QUIC
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

R. Marx, Ed.
Akamai
L. Niccolini, Ed.
Meta
M. Seemann, Ed.

L. Pardue, Ed.
Cloudflare

20 October 2025

HTTP/3 qlog event definitions
draft-ietf-quic-qlog-h3-events-12

Abstract

This document defines a qlog event schema containing concrete events for the core HTTP/3 protocol and selected extensions.

Note to Readers

Note to RFC editor: Please remove this section before publication.

Feedback and discussion are welcome at https://github.com/quicwg/qlog (https://github.com/quicwg/qlog). Readers are advised to refer to the "editor's draft" at that URL for an up-to-date version of this document.

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 23 April 2026.

Table of Contents

1. Introduction

This document defines a qlog event schema (Section 8 of [QLOG-MAIN]) containing concrete events for the core HTTP/3 protocol [HTTP/3] and selected extensions ([EXTENDED-CONNECT], [H3_PRIORITIZATION], and [H3-DATAGRAM]).

The event namespace with identifier http3 is defined; see Section 2. 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
http3:parameters_set	Base	Section 3.1
http3:parameters_restored	Base	Section 3.2
http3:stream_type_set	Base	Section 3.3
http3:priority_updated	Base	Section 3.4
http3:frame_created	Core	Section 3.5
http3:frame_parsed	Core	Section 3.6
http3:datagram_created	Base	Section 3.7
http3:datagram_parsed	Base	Section 3.8
http3:push_resolved	Extra	Section 3.9

Table 1: HTTP/3 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 HTTP/3 connection. 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).

1.2. Notational Conventions

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.

2. Event Schema Definition

This document describes how the core HTTP/3 protocol and selected extensions can be expressed in qlog using a newly defined event schema. Per the requirements in Section 8 of [QLOG-MAIN], this document registers the http3 namespace. The event schema URI is urn:ietf:params:qlog:events:http3.

2.1. 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:http3. 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 07 of this document is identified using the URI urn:ietf:params:qlog:events:http3-07.

The namespace identifier itself is not affected by this requirement.

3. HTTP/3 Events

HTTP/3 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].

```
HTTP3EventData = HTTP3ParametersSet /
                 HTTP3ParametersRestored /
                 HTTP3StreamTypeSet /
                 HTTP3PriorityUpdated /
                 HTTP3FrameCreated /
                 HTTP3FrameParsed /
                 HTTP3DatagramCreated /
                 HTTP3DatagramParsed /
                 HTTP3PushResolved

$ProtocolEventData /= HTTP3EventData
```

Figure 1: HTTP3EventData definition and ProtocolEventData extension

HTTP 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].

HTTP/3 frames are transmitted on QUIC streams, which allows them to span multiple QUIC packets. Some implementations might send a single large frame, rather than a sequence of smaller frames, in order to amortize frame header overhead. HTTP/3 frame headers are represented by the frame_created (Section 3.5) and frame_parsed (Section 3.6) events. Subsequent frame payload data transfer is indicated by stream_data_moved events. Furthermore, stream_data_moved events can appear before frame_parsed events because implementations need to read data from a stream in order to parse the frame header.

The concrete HTTP/3 event types are further defined below, their type identifier is the heading name.

3.1. parameters_set

The parameters_set event contains HTTP/3 and QPACK-level settings, mostly those received from the HTTP/3 SETTINGS frame. It has Base importance level.

All these parameters are typically set once and never change. However, they might be set at different times during the connection, therefore a qlog can have multiple instances of parameters_set with different fields set.

The "initiator" field reflects how Settings are exchanged on a connection. Sent settings have the value "local" and received settings have the value "received".

```
HTTP3ParametersSet = {
    ? initiator: Initiator

    ; RFC9114
    ? max_field_section_size: uint64

    ; RFC9204
    ? max_table_capacity: uint64
    ? blocked_streams_count: uint64

    ; RFC9220 (SETTINGS_ENABLE_CONNECT_PROTOCOL)
    ? extended_connect: uint16

    ; RFC9297 (SETTINGS_H3_DATAGRAM)
    ? h3_datagram: uint16

    ; qlog-specific
    ; indicates whether this implementation waits for a SETTINGS
    ; frame before processing requests
    ? waits_for_settings: bool

    * $$http3-parametersset-extension
}
```

Figure 2: HTTP3ParametersSet definition

The parameters_set event can contain any number of unspecified fields. This allows for representation of reserved settings (aka GREASE) or ad-hoc support for extension settings that do not have a related qlog schema definition.

