

Media Over QUIC
Internet-Draft
Intended status: Informational
Expires: 17 September 2026

L. Pardue
Cloudflare
M. Engelbart
Technical University of Munich
A. Sharma
Meta
16 March 2026

MoQ qlog event definitions
draft-pardue-moq-qlog-moq-events-06

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 17 September 2026.

Copyright Notice

Copyright (c) 2026 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	4
1.1. Usage with QUIC	5
2. Conventions and Definitions	6
3. Event Schema Definition	6
3.1. Draft Event Schema Identification	6
4. MOQT Events	7
4.1. control_message_created	7
4.2. control_message_parsed	8
4.3. stream_type_set	8
4.4. object_datagram_created	9
4.5. object_datagram_parsed	9
4.6. subgroup_header_created	10
4.7. subgroup_header_parsed	11
4.8. subgroup_object_created	11
4.9. subgroup_object_parsed	12
4.10. fetch_header_created	12
4.11. fetch_header_parsed	12
4.12. fetch_object_created	13
4.13. fetch_object_parsed	13
5. MOQT Data Type Definitions	14
5.1. Owner	14
5.2. MOQTLocation	14
5.3. MOQTSubscriptionFilter	14
5.4. MOQTSetupParameter	15
5.4.1. MOQTAuthoritySetupParameter	15

5.4.2.	MOQTPathSetupParameter	15
5.4.3.	MOQTMaxRequestIdSetupParameter	16
5.4.4.	MOQTMaxAuthTokenCacheSizeSetupParameter	16
5.4.5.	MOQTAuthorizationTokenSetupParameter	16
5.4.6.	MOQTImplementationSetupParameter	16
5.4.7.	MOQTUnknownSetupParameter	16
5.5.	MOQTParameter	17
5.5.1.	MOQTAuthorizationTokenParameter	17
5.5.2.	MOQTDeliveryTimeoutParameter	18
5.5.3.	MOQTMaxCacheDurationParameter	18
5.5.4.	MOQTPublisherPriorityParameter	18
5.5.5.	MOQTSubscriberPriorityParameter	18
5.5.6.	MOQTGroupOrderParameter	18
5.5.7.	MOQTSubscriptionFilterParameter	19
5.5.8.	MOQTExpiresParameter	19
5.5.9.	MOQTLargestObjectParameter	19
5.5.10.	MOQTForwardParameter	19
5.5.11.	MOQTDynamicGroupsParameter	19
5.5.12.	MOQTNewGroupRequestParameter	20
5.5.13.	MOQTUnknownParameter	20
5.6.	MOQTByteString	20
5.7.	MOQTControlMessage	21
5.7.1.	MOQTClientSetupMessage	21
5.7.2.	MOQTServerSetupMessage	22
5.7.3.	MOQTGoaway	22
5.7.4.	MOQTMaxRequestId	22
5.7.5.	MOQTRequestsBlocked	22
5.7.6.	MOQTRequestOk	22
5.7.7.	MOQTRequestError	23
5.7.8.	MOQTSubscribe	23
5.7.9.	MOQTSubscribeOk	23
5.7.10.	MOQTRequestUpdate	24
5.7.11.	MOQTUnsubscribe	24
5.7.12.	MOQTPublish	24
5.7.13.	MOQTPublishOk	24
5.7.14.	MOQTPublishDone	25
5.7.15.	MOQTFetch	25
5.7.16.	MOQTFetchOk	25
5.7.17.	MOQTFetchCancel	26
5.7.18.	MOQTTrackStatus	26
5.7.19.	MOQTPublishNamespace	26
5.7.20.	MOQTNamespace	26
5.7.21.	MOQTPublishNamespaceDone	27
5.7.22.	MOQTNamespaceDone	27
5.7.23.	MOQTPublishNamespaceCancel	27
5.7.24.	MOQTSubscribeNamespace	27
5.8.	MOQTExtensionHeader	28
6.	Security Considerations	28

7. IANA Considerations	28
8. Normative References	29
Acknowledgments	29
Authors' Addresses	29

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 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, 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 5.7.

```
MOQTControlMessageCreated = {  
    stream_id: uint64  
    ? length: uint16  
    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 5.7.

```
MOQTControlMessageParsed = {  
    stream_id: uint64  
    ? length: uint16  
    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" /  
    "subscribe_namespace"
```


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.

If the `publisher_priority` is absent, it means that the object inherits the Publisher Priority specified in the control message that established the subscription.

