Building Blocks for HTTP APIs
Internet-Draft
Intended status: Standards Track
Expires: 5 January 2026

R. Gupta
4 July 2025

HTTP Events Query
draft-gupta-httpapi-events-query-00

Abstract

Events Query is a minimal protocol built on top of HTTP that allows user agents to receive event notifications directly from any resource of interest. The Events Query Protocol (EQP) is predicated on the idea that the most intuitive source for event notifications is the resource itself.

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://CxRes.github.io/events-query/draft-gupta-httpapi-events-query.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-gupta-httpapi-events-query/.

Discussion of this document takes place on the Building Blocks for HTTP APIs Working Group mailing list (mailto:httpapi@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/httpapi/. Subscribe at https://www.ietf.org/mailman/listinfo/httpapi/.

Source for this draft and an issue tracker can be found at https://github.com/CxRes/events-query.

Status of This Memo

This Internet-Draft will expire on 5 January 2026.

Table of Contents

1. Introduction

HTTP was originally designed for transferring static documents within a single request and response. HTTP does not automatically inform clients of changes to a document. This design was adequate for web pages that were mostly static and written by hand.

But web-applications today are dynamic, requiring instantaneous updates from sources. The many workarounds developed over the years to provide real-time updates for resources using HTTP have proven to be inadequate. Web programmers instead resort to implementing custom messaging systems over alternate protocols such as WebSockets [WS], which requires additional layers of code, typically involving non-standard JavaScript frameworks to provide event notifications. It also requires additional work to coordinate a representation and notifications that are served on different protocols.

Events Query is a minimal protocol built on top of [HTTP] that allows applications to request event notifications directly from a resource of interest using the QUERY method ([HTTP-QUERY], Section 3).

The objective of this specification is to make the request and receipt of event notifications extremely convenient for consumers. Programmers implementing Events Query shall no longer be forced to switch to another protocol to incorporate real-time functionality in their web applications. Not only that, web-applications shall receive a representation and notifications in a single response, obviating any need for co-ordination and saving on unnecessary roundtrips.

With the help of a suitable composite media-type parser, Events Query responses can be consumed with just a few lines of code, as illustrated in the JavaScript example below:

```
const response = fetch("http://example.com/foo", {
  method: "QUERY",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/http"
  },
  body: JSON.stringify({
    state: { Accept: "text/plain" },
    events: { Accept: "example/event-request" }
  })
});

const splitResponse = splitHTTPResponseStream(response);
// splits the response into an iterable of representation and notifications

const {done, value: representation} = await splitResponse.next();
if (!done) {
  // do something with the representation
  // API identical to fetch Response
}

for await (const notification of splitResponse) {
  // do something with a notification
  // API identical to fetch Response
}
```

Figure 1: Events Query fetch example

Unlike other HTTP based event notification mechanisms, Events Query supports content negotiation for notifications, just like representations. Thus, the Events Query Protocol preserves the flexibility of interaction afforded by HTTP and extends it to event notifications.

When combined with suitable data synchronization mechanisms like Conflict Free Replicated Data Types (CRDT) or Operational Transforms (OT), event notifications can be used create representations that are "live" for user agents. This has the potential to immensely simplify the task of programming multi-author distributed real-time applications.

2. Design

Events Query is predicated on a resource being the most intuitive source for notifying its events. Events Query treats notifications as a response to a query for an event occurring on the resource. With HTTP allowing representations to provide a potentially unbounded stream of data, the Events Query Protocol is also able to communicate multiple events on the resource as a stream of notifications.

Unlike other protocols, Events Query does not (usually) require additional resources to be specifically dedicated as endpoints for delivering event notifications. By giving a resource the ability to send notifications when an event occurs, Events Query aims to reduce the complexity of both servers and clients implementing notifications, making it easier to develop real-time applications.

2.1. Goals

The goals of the Events Query are:

1. to use the Hypertext Transfer Protocol [HTTP] for reliable and in-order transfer of event notifications. Clients fetching resources using HTTP need not required to switch to another protocol for receiving event notifications.

2. to provide updates directly from a resource of interest, obviating the need to create another endpoint for event notifications. This eliminates the need to co-ordinate responses between a resource and the notification endpoint as is the case with existing approaches.

3. to allow clients to fetch representation and notifications in response to a single request, minimizing round-trips between clients and servers and eliminating the need to manage race conditions between responses.

4. to allow event notifications to be communicated in any arbitrary format that might be negotiated. Implementers shall be able to provide more expressive notifications in comparison to existing HTTP based messaging protocols such as Server Sent Events [SSE].