3.2. parameters_restored

When using QUIC 0-RTT, HTTP/3 clients are expected to remember and reuse the server's SETTINGs from the previous connection. The parameters_restored event is used to indicate which HTTP/3 settings were restored and to which values when utilizing 0-RTT. It has Base importance level.

```
HTTP3ParametersRestored = {
    ; RFC9114
    ? max_field_section_size: uint64

    ; RFC9204
    ? max_table_capacity: uint64
    ? blocked_streams_count: uint64

    ; RFC9220 (SETTINGS_ENABLE_CONNECT_PROTOCOL)
    ? extended_connect: uint16

    ; RFC9297 (SETTINGS_H3_DATAGRAM)
    ? h3_datagram: uint16

    * $$http3-parametersrestored-extension
}
```

Figure 3: HTTP3ParametersRestored definition

3.3. stream_type_set

The stream_type_set event conveys when a HTTP/3 stream type becomes known; see Sections 6.1 and 6.2 of [HTTP/3]. It has Base importance level.

Client bidirectional streams always have a stream_type value of "request". Server bidirectional streams have no defined use, although extensions could change that.

Unidirectional streams in either direction begin with with a variable-length integer type. Where the type is not known, the stream_type value of "unknown" type can be used and the value captured in the stream_type_bytes field; a numerical value without variable-length integer encoding.

The generic $HTTP3StreamType is defined here as a CDDL "type socket" extension point. It can be extended to support additional HTTP/3 stream types.

```
HTTP3StreamTypeSet = {
    ? initiator: Initiator
    stream_id: uint64
    stream_type: $HTTP3StreamType

    ; only when stream_type === "unknown"
    ? stream_type_bytes: uint64

    ; only when stream_type === "push"
    ? associated_push_id: uint64

    * $$http3-streamtypeset-extension
}

$HTTP3StreamType /=   "request" /
                      "control" /
                      "push" /
                      "reserved" /
                      "unknown" /
                      "qpack_encode" /
                      "qpack_decode"
```

Figure 4: HTTP3StreamTypeSet definition

3.4. priority_updated

The priority_updated event is emitted when the priority of a request stream or push stream is initialized or updated through mechanisms defined in [RFC9218]. It has Base importance level.

There can be several reasons why a priority_updated occurs, and why a particular value was chosen. For example, the priority can be updated through signals received from client and/or server (e.g., in HTTP/3 HEADERS or PRIORITY_UPDATE frames) or it can be changed or overridden due to local policies. The trigger and reason fields can be used to optionally capture such details.

```
HTTP3PriorityUpdated = {
    ; if the prioritized element is a request stream
    ? stream_id: uint64

    ; if the prioritized element is a push stream
    ? push_id: uint64

    ? old: HTTP3Priority
    new: HTTP3Priority

    ? trigger: "client_signal_received" /
               "local" /
               "other"

    ? reason: "client_signal_only" /
              "client_server_merged" /
              "local_policy" /
              "other"

    * $$http3-priorityupdated-extension
}
```

Figure 5: HTTP3PriorityUpdated definition

3.5. frame_created

The frame_created event is emitted when the HTTP/3 framing actually happens. It has Core importance level.

This event does not necessarily coincide with HTTP/3 data getting passed to the QUIC layer. For that, see the stream_data_moved event in [QLOG-QUIC].

```
HTTP3FrameCreated = {
    stream_id: uint64
    frame: $HTTP3Frame

    * $$http3-framecreated-extension
}
```

Figure 6: HTTP3FrameCreated definition

3.6. frame_parsed

The frame_parsed event is emitted when the HTTP/3 frame is parsed. It has Core importance level.

This event is not necessarily the same as when the HTTP/3 data is actually received on the QUIC layer. For that, see the stream_data_moved event in [QLOG-QUIC].

```
HTTP3FrameParsed = {
    stream_id: uint64
    frame: $HTTP3Frame

    * $$h3-frameparsed-extension
}
```

Figure 7: HTTP3FrameParsed definition

3.7. datagram_created

The datagram_created event is emitted when an HTTP/3 Datagram is created (see [RFC9297]). It has Base importance level.