```
MOQTObjectDatagramCreated = {  
    track_alias: uint64  
    group_id: uint64  
    ? object_id: uint64  
    ? publisher_priority: uint8  
    ? extension_headers_length: uint64  
    ? extension_headers: [* MOQTEExtensionHeader]  
    ? 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.

If the `publisher_priority` is absent, it means that the object inherits the Publisher Priority specified in the control message that established the subscription.

```

MOQTObjectDatagramParsed = {
    track_alias: uint64
    group_id: uint64
    ? object_id: uint64
    ? publisher_priority: uint8
    ? extension_headers_length: uint64
    ? extension_headers: [* MOQTEExtensionHeader]
    ? 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 5 bits in the type value to convey various properties. The `subgroup_header_created` event conveys these as explicit fields, such as `contains_end_of_group`.

If the `publisher_priority` is absent, it means that the subgroup inherits the Publisher Priority specified in the control message that established the subscription.

```

MOQTSubgroupHeaderCreated = {
    stream_id: uint64
    track_alias: uint64
    group_id: uint64
    subgroup_id_mode: uint8
    ? 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 5 bits in the type value to convey various properties. The `subgroup_header_created` event conveys these as explicit fields, such as `contains_end_of_group`.

If the `publisher_priority` is absent, it means that the subgroup inherits the Publisher Priority specified in the control message that established the subscription.

```
MOQTSubgroupHeaderParsed = {
    stream_id: uint64
    track_alias: uint64
    group_id: uint64
    subgroup_id_mode: uint8
    ? 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
    object_id_delta: 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  
  object_id_delta: uint64  
  ? extension_headers: [* MOQTEExtensionHeader]  
  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.

The `FETCH` object in MoQT uses serialization flags to encode various properties. The event conveys these as explicit fields, such as `end_of_unknown_range`.

```
MOQTFetchObjectCreated = {  
    stream_id: uint64  
    datagram: bool  
    end_of_nonexistent_range: bool  
    end_of_unknown_range: bool  
    ? subgroup_id_bits: uint8  
    ? 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 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.

The `FETCH` object in MoQT uses serialization flags to encode various properties. The event conveys these as explicit fields, such as `end_of_unknown_range`.

```
MOQTFetchObjectParsed = {  
    stream_id: uint64  
    datagram: bool  
    end_of_nonexistent_range: bool  
    end_of_unknown_range: bool  
    ? subgroup_id_bits: uint8  
    ? 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 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. MOQTLocation

A Location, as defined in Section 1.4.1 of [MOQT]

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

Figure 16: MOQTLocation definition

5.3. MOQTSubscriptionFilter

A Subscription Filter, as defined in Section 5.1.2 of [MOQT]

```

MOQTSubscriptionFilter = {
  filter_type: uint64
  ? start_location: MOQTLocation
  ? end_group: uint64
}

```

Figure 17: MOQTSubscriptionFilter definition

5.4. MOQTSSetupParameter

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

```

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

```

Figure 18: MOQTSSetupParameter type socket definition

```

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

$MOQTSSetupParameter /= MOQTBaseSetupParameters

```

Figure 19: MOQTBaseSetupParameters definition

5.4.1. MOQTAuthoritySetupParameter

```

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

```

Figure 20: MOQTAuthoritySetupParameter definition

5.4.2. MOQTPathSetupParameter

```

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

```

Figure 21: MOQTPathSetupParameter definition

5.4.3. MOQTMaXRequestIdSetupParameter

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

Figure 22: MOQTMaXRequestIdSetupParameter definition

5.4.4. MOQTMaXAuthTokenCacheSizeSetupParameter

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

Figure 23: MOQTMaXAuthTokenCacheSizeSetupParameter definition

5.4.5. 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 24: MOQTAuthorizationTokenSetupParameter definition

5.4.6. MOQTImplementationSetupParameter

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

Figure 25: MOQTImplementationSetupParameter definition

5.4.7. MOQTUnknownSetupParameter


```

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

```

Figure 26: MOQTUnknownSetupParameter definition

5.5. 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 27: MOQTParameter type socket definition

```

MOQTBaseParameters /= MOQTAuthorizationTokenParameter /
MOQTDeliveryTimeoutParameter /
MOQTMaxCacheDurationParameter /
MOQTPublisherPriorityParameter /
MOQTSubscriberPriorityParameter /
MOQTGroupOrderParameter /
MOQTSubscriptionFilterParameter /
MOQTExpiresParameter /
MOQTLargestObjectParameter /
MOQTForwardParameter /
MOQTDynamicGroupsParameter /
MOQTNewGroupRequestParameter /
MOQTUnknownParameter

```

