HTTP
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

L. Pardue

2 March 2026

HTTP/2 qlog event definitions
draft-pardue-httpbis-qlog-h2-events-00

Abstract

This document defines a qlog event schema containing concrete events for the core HTTP/2 protocol and selected extensions.

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-httpbis-qlog-h2-events/draft-pardue-httpbis-qlog-h2-events.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-pardue-httpbis-qlog-h2-events/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/.

Source for this draft and an issue tracker can be found at https://github.com/LPardue/draft-pardue-httpbis-qlog-h2-events.

Status of This Memo

Table of Contents

1. Introduction

This document defines a qlog event schema (Section 8 of [QLOG-MAIN]) containing concrete events for the core HTTP/2 protocol [HTTP/2] and selected extensions ([ALTSVC], [ORIGIN], [EXTENDED-CONNECT] and [EXTENSIBLE_PRIORITIZATION]).

The event namespace with identifier http2 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
http2:parameters_set	Core	Section 4.1
http2:parameters_restored	Core	Section 4.2
http2:frame_created	Core	Section 4.3
http2:frame_parsed	Core	Section 4.4

Table 1: HTTP/2 Events

2. Conventions and Definitions

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

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

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

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

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

3. Event Schema Definition

This document describes how HTTP/2 is expressed in qlog with an event schema. Per the requirements in Section 8 of [QLOG-MAIN], this document registers the http2 namespace. The event schema URI is urn:ietf:params:qlog:events:http2.

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:http2. 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:http2-99.

The namespace identifier itself is not affected by this requirement.

4. HTTP/2 Events

HTTP/2 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].

```
HTTP2EventData = HTTP2ParametersSet /
                 HTTP2ParametersRestored /
                 HTTP2FrameCreated /
                 HTTP2FrameParsed

$ProtocolEventData /= HTTP2EventData
```

Figure 1: HTTP2EventData definition and ProtocolEventData extension

4.1. parameters_set

The parameters_set event contains HTTP/2 settings, mostly those received from the SETTINGS frame. It has Core importance level.

Parameters are typically set once but they can 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".

```
HTTP2ParametersSet = {
    ? initiator: Initiator

    ? header_table_size: uint32
    ? enable_push: uint32
    ? max_concurrent_streams: uint32
    ? initial_window_size: uint32
    ? max_frame_size: uint32
    ? max_header_list_size: uint32

    ? extended_connect: uint32
    ? no_rfc7540_priorities: uint32

    * $$http2-parametersset-extension
}
```

Figure 2: HTTP2ParametersSet 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.

4.2. parameters_restored

When using TLS 0-RTT, HTTP/2 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/2 settings were restored and to which values when utilizing 0-RTT. It has Core importance level.

```
HTTP2ParametersRestored = {
    ? initiator: Initiator

    ? header_table_size: uint32
    ? enable_push: uint32
    ? max_concurrent_streams: uint32
    ? initial_window_size: uint32
    ? max_frame_size: uint32
    ? max_header_list_size: uint32

    ? extended_connect: uint32
    ? no_rfc7540_priorities: uint32

    * $$http2-parametersrestored-extension
}
```

Figure 3: HTTP2ParametersRestored definition

4.3. frame_created

The frame_created event is emitted when the HTTP/2 framing actually happens. It has Core importance level.

```
HTTP2FrameCreated = {
    stream_id: uint32
    frame: $HTTP2Frame

    * $$http2-framecreated-extension
}
```

Figure 4: HTTP2FrameCreated definition

4.4. frame_parsed

The frame_parsed event is emitted when the HTTP/2 frame is parsed. It has Core importance level.

```
HTTP2FrameParsed = {
    stream_id: uint32
    frame: $HTTP2Frame

    * $$http2-frameparsed-extension
}
```

Figure 5: HTTP2FrameParsed definition

5. HTTP/2 Data Type Definitions

The following data type definitions can be used in HTTP/2 events.