This event does not necessarily coincide with the HTTP/3 Datagram getting passed to the QUIC layer. For that, see the datagram_data_moved event in [QLOG-QUIC].

```
HTTP3DatagramCreated = {
    quarter_stream_id: uint64
    ? datagram: $HTTP3Datagram
    ? raw: RawInfo

    * $$http3-datagramcreated-extension
}
```

Figure 8: HTTP3DatagramCreated definition

3.8. datagram_parsed

The datagram_parsed event is emitted when the HTTP/3 Datagram is parsed (see [RFC9297]). It has Base importance level.

This event is not necessarily the same as when the HTTP/3 Datagram is actually received on the QUIC layer. For that, see the datagram_data_moved event in [QLOG-QUIC].

```
HTTP3DatagramParsed = {
    quarter_stream_id: uint64
    ? datagram: $HTTP3Datagram
    ? raw: RawInfo

    * $$http3-datagramparsed-extension
}
```

Figure 9: HTTP3DatagramParsed definition

3.9. push_resolved

The push_resolved event is emitted when a pushed resource (Section 4.6 of [HTTP/3]) is successfully claimed (used) or, conversely, abandoned (rejected) by the application on top of HTTP/3 (e.g., the web browser). This event provides additional context that can is aid debugging issues related to server push. It has Extra importance level.

```
HTTP3PushResolved = {
    ? push_id: uint64

    ; in case this is logged from a place that does not have access
    ; to the push_id
    ? stream_id: uint64
    decision: HTTP3PushDecision

    * $$http3-pushresolved-extension
}

HTTP3PushDecision = "claimed" /
                    "abandoned"
```

Figure 10: HTTP3PushResolved definition

4. HTTP/3 Data Type Definitions

The following data type definitions can be used in HTTP/3 events.

4.1. Initiator

```
Initiator = "local" /
            "remote"
```

Figure 11: Initiator definition

4.2. HTTP3Frame

The generic $HTTP3Frame is defined here as a CDDL "type socket" extension point. It can be extended to support additional HTTP/3 frame types.

```
; The HTTP3Frame is any key-value map (e.g., JSON object)
$HTTP3Frame /= {
    * text => any
}
```

Figure 12: HTTP3Frame type socket definition

The HTTP/3 frame types defined in this document are as follows:

```
HTTP3BaseFrames = HTTP3DataFrame /
                  HTTP3HeadersFrame /
                  HTTP3CancelPushFrame /
                  HTTP3SettingsFrame /
                  HTTP3PushPromiseFrame /
                  HTTP3GoawayFrame /
                  HTTP3MaxPushIDFrame /
                  HTTP3ReservedFrame /
                  HTTP3UnknownFrame

$HTTP3Frame /= HTTP3BaseFrames
```

Figure 13: HTTP3BaseFrames definition

4.3. HTTP3Datagram

The generic $HTTP3Datagram is defined here as a CDDL "type socket" extension point. It can be extended to support additional HTTP/3 datagram types. This document intentionally does not define any specific qlog schemas for specific HTTP/3 Datagram types.

```
; The HTTP3Datagram is any key-value map (e.g., JSON object)
$HTTP3Datagram /= {
    * text => any
}
```

Figure 14: HTTP3Datagram type socket definition

4.3.1. HTTP3DataFrame

```
HTTP3DataFrame = {
    frame_type: "data"
    ? raw: RawInfo
}
```

Figure 15: HTTP3DataFrame definition

4.3.2. HTTP3HeadersFrame

The payload of an HTTP/3 HEADERS frame is the QPACK-encoding of an HTTP field section; see Section 7.2.2 of [HTTP/3]. HTTP3HeaderFrame, in contrast, contains the HTTP field section without QPACK encoding.

```
HTTP3HTTPField = {
    ? name: text
    ? name_bytes: hexstring
    ? value: text
    ? value_bytes: hexstring
}
```

Figure 16: HTTP3HTTPField definition

```
HTTP3HeadersFrame = {
    frame_type: "headers"
    headers: [* HTTP3HTTPField]
    ? raw: RawInfo
}
```

