Building Blocks for HTTP APIs
Internet-Draft
Intended status: Standards Track
Expires: 19 October 2026

R. Gupta
17 April 2026

HTTP Events Query
draft-gupta-httpapi-events-query-02

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 19 October 2026.

Table of Contents

1. Introduction

HTTP was originally designed as a stateless, request/response protocol for transferring hypermedia documents on the web ([HTTP], Section 1.2). This design was adequate for web pages that were mostly static and written by hand.

But web applications have evolved over time to provide increasingly dynamic and interactive experiences, requiring near-instantaneous updates. HTTP does not automatically inform clients of changes to a resource. Developers have employed various techniques, such as Comet [COMET] and Server-Sent Events ([HTML], Section 9.2) to work around this constraint, but these can be suboptimal for many applications.

For this reason, web programmers often prefer implementing custom messaging systems over alternate protocols such as WebSocket [WS] and WebSub [WEBSUB]. Not only does this approach require additional layers of code, involving multiple Web APIs and/or userland libraries such as socket.io (https://socket.io/), it demands extra coordination effort to synchronize representation and notifications across multiple protocols. The dual-protocol approach thus compounds the development and maintenance overhead. Furthermore, deployment at scale is challenging with the notifications traffic now opaque to network intermediaries.

Events Query is a minimal protocol built on top of the Hypertext Transfer Protocol [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. Events Query allows programmers to incorporate real-time functionality in their web applications without the need to switch to another protocol. Further, Events Query can deliver representation and notifications from a resource in a single response, obviating any need for coordination and saving on unnecessary round trips.

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 = await fetch("http://example.com/foo", {
  method: "QUERY",
  headers: {
    Accept: "multipart/mixed",
    "Content-Type": "example/events-query"
  },
  body: JSON.stringify({
    state: { Accept: "text/plain" },
    events: { Accept: "example/event-notification" }
  })
});

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

// read the representation
const {done, value: representation} = await splitResponse.next();
if (!done) {
  // do something with the representation
}