5.1. Initiator

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

Figure 6: Initiator definition

5.2. HTTP2Frame

The generic $HTTP2Frame is defined here as a CDDL "type socket" extension point. It can be extended to support additional HTTP/2 frame types.

```
; The HTTP2Frame is any key-value map (e.g., JSON object)
$HTTP2Frame /= {
    * text => any
}
```

Figure 7: HTTP2Frame type socket definition

The HTTP/2 frame types defined in this document are as follows:

```
HTTP2BaseFrames = HTTP2DataFrame /
                  HTTP2HeadersFrame /
                  HTTP2PriorityFrame /
                  HTTP2RstStreamFrame /
                  HTTP2SettingsFrame /
                  HTTP2PushPromiseFrame /
                  HTTP2PingFrame /
                  HTTP2GoawayFrame /
                  HTTP2WindowUpdateFrame /
                  HTTP2ContinuationFrame /
                  HTTP2UnknownFrame

HTTP2ExtensionFrames = HTTP2AltSvcFrame /
                       HTTP2OriginFrame /
                       HTTP2PriorityUpdateFrame

$HTTP2Frame /= HTTP2BaseFrames / HTTP2ExtensionFrames
```

Figure 8: HTTP2Frames definition

5.2.1. HTTP2DataFrame

```
HTTP2DataFrame = {
    frame_type: "data"

    ? padded: bool .default false
    ? end_stream: bool .default false

    ? raw: RawInfo
}
```

Figure 9: HTTP2DataFrame definition

5.2.2. HTTP2HeadersFrame

The payload of an HTTP/2 HEADERS frame is the HPACK-encoding of an HTTP field section; see [RFC7541]. HTTP2HeadersFrame, in contrast, contains the HTTP field section without encoding.

```
HTTP2HTTPField = {
    ? name: text
    ? name_bytes: hexstring
    ? value: text
    ? value_bytes: hexstring
}
```

Figure 10: HTTP2HTTPField definition

```
HTTP2HeadersFrame = {
    frame_type: "headers"
    ? priority: bool .default false
    ? padded: bool .default false
    ? end_headers: bool .default false
    ? end_stream: bool .default false

    headers: [* HTTP2HTTPField]
    ? raw: RawInfo
}
```

Figure 11: HTTP2HeadersFrame 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 12: HTTP2HeadersFrame example

[HTTP/2] 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 HTTP2HTTPField. 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 HTTP2HTTPField MUST include either the name or name_bytes field and MAY include both. An HTTP2HTTPField MAY include a value or value_bytes field or neither.

5.2.3. HTTP2PriorityFrame

```
HTTP2PriorityFrame = {
    frame_type: "priority"

    exclusive: bool .default false
    stream_dependency: uint32
    weight: uint8
    ? raw: RawInfo
}
```

Figure 13: HTTP2PriorityFrame definition

5.2.4. HTTP2RstStreamFrame

```
HTTP2RstStreamFrame = {
    frame_type: "rst_stream"

    error_code: uint32

    ? raw: RawInfo
}
```

Figure 14: HTTPRstStreamFrame definition

5.2.5. HTTP2SettingsFrame

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/HTTP2-parameters (https://www.iana.org/assignments/HTTP2-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.

```
HTTP2SettingsFrame = {
    frame_type: "settings"

    ? ack: bool .default false

    settings: [* HTTP2Setting]
    ? raw: RawInfo
}

HTTP2Setting = {
    ? name: $HTTP2SettingsName
    ; only when name === "unknown"
    ? name_bytes: uint32

    value: uint32
}

$HTTP2SettingsName /= "settings_header_table_size" /
                      "settings_enable_push" /
                      "settings_max_concurrent_streams" /
                      "settings_initial_window_size" /
                      "settings_max_frame_size" /
                      "settings_max_header_list_size" /
                      "settings_extended_connect" /
                      "settings_no_rfc7540_priorities" /
                      "unknown"
```