5. to specify transparent semantics that allow intermediaries to participate in scaling, improving reliability, and reducing latency of event notifications as well as proactively update caches.

2.2. Constraints

To the extent feasible, the Events Query:

1. adheres to established practices and conventions. In particular, every attempt has been made to reuse existing protocols and specifications. Implementers shall be able to repurpose existing tools and libraries for implementing this specification.

2. conforms to Representational State Transfer (REST), best practices for Building Protocols with HTTP [RFC9205], and Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP [RFC6202]. This specification can, thus, be used to extend the capabilities of any existing HTTP resource to provide event notifications. Implementers shall be able to scale notifications along with their data/ applications.

2.3. Scope

The Events Query Protocol specifies:

1. A mechanism to discover notification capabilities on a resource (Section 7).

2. A mechanism to request event notifications from a resource (Sections 8.1 and 9.1) along with the representation (Section 10.1).

3. An abstract data model for the subscription (Section 6).

4. Semantics for a response carrying a single notification (Section 8.2).

5. Semantics for the response streaming multiple event notifications (Section 9.2) as well as the representation (Section 10.2), if requested.

The Events Query Protocol does not specify:

1. A realization of the abstract data model used for requesting event notifications. For the purposes of illustration, we shall use an imaginary example/event-request media-type for the request.

2. Specific events for which a notification is generated. Events for which notifications are generated can vary per resource.

3. The form or content of an event notification. Implementations have the flexibility to generate event notifications for the applications they wish to support on a resource. We shall use a very simple YAML notification using an imaginary example/event-response media-type for illustration.

4. Specific representations for the response stream with multiple notifications. For the purpose of illustration, we shall use the application/http media-type ([HTTP/1.1], Section 10.2) as the composite media-type for the response that includes a representation and/or multiple event notifications.

2.4. Limitations

Events Query only allows notifications to be sent for events on a given resource. To send notifications for events on multiple resources in a single response, implementations will need to mint separate resources as notification endpoints. This is no different from APIs built on top of existing messaging protocols (See, for example, [WS] and [WEBSUB]).

Browsers cap the number of persistent HTTP/1.1 connections per host, limiting the suitability of Events Query for web applications in the browser that require simultaneous event notifications from multiple resources on the same host. This limitation is identical to that of other HTTP streaming based protocols, such as Server-Sent Events [SSE]. Implementations are strongly encouraged to adopt HTTP/2 (or later). HTTP/1.1 servers might consider setting up a reverse proxy over HTTP/2 (or later) or implement mitigation strategies, such as to maximize the number of concurrent connections and to provide alternate hosts for resources. Implementations might alternatively consider using endpoints to provide event notifications for multiple resources as previously described. Clients on a browser requesting event notifications over an HTTP/1.1 connection are advised to exercise caution when opening multiple simultaneous persistent connections to any given host.

3. Conformance

3.1. Document Conventions

All examples and notes in this specification are non-normative.

3.2. Requirements Notation

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.

4. Terminology and Core Concepts

4.1. Event

An event is the instantaneous effect of the (normal or abnormal) termination of invocation of an operation on an object of interest [DESIGN-FRAMEWORK]. The entity invoking an operation is termed the *invoker*.

In the specific context of HTTP, the object of interest is data scoped to some resource. When the operation is an HTTP method, the invoker is a user agent. However, an operation need not be limited an HTTP method, it might just as easily have been invoked using another mechanism or protocol. Events are then an extension of resource state (See [HTTP], Section 3.2) in the temporal dimension.

4.2. Observation

An event is considered observable, if an entity outside the invoker and object of interest can detect its occurrence [DESIGN-FRAMEWORK]. This entity is the *observer*.

It follows from the HTTP uniform interface that the observer is always a server. The events that are observed, the mechanism of observation, and information recorded from the event are implementation details for the server.

That an origin server has to assume the role of an observer in order to generate event notifications is obvious. An intermediary while not observing the data scoped to a resource directly, still has the possibility to serve as an observer. An intermediary can observe events transmitted by an origin server or another intermediary, whether using Events Query or another mechanism, to generate event notifications for outbound consumers.

4.3. Event Notification

An event notification, or notification, is information transmitted by an observer upon an event or contiguous events on a resource.

Events Query extends "information hiding" behind the HTTP uniform interface to the temporal dimension by defining communication with respect to a transferable notification of the resource event(s), rather than transferring the event(s) themselves.