Figure 17: HTTP3HeadersFrame definition

For example, the HTTP field section

```
:path: /index.html
:method: GET
:authority: example.org
:scheme: https
```

would be represented in a JSON serialization as:

```
headers: [
  {
    "name": ":path",
    "value": "/"
  },
  {
    "name": ":method",
    "value": "GET"
  },
  {
    "name": ":authority",
    "value": "example.org"
  },
  {
    "name": ":scheme",
    "value": "https"
  }
]
```

Figure 18: HTTP3HeadersFrame example

Section 4.2 of [HTTP/3] and Section 5.1 of [HTTP] define rules for the characters used in HTTP field sections names and values. Characters outside the range are invalid and result in the message being treated as malformed. It can however be useful to also log these invalid HTTP fields. Characters in the allowed range can be safely logged by the text type used in the name and value fields of HTTP3HTTPField. Characters outside the range are unsafe for the text type and need to be logged using the name_bytes and value_bytes field. An instance of HTTP3HTTPField MUST include either the name or name_bytes field and MAY include both. An HTTP3HTTPField MAY include a value or value_bytes field or neither.

4.3.3. HTTP3CancelPushFrame

```
HTTP3CancelPushFrame = {
    frame_type: "cancel_push"
    push_id: uint64
    ? raw: RawInfo
}
```

Figure 19: HTTP3CancelPushFrame definition

4.3.4. HTTP3SettingsFrame

The settings field can contain zero or more entries. Each setting has a name field, which corresponds to Setting Name as defined (or as would be defined if registered) in the "HTTP/3 Settings" registry maintained at https://www.iana.org/assignments/http3-parameters (https://www.iana.org/assignments/http3-parameters).

An endpoint that receives unknown settings is not able to log a specific name. Instead, the name value of "unknown" can be used and the value captured in the name_bytes field; a numerical value without variable-length integer encoding.

```
HTTP3SettingsFrame = {
    frame_type: "settings"
    settings: [* HTTP3Setting]
    ? raw: RawInfo
}

HTTP3Setting = {
    ? name: $HTTP3SettingsName
    ; only when name === "unknown"
    ? name_bytes: uint64

    value: uint64
}

$HTTP3SettingsName /= "settings_qpack_max_table_capacity" /
                      "settings_max_field_section_size" /
                      "settings_qpack_blocked_streams" /
                      "settings_enable_connect_protocol" /
                      "settings_h3_datagram" /
                      "reserved" /
                      "unknown"
```

Figure 20: HTTP3SettingsFrame definition

4.3.5. HTTP3PushPromiseFrame

```
HTTP3PushPromiseFrame = {
    frame_type: "push_promise"
    push_id: uint64
    headers: [* HTTP3HTTPField]
    ? raw: RawInfo
}
```

Figure 21: HTTP3PushPromiseFrame definition

4.3.6. HTTP3GoAwayFrame

```
HTTP3GoawayFrame = {
    frame_type: "goaway"

    ; Either stream_id or push_id.
    ; This is implicit from the sender of the frame
    id: uint64
    ? raw: RawInfo
}
```

Figure 22: HTTP3GoawayFrame definition

4.3.7. HTTP3MaxPushIDFrame

```
HTTP3MaxPushIDFrame = {
    frame_type: "max_push_id"
    push_id: uint64
    ? raw: RawInfo
}
```

Figure 23: HTTP3MaxPushIDFrame definition

4.3.8. HTTP3PriorityUpdateFrame

The PRIORITY_UPDATE frame is defined in [RFC9218].

```
HTTP3PriorityUpdateFrame = {
    frame_type: "priority_update"

    ; if the prioritized element is a request stream
    ? stream_id: uint64

    ; if the prioritized element is a push stream
    ? push_id: uint64

    priority_field_value: HTTP3Priority
    ? raw: RawInfo
}

; The priority value in ASCII text, encoded using Structured Fields
; Example: u=5, i
HTTP3Priority = text
```

Figure 24: HTTP3PriorityUpdateFrame definition

4.3.9. HTTP3ReservedFrame

The frame_type_bytes field is the numerical value without variable-length integer encoding.

```
HTTP3ReservedFrame = {
    frame_type: "reserved"
    frame_type_bytes: uint64
    ? raw: RawInfo
}
```