Figure 15: HTTP2SettingsFrame definition

5.2.6. HTTP2PushPromiseFrame

```
HTTP2PushPromiseFrame = {
    frame_type: "push_promise"

    ? padded: bool .default false
    ? end_headers: bool .default false

    promised_stream_id: uint32
    headers: [* HTTP2HTTPField]
    ? raw: RawInfo
}
```

Figure 16: HTTP2PushPromiseFrame definition

5.2.7. HTTP2PingFrame

```
HTTP2PingFrame = {
    frame_type: "ping"

    ? ack: bool .default false

    opaque_data: hexstring
    ? raw: RawInfo
}
```

Figure 17: HTTP2PingFrame definition

5.2.8. HTTP2GoAwayFrame

```
HTTP2GoawayFrame = {
    frame_type: "goaway"

    last_stream_id: uint32
    error_code: uint32
    ? additional_debug_data: hexstring
    ? raw: RawInfo
}
```

Figure 18: HTTP2GoawayFrame definition

5.2.9. HTTP2WindowUpdateFrame

```
HTTP2WindowUpdateFrame = {
    frame_type: "window_update"

    window_size_increment: uint32
    ? raw: RawInfo
}
```

Figure 19: HTTP2WindowUpdateFrame definition

5.2.10. HTTP2ContinuationFrame

```
HTTP2ContinuationFrame = {
    frame_type: "continuation"

    ? end_headers: bool .default false

    headers: [* HTTP2HTTPField]
    ? raw: RawInfo
}
```

Figure 20: HTTP2ContinuationFrame definition

5.2.11. HTTP2UnknownFrame

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

```
HTTP2UnknownFrame = {
    frame_type: "unknown"
    frame_type_bytes: uint32
    ? raw: RawInfo
}
```

Figure 21: HTTP2UnknownFrame definition

5.2.12. HTTP2AltSvcFrame

The ALTSVC frame is defined in [ALTSVC].

```
HTTP2AltSvcFrame = {
  frame_type: "altsvc"

  origin_len: uint16
  ? origin: text
  ? alt-svc-field-value: text
  ? raw: RawInfo
}
```

Figure 22: HTTP2AltSvcFrame definition

5.2.13. HTTP2OriginFrame

The ORIGIN frame is defined in [ORIGIN].

```
HTTP2OriginEntry = {
  origin_len: uint16
  ? ASCII-Origin: text
}

HTTP2OriginFrame = {
  frame_type: "origin"

  origin_entries: [* HTTP2OriginEntry]
  ? raw: RawInfo
}
```

Figure 23: HTTP2OriginFrame definition

5.2.14. HTTP2PriorityUpdateFrame

The PRIORITY_UPDATE frame is defined in [EXTENSIBLE_PRIORITIZATION].

```
HTTP2PriorityUpdateFrame = {
  frame_type: "priority_update"

  prioritized_stream_id: uint32
  priority_field_value: HTTP2Priority
  ? raw: RawInfo
}

; The priority value in ASCII text, encoded using Structured Fields
; Example: u=5, i
HTTP2Priority = text
```

Figure 24: HTTP2PriorityUpdateFrame definition

6. Security Considerations

The security and privacy considerations discussed in Section 14 of [QLOG-MAIN] apply to this document as well.

7. IANA Considerations

This document will have IANA actions if adopted.

8. References

8.1. Normative References

[ALTSVC] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/rfc/rfc7838>.

[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] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <https://www.rfc-editor.org/rfc/rfc8441>.

[EXTENSIBLE_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/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[ORIGIN] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", RFC 8336, DOI 10.17487/RFC8336, March 2018, <https://www.rfc-editor.org/rfc/rfc8336>.

[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>.

[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>.

8.2. Informative References

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/rfc/rfc7541>.

Acknowledgments

TODO acknowledge.

Author’s Address

Lucas Pardue

Email: lucas@lucaspardue.com