A target resource might be capable of generating multiple notifications for the same event(s) that a subscriber can select from using content negotiation. Hypertext notifications can not only provide information of the resource events but also processing instructions that help guide the recipient’s future actions, for example, the possibility to determine the current representation from a previous representation.

4.4. Subscription

A subscription is an expression of interest to receive event notifications sent to an observer. The requesting entity is a *subscriber*.

Due to the request/response semantics of HTTP, the subscriber coincides with the recipient of event notifications ([DESIGN-FRAMEWORK] uses the term _requester_ or _broker_ to identify a requesting entity, with the _broker_ and _recipient_ together forming the subscriber; for this specification the distinction is not necessary).

The subscription in the form of a query affords the user agent the opportunity to engage in content negotiation for preferred form of event notifications (as well as the representation, if simultaneously requested).

5. Events Header Field

The Events header field when used by a client in a request communicates preferences for the Events Query response. The Events header field is not meant for content negotiation.

The Events header field allows a server to communicate the properties of a response carrying event notifications.

Events is a Dictionary structured header field ([HTTP-SF], Section 3.2). The order of keys is insignificant. Senders SHOULD NOT generate keys not registered with the HTTP Event Field Registry (An exception is made to allow for experimentation). Recipients MAY ignore keys that they are unaware of.

5.1. duration Property

The duration property when be used by a client in a request specifies the duration for which they prefer the response stream to remain open. A server is completely free to ignore this property.

The duration property when used by a server in a response indicates the minimum duration for which a server intends to keep the response stream open. This property is merely advisory, and a server might still close the response stream before this duration.

The duration property is a key specified in the Events header field. It is of the type Integer ([HTTP-SF], Section 3.3.1) with its value indicating duration in seconds. Only positive integer values are valid. A value of 0 indicates an indefinite duration. A sender MUST conform to these stipulations when generating the duration property. If the value of the duration property fails to conform to these stipulations, it MUST be ignored by the recipient.

6. Subscription Data Model

The abstract data model specifies the semantics of an Events Query.

A realization of the data model MUST allow a client to specify in a subscription request:

* interest in receiving a representation in a preferred form.

* interest in receiving event notifications in a preferred form.

Implementations can choose an appropriate media-type to realize the subscription data model. Implementations are free to extend the data model to include additional data. A specific realization of the data model is outside the scope of this specification.

The following example shows the body of a subscription request wherein the state and events properties are used to specify request headers for representation and event notifications respectively in a YAML like syntax.

```
state:
  Accept: text/html
events:
  Accept: example/event-response
```

Figure 2: Events Query Data Model in a YAML like syntax

7. Discovery

A user agent can discover if a server enables Events Query on a resource by examining support for query with a media-type that can realize the Subscription Data Model. A server MUST advertise media-types accepted for Events Query using the Accept-Query header field ([HTTP-QUERY], Section 3) in a response.

```
HEAD /foo HTTP/1.1
Host: example.org
```

Figure 3: Discovery Request

```
HTTP/1.1 200 OK
Accept: text/html
Accept-Query: "example/events-request"
```

Figure 4: Discovery Response

> | *Implementation Advice*
> |
> | Servers are advised against enabling event notifications on long-lived resources over HTTP. A resource might be considered long-lived, if a server determines a low probability of an event on the resource in the duration of the response. In such instances, servers are strongly advised to respond with the Cache-Control ([HTTP-CACHING], Section 5) header field and the max-age parameter ([HTTP-CACHING], Section 5.2.2.1) set in it.

8. Single Notification

The simplest event query is to ask for a notification for the next event(s) on a resource. This, in effect, adds long-polling capability ([RFC6202], Section 2) to a resource.

8.1. Request

To be notified of the next event(s) on a resource using Events Query, a client can send an empty query. A server MUST consider a request with an empty query in an appropriate media-type made using the QUERY method ([HTTP-QUERY], Section 3) as a subscription request for a single event notification.

A client can, as usual, negotiate the form of the event notification using header fields.

```
QUERY /foo HTTP/1.1
Host: example.org
Content-Type: example/event-request
Accept: example/event-response
Events: duration=0
```

Figure 5: Single Event Notification Request

8.2. Response

When providing a single notification, the server MUST close the connection immediately after transmitting the event notification.

```
HTTP/1.1 200 OK
Host: example.org
Accept-Query: example/event-request
Content-Type: example/event-response
Incremental: ?1

Event-ID: 456
Type: Update
```

Figure 6: Single Event Notification Response

> | *Implementation Advice*
> |
> | When a user navigates away from a website or an application using Events Query, user agents are strongly encouraged to properly close the response and release the connection.