```

$MOQTParameter /= MOQTBaseParameters

```

Figure 28: MOQTBaseParameters definition

5.5.1. MOQTAuthorizationTokenParameter

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

Figure 29: MOQTAuthorizationTokenParameter definition

5.5.2. MQQTDeliveryTimeoutParameter

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

Figure 30: MQQTDeliveryTimeoutParameter definition

5.5.3. MQQTMaxCacheDurationParameter

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

Figure 31: MQQTMaxCacheDurationParameter definition

5.5.4. MQQTPublisherPriorityParameter

```
MQQTPublisherPriorityParameter = {  
  name: "publisher_priority"  
  value: uint64  
}
```

Figure 32: MQQTPublisherPriorityParameter definition

5.5.5. MQQTSubscriberPriorityParameter

```
MQQTSubscriberPriorityParameter = {  
  name: "subscriber_priority"  
  value: uint64  
}
```

Figure 33: MQQTSubscriberPriorityParameter definition

5.5.6. MQQTGroupOrderParameter

```
MOQTGroupOrderParameter = {  
  name: "group_order"  
  value: uint64  
}
```

Figure 34: MOQTGroupOrderParameter definition

5.5.7. MOQTSubscriptionFilterParameter

```
MOQTSubscriptionFilterParameter = {  
  name: "subscription_filter"  
  value: MOQTSubscriptionFilter  
}
```

Figure 35: MOQTSubscriptionFilterParameter definition

5.5.8. MOQTExpiresParameter

```
MOQTExpiresParameter = {  
  name: "expires"  
  value: uint64  
}
```

Figure 36: MOQTExpiresParameter definition

5.5.9. MOQTLargestObjectParameter

```
MOQTLargestObjectParameter = {  
  name: "largest_object"  
  value: MOQTLocation  
}
```

Figure 37: MOQTLargestObjectParameter definition

5.5.10. MOQTForwardParameter

```
MOQTForwardParameter = {  
  name: "forward"  
  value: uint64  
}
```

Figure 38: MOQTForwardParameter definition

5.5.11. MOQTDynamicGroupsParameter

```
MOQTDynamicGroupsParameter = {  
  name: "dynamic_groups"  
  value: uint64  
}
```

Figure 39: MOQTDynamicGroupsParameter definition

5.5.12. MOQTNewGroupRequestParameter

```
MOQTNewGroupRequestParameter = {  
  name: "new_group_request"  
  value: uint64  
}
```

Figure 40: MOQTNewGroupRequestParameter definition

5.5.13. MOQTUnknownParameter

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

Figure 41: MOQTUnknownParameter definition

5.6. 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 42: MOQTByteString definition

5.7. 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 43: MOQTControlMessage type socket definition

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

```
MOQTBaseControlMessages = MOQTClientSetupMessage /
                           MOQTServerSetupMessage /
                           MOQTGoaway /
                           MOQTMaxRequestId /
                           MOQTRequestsBlocked /
                           MOQTRequestOk /
                           MOQTRequestError /
                           MOQTSubscribe /
                           MOQTSubscribeOk /
                           MOQTRequestUpdate /
                           MOQTUnsubscribe /
                           MOQTPublish /
                           MOQTPublishOk /
                           MOQTPublishDone /
                           MOQTFetch /
                           MOQTFetchOk /
                           MOQTFetchCancel /
                           MOQTTrackStatus /
                           MOQTPublishNamespace /
                           MOQTNamespace /
                           MOQTPublishNamespaceDone /
                           MOQTNamespaceDone /
                           MOQTPublishNamespaceCancel /
                           MOQTSubscribeNamespace

$MOQTControlMessage /= MOQTBaseControlMessages
```

Figure 44: MOQTBaseControlMessages definition

5.7.1. MOQTClientSetupMessage

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

Figure 45: MOQTClientSetupMessage definition

5.7.2. MOQTServerSetupMessage

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

Figure 46: MOQTServerSetupMessage definition

5.7.3. MOQTGoaway

```
MOQTGoaway = {  
  type: "goaway"  
  new_session_uri: RawInfo  
}
```

Figure 47: MOQTGoaway definition

5.7.4. MOQTMaxRequestId

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

Figure 48: MOQTMaxRequestId definition

5.7.5. MOQTRequestsBlocked

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

Figure 49: MOQTRequestsBlocked definition

5.7.6. MOQTRequestOk