Figure 25: HTTP3ReservedFrame definition

4.3.10. HTTP3UnknownFrame

The frame_type_bytes field is the numerical value without variable-length integer encoding.

```
HTTP3UnknownFrame = {
    frame_type: "unknown"
    frame_type_bytes: uint64
    ? raw: RawInfo
}
```

Figure 26: HTTP3UnknownFrame definition

4.3.11. HTTP3ApplicationError

```
HTTP3ApplicationError = "http_no_error" /
                        "http_general_protocol_error" /
                        "http_internal_error" /
                        "http_stream_creation_error" /
                        "http_closed_critical_stream" /
                        "http_frame_unexpected" /
                        "http_frame_error" /
                        "http_excessive_load" /
                        "http_id_error" /
                        "http_settings_error" /
                        "http_missing_settings" /
                        "http_request_rejected" /
                        "http_request_cancelled" /
                        "http_request_incomplete" /
                        "http_early_response" /
                        "http_connect_error" /
                        "http_version_fallback"
```

Figure 27: HTTP3ApplicationError definition

The HTTP3ApplicationError extends the general $ApplicationError definition in the qlog QUIC document, see [QLOG-QUIC].

```
; ensure HTTP errors are properly validated in QUIC events as well
; e.g., QUIC's ConnectionClose Frame
$ApplicationError /= HTTP3ApplicationError
```

5. Security and Privacy Considerations

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

6. 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:http3

Namespace http3

Event Types parameters_set, parameters_restored, stream_type_set, priority_updated, frame_created, frame_parsed, datagram_created, datagram_parsed, push_resolved

Description: Event definitions related to the HTTP/3 application protocol.

Reference: This Document

7. 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>.

[EXTENDED-CONNECT] Hamilton, R., "Bootstrapping WebSockets with HTTP/3", RFC 9220, DOI 10.17487/RFC9220, June 2022, <https://www.rfc-editor.org/rfc/rfc9220>.

[H3-DATAGRAM] Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <https://www.rfc-editor.org/rfc/rfc9297>.

[H3_PRIORITIZATION] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", RFC 9218, DOI 10.17487/RFC9218, June 2022, <https://www.rfc-editor.org/rfc/rfc9218>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[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>.

[RFC9218] Oku, K. and L. Pardue, "Extensible Prioritization Scheme for HTTP", RFC 9218, DOI 10.17487/RFC9218, June 2022, <https://www.rfc-editor.org/rfc/rfc9218>.

[RFC9297] Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <https://www.rfc-editor.org/rfc/rfc9297>.

Acknowledgements

Much of the initial work by Robin Marx was done at the Hasselt and KU Leuven Universities.

Thanks to Jana Iyengar, Brian Trammell, Dmitri Tikhonov, Stephen Petrides, Jari Arkko, Marcus Ihlar, Victor Vasiliev, Mirja Kuehlewind, Jeremy Laine, Kazu Yamamoto, Christian Huitema, Hugo Landau, Kazuho Oku, and Jonathan Lennox for their feedback and suggestions.

Change Log

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

Since draft-ietf-quic-qlog-h3-events-11:

* Replaced all length fields with raw.length (#495)

* Renamed owner to initiator (#498)

Since draft-ietf-quic-qlog-h3-events-09:

* Several editorial changes

* Consistent use of RawInfo and _bytes fields to log raw data (#450)

Since draft-ietf-quic-qlog-h3-events-08:

* Removed individual categories and put every event in the single http3 event schema namespace. Major change (#439)

* Changed protocol id from HTTP3 to HTTP/3 (#428)

Since draft-ietf-quic-qlog-h3-events-07:

* TODO (we forgot...)