9. Notification Stream

Instead of long-polling for event notifications, Events Query can also be used request a stream ([RFC6202], Section 3) of multiple event notifications.

9.1. Request

To request a stream of event notifications from a resource in an Events Query, a client MUST express the interest in receiving the event notifications in a preferred form using a realization of the subscription data model with the QUERY method ([HTTP-QUERY], Section 3).

A client can also negotiate the response that encapsulates event notifications using header fields. Since the response carries an encapsulating representation, header fields can no longer be used to negotiate the form of an event notification itself like in the case of a Single Notification Request (Section 8.1).

The following example shows subscription request for a stream of event notifications transmitted using the application/http media-type. The events property indicates the interest in receiving event notifications. Preferences are specified using the request headers in the events property.

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: application/http
Content-Type: example/event-request
Events: duration=0

events:
  Accept: example/event-response
```

Figure 7: Notifications Stream Request

9.2. Response

The response stream encapsulates multiple event notifications (typically, but not necessarily) in a composite media-type. We shall be using application/http media-type ([HTTP/1.1], Section 10.2) for the purpose of illustration.

9.2.1. Headers

A server able to provide a stream of event notifications immediately sends the header which MUST include:

* The Events header field to communicate the properties of the notifications stream.
  - The duration property set with the time for which the server intends to serve notifications.
* The Incremental header field ([INCREMENTAL-HTTP-MESSAGES], Section 3) set to ?1 to indicate that the response is to be immediately forwarded by intermediaries and not buffered.

```
HTTP/1.1 200 OK
Accept-Query: example/event-request
Content-Type: application/http
Transfer-Encoding: chunked
Incremental: ?1
Events: duration=600
```

Figure 8: Notifications Stream Response Headers

9.2.2. Notifications

Subsequently, when event(s) occur, the server transmits a notification identical to the Single Notification Response (Section 8.2), except header fields redundant with response header (Section 9.2.1) are omitted.

```
HTTP/1.1 200 OK                                  | Notification
Content-Type: example/event-response             |
Content-Length: 31                               |
                                                 |
Event-ID: 456                                    |
Type: Update                                     |
```

Figure 9: Update Notification

A server MUST end the response immediately after transmitting the event notification upon a resource being deleted.

```
HTTP/1.1 200 OK                                  | Notification
Content-Type: example/event-response             |
Content-Length: 31                               |
                                                 |
Event-ID: 789                                    |
Type: Delete                                     |
```

Figure 10: Delete Notification

Otherwise, a server MUST end the response when the connection duration has exceeded the period set in the duration property of the Events header field. A server MAY terminate the response earlier.

10. Representation

Events Query lets a user agent to ask and receive the current representation and subsequent event notifications in a single request/response. When compared to using, say, Fetch [FETCH] and EventSource [SSE] in conjunction, Events Query not only saves on an extra round trip, but relieves a user agent from the burden of handling the possible race condition.

10.1. Request

To request a representation of the resource in an Events Query, a client MUST express the interest in receiving the representation in a preferred form using a realization of the subscription data model (Section 6) with the QUERY method ([HTTP-QUERY], Section 3).

The following example shows subscription request for representation along with notifications transmitted using the application/http media-type. The state property indicates the interest in receiving representation. The preferred form of representation is specified using the request headers in the state property.

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: application/http
Content-Type: example/event-request

state:
  Accept: text/html
events:
  Accept: example/event-response
```

Figure 11: Representation and Notifications Request

10.2. Response

A server unable to provide a representation MUST NOT serve event notifications. This does not apply in case of conditional request for representation that is not fulfilled.

A server able to provide a stream with a representation and event notifications transmits the representation immediately following the response header (Section 9.2.1). Otherwise, the response is the same as that described in Section 9.2.

Again, we shall use the application/http media-type ([HTTP/1.1], Section 10.2) for the purpose of illustration. Chunks have been omitted for clarity.

```
HTTP/1.1 200 OK
Accept-Query: example/event-request
Content-Type: application/http
Transfer-Encoding: chunked
Incremental: ?1
Events: duration=600

HTTP/1.1 200 OK                                  | Representation
Content-Type: text/plain                         |
Content-Length: 14                               |
                                                 |
Hello World!                                     |
```

Figure 12: Representation Response

While this is default behaviour, there is no requirement that a representation is the first message or is sent only once. In such cases, the encapsulated message needs to indicate if it is a representation and not an event notification. Such a mechanism is not defined in this specification.