```
MOQTRequestOk = {  
  type: "request_ok"  
  request_id: uint64  
  number_of_parameters: uint64  
  ? parameters: [* $MOQTParameter]  
}
```

Figure 50: MOQTRequestOk definition

5.7.7. MOQTRequestError

```
MOQTRequestError = {  
  type: "request_error"  
  request_id: uint64  
  error_code: uint64  
  retry_interval: uint64  
  ? reason: text  
  ? reason_bytes: hexstring  
}
```

Figure 51: MOQTRequestError definition

5.7.8. MOQTSubscribe

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

Figure 52: MOQTSubscribe definition

5.7.9. MOQTSubscribeOk

```
MOQTSubscribeOk = {  
  type: "subscribe_ok"  
  request_id: uint64  
  track_alias: uint64  
  number_of_parameters: uint64  
  ? parameters: [* $MOQTParameter]  
  ? track_extensions: [* MOQTExtensionHeader]  
}
```

Figure 53: MOQTSubscribeOk definition

5.7.10. MOQTRequestUpdate

```
MOQTRequestUpdate = {  
  type: "request_update"  
  request_id: uint64  
  existing_request_id: uint64  
  number_of_parameters: uint64  
  ? parameters: [* $MOQTPParameter]  
}
```

Figure 54: MOQTRequestUpdate definition

5.7.11. MOQTUnsubscribe

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

Figure 55: MOQTUnsubscribe definition

5.7.12. MOQTPublish

```
MOQTPublish = {  
  type: "publish"  
  request_id: uint64  
  track_namespace: [ *MOQTByteString]  
  track_name: MOQTByteString  
  track_alias: uint64  
  number_of_parameters: uint64  
  ? parameters: [* $MOQTPParameter]  
  ? track_extensions: [* MOQTEExtensionHeader]  
}
```

Figure 56: MOQTPublish definition

5.7.13. MOQTPublishOk

```
MOQTPublishOk = {  
  type: "publish_ok"  
  request_id: uint64  
  number_of_parameters: uint64  
  ? parameters: [* $MOQTPParameter]  
}
```

Figure 57: MOQTPublishOk definition

5.7.14. MOQTPublishDone

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

Figure 58: MOQTPublishDone definition

5.7.15. MOQTFetch

```
MOQTFetch = {  
  type: "fetch"  
  request_id: uint64  
  fetch_type: $MOQTFetchType  
  ? standalone_fetch: $MOQTStandaloneFetch  
  ? joining_fetch: $MOQTJoiningFetch  
  number_of_parameters: uint64  
  ? 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 59: MOQTFetch definition

5.7.16. MOQTFetchOk

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

Figure 60: MOQTFetchOk definition

5.7.17. MOQTFetchCancel

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

Figure 61: MOQTFetchCancel definition

5.7.18. MOQTTrackStatus

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

Figure 62: MOQTTrackStatus definition

5.7.19. MOQTPublishNamespace

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

Figure 63: MOQTPublishNamespace definition

5.7.20. MOQTNamespace

```
MOQTNamespace = {  
  type: "namespace"  
  track_namespace_suffix: [ *MOQTByteString]  
}
```

Figure 64: MOQTNamespace definition

5.7.21. MQTTPublishNamespaceDone

```
MQTTPublishNamespaceDone = {  
  type: "publish_namespace_done"  
  request_id: uint64  
}
```

Figure 65: MQTTPublishNamespaceDone definition

5.7.22. MOQTNamespaceDone

```
MOQTNamespaceDone = {  
  type: "namespace_done"  
  track_namespace_suffix: [ *MOQTByteString]  
}
```

Figure 66: MOQTNamespaceDone definition

5.7.23. MQTTPublishNamespaceCancel

```
MQTTPublishNamespaceCancel = {  
  type: "publish_namespace_cancel"  
  request_id: uint64  
  error_code: uint64  
  ? reason: text  
  ? reason_bytes: hexstring  
}
```

Figure 67: MQTTPublishNamespaceCancel definition

5.7.24. MQQTSubscribeNamespace

```

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

$MOQTSubscribeOptions /= "publish" /
                        "namespace" /
                        "both"

```

Figure 68: MOQTSubscribeNamespace definition

5.8. MOQTExtensionHeader

```

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

```

Figure 69: 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-17, 2 March 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-17>>.
- [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