// iterate over notifications
for await (const notification of splitResponse) {
  // do something with a notification
}
```

Figure 1: Events Query fetch example

Unlike other 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 technologies like Conflict Free Replicated Data Types (CRDT) or Operational Transforms (OT), event notifications can be used to create "live" representations. This can immensely simplify the task of programming multi-author distributed real-time applications.

2. Design

Events Query is predicated on a resource itself being the most intuitive source for events on it. 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.

2.1. Goals

To aid the development of real-time applications, it is imperative that the Events Query Protocol reduces the complexity of both servers and clients implementing event notifications. With this in mind, the goals of the Events Query Protocol are:

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

2. to send updates directly from a resource of interest, obviating the need for additional resources to be specifically dedicated as notification endpoints. In contrast to existing approaches, this frees up the client from the burden of coordinating the response from the resource of interest with notifications from an endpoint.

3. to deliver the representation and notifications in response to a single request, minimizing round trips between clients and servers. It also eliminates the need to synchronize the delivery of the representation and notifications.

4. to enable the transfer of event notifications using arbitrary formats that might be content-negotiated. This allows implementers to serve notifications that are more expressive, say, in comparison to existing HTTP-based messaging protocols such as Server-Sent Events ([HTML], Section 9.2).

5. to specify transparent notification semantics that empowers intermediaries to scale event notifications, improve reliability and reduce latency. Intermediaries shall also be able to proactively update caches in real-time.

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 or future resource to additionally serve event notifications over HTTP. This is to afford implementers the opportunity to scale notifications with their data and application.

2.3. Scope

The Events Query Protocol specifies:

1. A mechanism to discover support for Events Query 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 requesting event notifications (Section 6).

4. Response semantics for a single notification (Section 8.2).

5. Response semantics for a stream carrying the representation (Section 10.2) (if requested) and multiple event notifications (Section 9.2).

The Events Query Protocol does not specify:

1. A realization of the abstract data model used for requesting event notifications.

2. The events for which a notification might be generated. This can be varied 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.

4. Specific representations for the response stream with multiple notifications.

2.4. Limitations

Events Query only allows notifications to be sent for events on a given resource. To transfer event notifications originating from multiple resources in a single response, implementations will need to mint additional resources to serve as notification endpoints. This is no different from APIs built on top of existing messaging protocols (See, for example, WebSocket [WS] and WebSub [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-based streaming protocols, such as Server-Sent Events ([HTML], Section 9.2). 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, maximizing the number of concurrent connections and providing 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 simultaneously opening multiple persistent connections to a given host.

2.5. Comparison with Server-Sent Events

At the time of writing, Server-Sent Events (SSE) has emerged as the de facto mechanism for transferring event notifications over HTTP.

Server-Sent Events is standardized as part of HTML Living Standard by the WHATWG
// As of October 2022, HTML Living Standard maintainer Anne van

// Kesteren has stated (https://github.com/whatwg/html/
// issues/8297#issuecomment-1291658863) that WHATWG does not intend
// to extend the EventSource API, with reasons for the decision cited
// in a subsequent comment (https://github.com/whatwg/html/
// issues/8297#issuecomment-1291658863). through the EventSource API
([HTML], Section 9.2.2) on user agents and the text/event-stream
media type ([HTML], Section 17.7) to transmit event notifications.
This approach is fundamentally inconsistent with terminology defined
in HTTP Semantics ([HTTP], Section 3.2), transferring a sequence of
asynchronous messages instead of a representation "intended to
reflect a past, current, or desired state of a given resource".
Practically, the text/event-stream media type being limited to only
textual content forces developers to opt for alternative protocols
such as WebSockets [WS], despite the increased complexity. The
EventSource API does not even allow for headers to be sent with
requests.

This has led to a proliferation of non-standard implementations of
Server-Sent Events
// such as this (https://www.npmjs.com/package/event-source-plus),
// this (https://www.npmjs.com/package/@microsoft/fetch-event-
// source), this (https://www.npmjs.com/package/fetch-event-stream),
// this (https://www.npmjs.com/package/extended-eventsource), this
// (https://www.npmjs.com/package/ngx-sse-client) and this
// (https://www.npmjs.com/package/sse.js) that provide support for
custom headers, such as for sending credentials, or use the POST
method to communicate additional configuration in the request body.

Events Query takes a principled approach to event notifications, as
established in Section 4. In particular, the definition of an event
notification as a "representation" of event(s) (Section 4.3) admits
the use of arbitrary media types for event notifications. Clients
can use content negotiation as well as preconditions in a
subscription request. Further, Events Query not only lets clients
receive both the representation and notifications over the same
protocol, viz. HTTP, but these can be encapsulated within a single
HTTP response. Clients do not need to, say, authenticate multiple
times, possibly with different mechanisms, to obtain data that is
logically from the same resource. Clients also do not have to
coordinate and synchronize multiple response streams carrying a
representation and notifications from the same effective resource.

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 the 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 to 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 about 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

"Events" is a Dictionary structured header field ([HTTP-SF], Section 3.2) to communicate the properties of a response stream to an Events Query.

In a request, the Events header field allows a client to indicate its preferences for the properties of the response stream carrying event notifications. The Events header field is not meant for content negotiation.

In a response, the Events header field allows a server to specify the properties of a response stream carrying event notifications.

The order of keys in the Events header field is insignificant. Senders SHOULD NOT generate keys not registered with the HTTP Event Field Registry (with the exception made to allow experimentation). Recipients MAY ignore keys that they are unaware of.

5.1. duration Property

The "duration" property is an Integer ([HTTP-SF], Section 3.3.1) or Decimal ([HTTP-SF], Section 3.3.2) valued Dictionary key specified on the Events header field to communicate the response duration in seconds.

In a request, the duration property indicates the duration for which a client wants to receive event notifications. A server MAY ignore this property.

In a response, the duration property specifies the maximum duration for which a server intends to serve event notifications. This property is merely advisory, and a server MAY close the response stream before this duration.

Only positive 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 allows a client to specify in a subscription request:

* an interest in receiving a stream of event notifications from a resource in a preferred form.

* an interest in receiving a representation of a resource in a preferred form.

Implementations can choose appropriate media types 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-notification
```

