: MOQTMaxCacheDurationParameter definition

5.3.4. MOQTUnknownParameter

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

Figure 29: 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 30: MOQTByteString definition

5.5. MOQTLocation

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

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

Figure 31

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 32: 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 /
                           MOQTSubscribeDone /
                           MOQTPublish /
                           MOQTPublishOk /
                           MOQTPublishError /
                           MOQTFetch /
                           MOQTFetchOk /
                           MOQTFetchError /
                           MOQTFetchCancel /
                           MOQTTrackStatusRequest /
                           MOQTTrackStatus /
                           MOQTAnnounce /
                           MOQTAnnounceOk /
                           MOQTAnnounceError /
                           MOQTUnannounce /
                           MOQTAnnounceCancel /
                           MOQTSubscribeAnnounces /
                           MOQTSubscribeAnnouncesOk /
                           MOQTSubscribeAnnouncesError /
                           MOQTUnsubscribeAnnounces

$MOQTControlMessage /= MOQTBaseControlMessages

```

Figure 33: MOQTBaseControlMessages definition

5.6.1. MOQTClientSetupMessage

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

Figure 34: MOQTClientSetupMessage definition

5.6.2. MOQTSERVERSetupMessage

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

Figure 35: MOQTSERVERSetupMessage definition

5.6.3. MOQTGoaway

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

Figure 36: MOQTGoaway definition

5.6.4. MOQTMaxRequestId

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

Figure 37: MOQTMaxRequestId definition

5.6.5. MOQTRequestsBlocked

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

Figure 38: 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  
  ? subscribe_parameters: [ * $MOQTParameter]  
}
```

Figure 39: 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  
  ? subscribe_parameters: [ * $MOQTParameter]  
}
```

Figure 40: MOQTSubscribeOk definition

5.6.8. MOQTSubscribeError

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

Figure 41: MOQTSubscribeError definition

5.6.9. MOQTSubscribeUpdate

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

Figure 42: MOQTSubscribeUpdate definition

5.6.10. MOQTUnsubscribe

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

Figure 43: MOQTUnsubscribe definition

5.6.11. MOQTSubscribeDone

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

Figure 44: MOQTSubscribeDone 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 45: 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 46: MOQTPublishOk definition

5.6.14. MQTTPublishError

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

Figure 47: MQTTPublishError definition

5.6.15. MQTTFetch

```
MOQTFetch = {  
  type: "fetch"  
  request_id: uint64  
  subscriber_priority: uint8  
  group_order: uint8  
  fetch_type: uint64  
  
  track_namespace: [ *MOQTByteString]  
  ? track_name: MOQTByteString  
  ? start_location: MOQTLocation  
  ? end_location: MOQTLocation  
  
  ? joining_request_id: uint64  
  ? preceding_group_offset: uint64  
  
  number_of_parameters: uint64  
  ? parameters: [ * $MOQTParameter]  
}
```

Figure 48: 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  
  ? subscribe_parameters: [ * $MOQTParameter]  
}
```

Figure 49: MOQTFetchOk definition

5.6.17. MOQTFetchError

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

Figure 50: MOQTFetchError definition

5.6.18. MOQTFetchCancel

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

Figure 51: MOQTFetchCancel definition

5.6.19. MOQTTrackStatusRequest

```
MOQTTrackStatusRequest = {  
  type: "track_status_request"  
  request_id: uint64  
  track_namespace: [ *MOQTByteString]  
  track_name: MOQTByteString  
  number_of_parameters: uint64  
  ? parameters: [ * $MOQTParameter]  
}
```

Figure 52: MOQTTrackStatusRequest definition

5.6.20. MOQTTrackStatus

```
MOQTTrackStatus = {  
  type: "track_status"  
  request_id: uint64  
  status_code: uint64  
  last_group_id: uint64  
  last_object_id: uint64  
  number_of_parameters: uint64  
  ? parameters: [ * $MOQTParameter]  
}
```

Figure 53: MOQTTrackStatus definition

5.6.21. MOQTAnnounce

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

Figure 54: MOQTAnnounce definition

5.6.22. MOQTAnnounceOk

```
MOQTAnnounceOk = {  
  type: "announce_ok"  
  request_id: uint64  
}
```

Figure 55: MOQTAnnounceOk definition

5.6.23. MOQTAnnounceError

```
MOQTAnnounceError = {  
  type: "announce_error"  
  request_id: uint64  
  error_code: uint64  
  ? reason: text  
  ? reason_bytes: hexstring  
}
```

Figure 56: MOQTAnnounceError definition

5.6.24. MOQTUnannounce

```
MOQTUnannounce = {  
  type: "unannounce"  
  track_namespace: [ *MOQTByteString]  
}
```

Figure 57: MOQTUnannounce definition

5.6.25. MOQTAnnounceCancel

```
MOQTAnnounceCancel = {  
  type: "announce_cancel"  
  request_id: uint64  
  error_code: uint64  
  ? reason: text  
  ? reason_bytes: hexstring  
}
```

Figure 58: MOQTAnnounceCancel definition

5.6.26. MOQTSubscribeAnnounces


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

Figure 59: MOQTSubscribeAnnounces definition

5.6.27. MOQTSubscribeAnnouncesOk

```
MOQTSubscribeAnnouncesOk = {  
  type: "subscribe_announces_ok"  
  request_id: uint64  
}
```

Figure 60: MOQTSubscribeAnnouncesOk definition

5.6.28. MOQTSubscribeAnnouncesError

```
MOQTSubscribeAnnouncesError = {  
  type: "subscribe_announces_error"  
  request_id: uint64  
  error_code: uint64  
  ? reason: text  
  ? reason_bytes: hexstring  
}
```

Figure 61: MOQTSubscribeAnnouncesError definition

5.6.29. MOQTUnsubscribeAnnounces

```
MOQTUnsubscribeAnnounces = {  
  type: "unsubscribe_announces"  
  track_namespace_prefix: [ *MOQTByteString]  
}
```

Figure 62: MOQTUnsubscribeAnnounces definition

5.7. MOQTExtensionHeader

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

Figure 63: 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, object_datagram_status_created, object_datagram_status_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-13, 7 July 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-13>>.

[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-12, 7 July 2025,
<<https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-main-schema-12>>.

[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-11, 7 July 2025,
<<https://datatracker.ietf.org/doc/html/draft-ietf-quic-qlog-quic-events-11>>.

[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 Sujay Patel for contributions to this document.

Authors' Addresses

Lucas Pardue
Cloudflare
Email: lucas@lucaspardue.com

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