Notifications are transmitted just as described in Section 9.2. See Appendix A for a complete example of a response with representation and notifications.

11. Security Considerations

Events Query is subject to the security considerations of the HTTP QUERY method ([HTTP-QUERY], Section 2) and more generally HTTP Semantics. Considerations relevant to the use of HTTP QUERY method are discussed in Section 4 of [HTTP-QUERY] and HTTP Semantics and its use for transferring information over the Internet are discussed in Section 17 of [HTTP].

When serving a stream of event notifications, resources are required to keep the response stream open for an extended period of time, making them more susceptible to Denial-of-Service attacks because the effort required to request notifications from the same resource is tiny compared to the time, memory, and bandwidth consumed by attempting to serve the notifications. Servers ought to ignore, coalesce, or reject egregious event notification request, such as repeated notification requests to a resource from the same origin.

12. IANA Considerations

The change controller for the following registrations is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

12.1. HTTP Field Registration

IANA is requested to add the following entry in the "Hypertext Transfer Protocol (HTTP) Field Name Registry (https://www.iana.org/assignments/http-fields/)" (See Section 16.1.1 of [HTTP]):

Header Field Names	Status	Structured-Type	Reference
Events	Permanent	Dictionary	Section 5

Table 1: List of HTTP Field Name registrations

12.2. The HTTP Events Field Registry

IANA is requested to create a new registry, "HTTP Events Field Registry", under the Hypertext Transfer Protocol (HTTP) Parameters (https://www.iana.org/assignments/http-parameters/) registry to register properties for use in the Events header field. New registrations will use the Specification Required policy ([RFC8126], Section 4.6).

12.2.1. Template

The registration of an Events property MUST include the following fields:

* Property Name: A Dictionary ([HTTP-SF], Section 3.2) key to be used in the Events header field.

* Structured Type: The Structured Data Type ([HTTP-SF], Section 3.3) of the value associated with the key, according to requirements in Section 3.2 of [HTTP-SF].

* Reference: A pointer to the specification text.

The registration MAY also include the following fields:

* Optional Parameters: An enumeration of optional parameters and the Structured Data Type (Section 3.3 of [HTTP-SF]) of value associated with the parameter, according to requirements in Section 3.1.2 of [HTTP-SF]

* Comments: Additional information to be included in the template.

12.2.2. Initial Registry Contents

The initial contents of the HTTP Events Field Registry are:

Property Name	Structured-Type	Reference
duration	Integer Item	Section 5.1

Table 2: List of HTTP Events Field property name registrations

13. End User Considerations

// If we, the IETF, claim that the Internet is for the end user
// [RFC8890] and promote the end-to-end principle [RFC3724], every
// specification we produce ought to consider its impact on the
// Internet end user. For this reason, I propose that specifications
// must include a considerations section where authors assess the
// impact of their proposal on the internet end user, aligned with
// the mission of IETF [RFC3935].

End users of the HTTP protocol can be classified into two groups: publishers and consumers. Consumers have an incentive to subscribe to event notifications from many resources and to hold on to a connection for as long as possible. Whereas publishers bear the cost of server infrastructure. Consumers also typically outnumber publishers, in many cases by multiple orders of magnitude. This creates an imbalance in the effort to subscribe versus effort to deliver; consumers can easily place a disproportionate burden on servers, reminiscent of a denial-of-service attack.

At the outset, requiring that clients subscribe to event notifications per resource serves as an effective filtering mechanism that limits the burden on servers. Compare this to the typical implementation of protocols such as WebSockets [WS], where clients connect to dedicated endpoints to receive notifications; the server either has to broadcast notifications for multiple resources or track resources of interest for each client to filter event notifications accordingly.

Events Query empowers servers to decide content and duration for which event notifications are served on any given resource, as well as allowing servers to close the response stream at any time. Servers may also limit event notifications and/or their content, except to authenticated consumers. Such authenticated consumers might, for example, be asked to share the cost burden with publishers in return for a higher quality of service.

The use of HTTP Semantics also enables intermediation of event notifications, unlike existing mechanisms built with protocols such as WebSockets [WS] or WebSub [WEBSUB]. Intermediaries can help with improving the latency and reliability of transmission of event notifications as well as scaling of the event notification traffic to reach a significantly larger base of consumers. On the flip side, economies of scale will likely lead to greater consolidation of intermediary service providers (though not centralization) with the attendant risk of anti-consumer behaviour. In the opinion of the authors, policies designed to treat network traffic as a public utility might provide better outcomes for the end user.