Since draft-ietf-quic-qlog-h3-events-06:

* ProtocolEventBody is now called ProtocolEventData (#352)

* Editorial changes (#402)

Since draft-ietf-quic-qlog-h3-events-05:

* Removed all qpack event definitions (#335)

* Various editorial changes

Since draft-ietf-quic-qlog-h3-events-04:

* Renamed ’http’ category to ’h3’ (#300)

* H3HTTPField.value is now optional (#296)

* Added definitions for RFC9297 (HTTP/3 Datagram extension) (#310)

* Added definitions for RFC9218 (HTTP Extensible Prioritizations extension) (#312)

* Added definitions for RFC9220 (Extended Connect extension) (#325)

* Editorial and formatting changes (#298, #258, #299, #304, #327)

Since draft-ietf-quic-qlog-h3-events-03:

* Ensured consistent use of RawInfo to indicate raw wire bytes (#243)

* Changed HTTPStreamTypeSet:raw_stream_type to stream_type_value (#54)

* Changed HTTPUnknownFrame:raw_frame_type to frame_type_value (#54)

* Renamed max_header_list_size to max_field_section_size (#282)

Since draft-ietf-quic-qlog-h3-events-02:

* Renamed HTTPStreamType data to request (#222)

* Added HTTPStreamType value unknown (#227)

* Added HTTPUnknownFrame (#224)

* Replaced old and new fields with stream_type in HTTPStreamTypeSet (#240)

* Changed HTTPFrame to a CDDL plug type (#257)

* Moved data definitions out of the appendix into separate sections

* Added overview Table of Contents

Since draft-ietf-quic-qlog-h3-events-01:

* No changes - new draft to prevent expiration

Since draft-ietf-quic-qlog-h3-events-00:

* Change the data definition language from TypeScript to CDDL (#143)

Since draft-marx-qlog-event-definitions-quic-h3-02:

* These changes were done in preparation of the adoption of the drafts by the QUIC working group (#137)

* Split QUIC and HTTP/3 events into two separate documents

* Moved RawInfo, Importance, Generic events and Simulation events to the main schema document.

Since draft-marx-qlog-event-definitions-quic-h3-01:

Major changes:

* Moved data_moved from http to transport. Also made the "from" and "to" fields flexible strings instead of an enum (#111,#65)

* Moved packet_type fields to PacketHeader. Moved packet_size field out of PacketHeader to RawInfo:length (#40)

* Made events that need to log packet_type and packet_number use a header field instead of logging these fields individually

* Added support for logging retry, stateless reset and initial tokens (#94,#86,#117)

* Moved separate general event categories into a single category "generic" (#47)

* Added "transport:connection_closed" event (#43,#85,#78,#49)

* Added version_information and alpn_information events (#85,#75,#28)

* Added parameters_restored events to help clarify 0-RTT behaviour (#88)

Smaller changes:

* Merged loss_timer events into one loss_timer_updated event

* Field data types are now strongly defined (#10,#39,#36,#115)

* Renamed qpack instruction_received and instruction_sent to instruction_created and instruction_parsed (#114)

* Updated qpack:dynamic_table_updated.update_type. It now has the value "inserted" instead of "added" (#113)

* Updated qpack:dynamic_table_updated. It now has an "owner" field to differentiate encoder vs decoder state (#112)

* Removed push_allowed from http:parameters_set (#110)

* Removed explicit trigger field indications from events, since this was moved to be a generic property of the "data" field (#80)

* Updated transport:connection_id_updated to be more in line with other similar events. Also dropped importance from Core to Base (#45)

* Added length property to PaddingFrame (#34)

* Added packet_number field to transport:frames_processed (#74)

* Added a way to generically log packet header flags (first 8 bits) to PacketHeader

* Added additional guidance on which events to log in which situations (#53)

* Added "simulation:scenario" event to help indicate simulation details

* Added "packets_acked" event (#107)

* Added "datagram_ids" to the datagram_X and packet_X events to allow tracking of coalesced QUIC packets (#91)

* Extended connection_state_updated with more fine-grained states (#49)

Since draft-marx-qlog-event-definitions-quic-h3-00:

* Event and category names are now all lowercase

* Added many new events and their definitions

* "type" fields have been made more specific (especially important for PacketType fields, which are now called packet_type instead of type)

* Events are given an importance indicator (issue #22)

* Event names are more consistent and use past tense (issue #21)

* Triggers have been redefined as properties of the "data" field and updated for most events (issue #23)

Authors' Addresses

Robin Marx (editor)
Akamai
Email: rmarx@akamai.com

Luca Niccolini (editor)
Meta
Email: lniccolini@meta.com

Marten Seemann (editor)
Email: martenseemann@gmail.com

Lucas Pardue (editor)
Cloudflare
Email: lucas@lucaspardue.com