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
Date: Thu, 02 Jan 2025 10:00:00 GMT
Accept: text/html
Accept-Query: "example/events-query"
```

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 Events Query is to request 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 receive a single notification, a client makes a QUERY request ([HTTP-QUERY], Section 3) using a realization of the subscription data model that MUST NOT include an interest in receiving a stream of event notifications.

Since the content of the response is an event notification, a client can negotiate its form with header fields in the usual manner.

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

---
```

Figure 5: Single Event Notification Request

8.2. Response

When a single notification is requested, the server MUST close the connection immediately after sending the event notification.

```
HTTP/1.1 200 OK
Date: Thu, 02 Jan 2025 10:10:10 GMT
Accept-Query: example/events-query
Content-Type: example/event-notification
Incremental: ?1

published: 2025-01-02T10:11:12.345Z
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. Notifications Stream

Events Query can also be used to request a stream of multiple event notifications ([RFC6202], Section 3).

9.1. Request

To receive multiple notifications, a client makes a QUERY request ([HTTP-QUERY], Section 3) using a realization of the subscription data model that MUST include an interest in receiving a stream of event notifications in a preferred form.

Since the response transmits event notifications within an encapsulating representation (Section 9.2), it follows that header fields cannot be used to negotiate the form of event notifications as in the case of Single Notification Request (Section 8.1). Instead, header fields are useful for negotiating the representation that encapsulates event notifications. The following examples illustrate subscription requests that negotiate a stream of event notifications to be transferred respectively using a composite media type (multipart/mixed) and a discrete media type (application/json-seq):

The first example shows a subscription request for effectively a stream of discrete Single Notifications (Section 8) to be transferred using multipart/mixed ([RFC2046], Section 5.1.3) as the encapsulating media type. The events property in this example indicates an interest in receiving event notifications and its sub-properties describe the preferred form of notifications. Since the notifications are transferred as a pipeline of HTTP messages, these sub-properties are identical to header fields (Section 8.1, Paragraph 2) used for specifying preconditions and content negotiation in a Single Notification Request (Section 8.1).

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: multipart/mixed
Content-Type: example/events-query
Events: duration=0

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

Figure 7: Request for Multipart Notifications Stream

The second example shows a subscription request for a stream of JSON notifications to be transferred using application/json-seq ([RFC7464]) as the encapsulating media type. The events property in this example is being used to communicate the preferred schema for the requested event notifications.

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: application/json-seq
Content-Type: example/events-query
Events: duration=0

events:
  @context: activity+json
```

Figure 8: Request for JSON Notifications Stream

9.2. Response

The response stream transmits multiple event notifications in an encapsulating media type. The following illustrates event notifications streamed in both a composite media type (multipart/mixed) and a discrete media type (application/json-seq) in response to the example requests (Section 9.1) in Section 9.1.

9.2.1. Headers

A server able to provide a stream of event notifications MUST immediately send headers, which include:

* The Events header field, to communicate the properties of the notifications stream.
  - The duration property, set to the maximum duration for which the server intends to serve event 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
Date: Thu, 02 Jan 2025 10:10:10 GMT
Accept-Query: example/events-query
Events: duration=600
Incremental: ?1
Transfer-Encoding: chunked
```

Figure 9: Notifications Stream Response Headers

The Content-type header field in response to the request in the first example is:

```
Content-type: multipart/mixed; boundary="THIS_STRING_SEPARATES"
```

Figure 10: Notifications Stream Multipart Content Response Header

whereas in response to the request in the second example is:

```
Content-type: application/json-seq
```

Figure 11: Notifications Stream JSON Content Response Headers

9.2.2. Notifications

Subsequently, when event(s) occur, the server transmits a notification.

An event notification transferred in a multipart/mixed response stream is identical to the Single Notification Response (Section 8.2), except that non "Content-*" fields are excluded as required by [RFC2046], Section 5.1:

```
--THIS_STRING_SEPARATES
Content-Length: 31                                | Notification
Content-Type: example/event-notification          |
                                                  |
published: 2025-01-02T10:11:12.345Z               |
event-id: 456                                     |
type: update                                      |
```

Figure 12: Multipart Update Notification

The same event notification when transferred in an application/json-seq response stream is as follows:

```
蝊桀                                               | Notification
  "published": "2025-01-02T10:11:12.345Z",        |
  "event-id": 456,                                |
  "type": "Update"                                |
}                                                 |
```

Figure 13: JSON Update Notification

A server MUST end the response immediately after transmitting the event notification that signals the deletion of a resource.

The notification for a delete event expressed inside a multipart message might be as follows:

```
--THIS_STRING_SEPARATES
Content-Length: 31                                | Notification
Content-Type: example/event-notification          |
                                                  |
published: 2025-01-02T11:12:13.456Z               |
event-id: 789                                     |
type: delete                                      |
--THIS_STRING_SEPARATES--
```

Figure 14: Multipart Delete Notification

The same notification for a delete event as JSON would be as follows:

```
竦粲                                               | Notification
  "published": "2025-01-02T11:12:13.456Z",        |
  "event-id": 789,                                |
  "type": "delete"                                |
}                                                 |
```

Figure 15: JSON Delete Notification

Otherwise, a server MUST end the response when the connection duration exceeds the period set in the duration property of the Events header field.

10. Representation

Events Query enables 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 ([HTML], Section 9.2.2) in conjunction, Events Query not only saves on an extra round trip, but relieves a user agent from the burden of synchronizing the representation with event notifications.

10.1. Request

To receive a representation of the resource alongside notifications, a client makes a QUERY request ([HTTP-QUERY], Section 3) using a realization of the subscription data model that MUST also include an interest in receiving the representation in a preferred form.

The following example shows a subscription request for the current representation along with the subsequent event notifications transmitted using the multipart/mixed media type. The state property indicates interest in receiving representation and its sub-properties describe the preferred form of notifications. Since the representation is being transferred in an HTTP message pipeline, these sub-properties are identical to header fields used for specifying preconditions and content negotiation in a GET request on the said resource.

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: multipart/mixed
Content-Type: example/events-query

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

Figure 16: HTTP Representation and Notifications Request

10.2. Response

A server unable to provide a representation MUST NOT serve event notifications. This does not apply to a 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 headers (Section 9.2.1). Otherwise, the response is the same as that described in Section 9.2.

Again, the multipart-mixed media type ([RFC2046], Section 5.1.3) is used for the purpose of illustration. Chunks have been omitted for clarity.

```
HTTP/1.1 200 OK
Date: Thu, 02 Jan 2025 10:10:10 GMT
Accept-Query: example/events-query
Events: duration=600
Incremental: ?1
Transfer-Encoding: chunked
Content-type: multipart/mixed; boundary="THIS_STRING_SEPARATES"

