HTTP Events Query

Rahul Gupta

HTTPAPI, IETF 123

In Memoriam

Josh Cohen

Terminology and Core Concepts

- Event
- Observation
- Notification
- Subscription

Terminology and Core Concepts

- **Event**

  the instantaneous effect of the termination of invocation of an operation on an object of interest

- Observation
- Notification
- Subscription

Terminology and Core Concepts

- Event
- **Observation**

  detection of the occurrence of an event by an external entity. Observers can be:
  - Origin servers
  - Intermediaries
- Notification
- Subscription

Terminology and Core Concepts

- Event
- Observation
- **Notification**

  information transferred by an observer, in our case, a server, upon an event or series of contiguous events
  - Extends “information hiding” to the temporal dimension
- Subscription

Terminology and Core Concepts

- Event
- Observation
- Notification
- **Subscription**

  act of requesting notifications from an observer

Design Goals

- Add notifications to any resource
  - Discourages (not prohibits) end-points
- NOT Switch protocols for notifications
- Fetch representation and notifications in a single request
- Content Negotiation
- Reliable and In-order notifications
- Allow intermediaries to participate

Events Query

- Uses the QUERY method to request notifications
- Defines an abstract data model for the request
- Defines an “Events” header field
- Two response modes:
  - Long Polling
  - Streaming

What about Per Resource Events (PREP)

- GET transfers representation of state [RFC9110]
  - Semantics had to be modified, à la “Range” requests
- Required a discovery mechanism to be defined
  i.e. an additional response header
- Structured headers not suitable for content negotiation of multiple concerns in a single request

QUERY Method

- QUERY is performed over *“some set of data at the resource”*
- `Accept-Query` header field advertises that server can accept an **Events Query**
- Request body can describe the requested form of representation and notifications

Discovery

Request:

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

Response:

```
HTTP/1.1 200 OK
Accept-Query: example/events-request+json
```

Long Polling: Request

```
QUERY /missing/John_Doe HTTP/1.1
Host: example.org
Content-Type: example/events-request+json

{}
```

Long Polling: Response

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

Long Polling: Response

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

Event-ID: 456
Type: Update
```

Event Streaming: Request

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

{
  events: {
    Accept: "example/event-response"
  }
}
```

Event Streaming: Response

```
HTTP/1.1 200 OK
Accept-Query: example/events-request+json
Content-Type: application/http
Transfer Encoding: Chunked
Events: duration=1200
Incremental: ?1
```

Event Streaming: Response

```
HTTP/1.1 200 OK
Accept-Query: example/events-request+json
Content-Type: application/http
Transfer Encoding: Chunked
Events: duration=1200
Incremental: ?1

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

Event Streaming: Response /2

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

Streaming with Representation: Request

```
QUERY /foo HTTP/1.1
Host: example.org
Accept: application/http
Content-Type: example/events-request+json
Events: duration=1200

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

Streaming with Representation: 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!                             |
```

Streaming with Representation: Response /2

```
HTTP/1.1 200 OK                                | Notification
Content-Type: example/event-response           |
Content-Length: 31                             |
                                               |
Event-ID: 567                                  |
Type: Update                                   |
```

Streaming with Representation: Response /2

```
HTTP/1.1 200 OK                                | Notification
Content-Type: example/event-response           |
Content-Length: 31                             |
                                               |
Event-ID: 567                                  |
Type: Update                                   |

HTTP/1.1 200 OK                                | Notification
Content-Type: example/event-response           |
Content-Length: 31                             |
                                               |
Event-ID: 678                                  |
Type: Delete                                   |
```

Usage: Fetch Request

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

Usage: Fetch Response

```
const splitRes = splitHTTPResponseStream(response);
// split into iterable of representation and
// notifications

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

for await (const notification of splitRes) {
  // do something with a notification
}
```

End Users Considerations

- Two classes of end users
  - Consumers are numerous and greedy
  - Publishers are few and bear costs
- Per resource notifications ⇒ filtering
- Servers should have control
  - Can reject requests
  - Can charge for (better) notifications
- Intermediaries: Scaling v/s Consolidation


Thank You
questions
interest
comments
feedback
opinions
concerns