14. References

14.1. Normative References

[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-QUERY] Reschke, J., Snell, J. M., and M. Bishop, "The HTTP QUERY Method", Work in Progress, Internet-Draft, draft-ietf-httpbis-safe-method-w-body-10, 29 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-safe-method-w-body-10>.

[HTTP-SF] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <https://www.rfc-editor.org/rfc/rfc9651>.

[INCREMENTAL-HTTP-MESSAGES] Oku, K., Pauly, T., and M. Thomson, "Incremental HTTP Messages", Work in Progress, Internet-Draft, draft-ietf-httpbis-incremental-00, 29 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-incremental-00>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[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>.

14.2. Informative References

[DESIGN-FRAMEWORK] Rosenblum, D. and A. Wolf, "A design framework for Internet-scale event observation and notification", Association for Computing Machinery (ACM), ACM SIGSOFT Software Engineering Notes vol. 22, no. 6, pp. 344-360, DOI 10.1145/267896.267920, November 1997, <https://doi.org/10.1145/267896.267920>.

[FETCH] van Kesteren, A., "Fetch", WHATWG Living Standard, May 2025, <https://fetch.spec.whatwg.org>.

[HTTP-CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/rfc/rfc9111>.

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[REST] Fielding, R., "Representational State Transfer (REST)", Chapter 5, Architectural Styles and the Design of Network-based Software Architectures, Doctoral Dissertation University of California, Irvine, <https://roy.gbiv.com/pubs/dissertation/rest_arch_style.htm>.

[RFC3724] Kempf, J., Ed., Austein, R., Ed., and IAB, "The Rise of the Middle and the Future of End-to-End: Reflections on the Evolution of the Internet Architecture", RFC 3724, DOI 10.17487/RFC3724, March 2004, <https://www.rfc-editor.org/rfc/rfc3724>.

[RFC3935] Alvestrand, H., "A Mission Statement for the IETF", BCP 95, RFC 3935, DOI 10.17487/RFC3935, October 2004, <https://www.rfc-editor.org/rfc/rfc3935>.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, April 2011, <https://www.rfc-editor.org/rfc/rfc6202>.

[RFC7838] 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>.

[RFC8890] Nottingham, M., "The Internet is for End Users", RFC 8890, DOI 10.17487/RFC8890, August 2020, <https://www.rfc-editor.org/rfc/rfc8890>.

[RFC9205] Nottingham, M., "Building Protocols with HTTP", BCP 56, RFC 9205, DOI 10.17487/RFC9205, June 2022, <https://www.rfc-editor.org/rfc/rfc9205>.

[SSE] "Server-Sent Events", W3C REC eventsource, W3C eventsource, <https://www.w3.org/TR/eventsource/>.

[WEBSUB] "WebSub", W3C REC websub, W3C websub, <https://www.w3.org/TR/websub/>.

[WS] "The WebSocket API", W3C NOTE websockets, W3C websockets, <https://www.w3.org/TR/websockets/>.

Appendix A. Example

The following example illustrates a complete request and response for representation and notifications using Events Query. Chunks have been omitted for clarity.

A.1. Request

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: application/http
Content-Type: example/event-request

state:
  Accept: text/html
events:
  Accept: example/event-response
```

Figure 13: Request for Representation and Notifications

A.2. Response

```
HTTP/1.1 200 OK
Accept-Query: example/event-request
Content-Type: application/http
Transfer-Encoding: chunked
Incremental: ?1
Events: duration=600

HTTP/1.1 200 OK                              | Representation
Content-Type: text/plain                     |
Content-Length: 14                           |
                                             |
Hello World!                                 |

HTTP/1.1 200 OK                              | Notification
Content-Type: example/event-response         |
Content-Length: 31                           |
                                             |
Event-ID: 456                                |
Type: Update                                 |

HTTP/1.1 200 OK                              | Notification
Content-Type: example/event-response         |
Content-Length: 31                           |
                                             |
Event-ID: 789                                |
Type: Delete                                 |
```

Figure 14: Response Stream with Representation and Notifications

Acknowledgments

We thank members of the HTTP Working Group, the Solid community, the Braid community and others for discussions, ideas, reviews, and feedback on previous work that has led to specification.

Index

D E I N O S

D

E

Author’s Address

Rahul Gupta
Email: cxres+ietf@protonmail.com