--THIS_STRING_SEPARATES
Content-Length: 14                              | Representation
Content-Type: text/plain                        |
                                                |
Hello World!                                    |
```

Figure 17: Representation Response before Notifications

While this is default behaviour, there is no requirement that a representation is the first message or that representations are 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 in the case of regular streaming (Section 9.2). See Appendix A.1 for a complete example of a response with representation and notifications.

11. Implementation Status

A toy server built in Express.js demonstrating the Events Query Protocol is available at https://github.com/CxRes/events-query-express-demo. This demo is powered by the following libraries:

* Negotiate Events Field (https://github.com/CxRes/negotiate-events-field): To read the request Events header field (Section 5) and negotiate the response duration (Section 5.1),

* NOSE (https://github.com/CxRes/nose): To observe events on resources and generate notifications in a preferred format, and

* Extended Response (https://github.com/CxRes/extended-response): To write the representation and event notifications on the response stream for a given duration (Section 5.1).

The demonstration and libraries are Free and Open Source Software, released under the Mozilla Public License, v. 2.0. Please contact the author for more information about these programs.

12. 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]. HTTP Semantics and its use for transferring information over the Internet are discussed in Section 17 of [HTTP].

When serving event notifications, servers are required to keep the response stream open for an extended period of time. Since the effort required to request notifications is tiny compared to the time, memory, and bandwidth consumed attempting to serve the notifications, servers implementing Events Query have increased susceptibility to Denial-of-Service attacks. Servers ought to ignore, coalesce, or reject egregious subscription requests, such as repeated requests from the same origin within a short interval of time.

13. IANA Considerations

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

13.1. 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

13.2. 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).

13.2.1. 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.

13.2.2. Initial Registry Contents

The initial contents of the HTTP Events Field Registry are:

Property Name	Structured-Type	Reference
duration	Integer or Decimal Item	Section 5.1

Table 2: List of HTTP Events Field property name registrations

14. End User Considerations

// "The mission of the IETF is to produce high quality, relevant
// technical and engineering documents that influence the way people
// design, use, and manage the Internet in such a way as to make the
// Internet work better" [RFC3935]. The mission statement further
// goes on to state, "We want the Internet to be useful for
// communities that share our commitment to openness and fairness.
// We embrace technical concepts such as decentralized control, edge-
// user empowerment and sharing of resources, because those concepts
// resonate with the core values of the IETF community." [RFC3935].

// The Internet Architecture Board says that the "Internet is for end
// users" and "argues that the IETF should favor their interests over
// those of other parties" [RFC8890]. It logically follows that
// technical documents produced by the IETF ought to consider their
// impact on the Internet end user. For this reason, I suggest that
// specifications include a considerations section where authors
// assess the technical decisions and trade-off in their proposals
// based on their ramifications on the interests of end users.

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 the content and the 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 for 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.

15. References

15.1. 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-12, 29 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-safe-method-w-body-12>.

[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>.

15.2. Informative References

[COMET] Russell, A., "Comet: Low Latency Data for the Browser", Infrequently Noted, March 2006, <https://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/>.

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

[HTML] WHATWG, "HTML", WHATWG Living Standard, April 2026, <https://html.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>.

[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>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/rfc/rfc2046>.

[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>.

[RFC7464] Williams, N., "JavaScript Object Notation (JSON) Text Sequences", RFC 7464, DOI 10.17487/RFC7464, February 2015, <https://www.rfc-editor.org/rfc/rfc7464>.

[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>.

[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. Examples

Some examples used in this specification are consolidated below. Chunks have been omitted for clarity.

A.1. Representation and Notifications

The following example illustrates a complete request and response for representation and notifications transferred with the multipart/mixed media-type as described in Section 10: Representation.

A.1.1. Request

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: multipart/mixed
Content-Type: example/events-query

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

Figure 18: Request for Representation and Notifications

A.1.2. Response

```
HTTP/1.1 200 OK
Date: Thu, 02 Jan 2025 10:10:10 GMT
Accept-Query: example/events-query
Events: duration=600
Incremental: ?1
Transfer-Encoding: chunked
Content-Type: multipart/mixed; boundary="THIS_STRING_SEPARATES"

--THIS_STRING_SEPARATES
Content-Length: 14                              | Representation
Content-Type: text/plain                        |
                                                |
Hello World!                                    |

--THIS_STRING_SEPARATES
Content-Length: 31                              | Notification
Content-Type: example/event-notification        |
                                                |
published: 2025-01-02T10:11:12.345Z             |
event-id: 456                                   |
type: update                                    |

--THIS_STRING_SEPARATES
Content-Length: 31                              | Notification
Content-Type: example/event-notification        |
                                                |
published: 2025-01-02T11:12:13.456Z             |
event-id: 789                                   |
type: delete                                    |
--THIS_STRING_SEPARATES--
```

Figure 19: Response with Representation and Notifications

A.2. Notifications Stream

This following example illustrates complete request and response for JSON formatted notifications transferred with the application/json-seq media-type as described in Notifications Stream: Notifications Stream.

A.2.1. Request

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: application/json-seq
Content-Type: example/events-query
Events: duration=0

events:
  @context: activity+json
```

Figure 20: Request for JSON Notifications

A.2.2. Response

```
HTTP/1.1 200 OK
Date: Thu, 02 Jan 2025 10:10:10 GMT
Accept-Query: example/events-query
Events: duration=600
Incremental: ?1
Transfer-Encoding: chunked
Content-Type: application/json-seq

蠑桀                                                 | Notification
  "published": "2025-01-02T10:11:12.345Z",          |
  "event-id": 456,                                  |
  "type": "Update"                                  |
}                                                   |
蠑桀                                                 | Notification
  "published": "2025-01-02T11:12:13.456Z",          |
  "event-id": 789,                                  |
  "type": "delete"                                  |
}                                                   |
```

Figure 21: Response with JSON Notifications

Acknowledgments

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

Index

Author’s Address

Rahul Gupta
Email: cxres+ietf@protonmail.com