Workgroup: Network Working Group
Internet-Draft: draft-dunglas-mercure-07
Published: 8 July 2020
Intended Status: Standards Track
Expires: 9 January 2021
Authors: K. Dunglas
Les-Tilleuls.coop

The Mercure Protocol

Abstract

Mercure is a protocol enabling the pushing of data updates to web browsers and other HTTP clients in a fast, reliable and battery-efficient way. It is especially useful for publishing real-time updates of resources served through web APIs to web and mobile apps.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 9 January 2021.

Table of Contents

1. Terminology

The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL**, when they appear in this document, are to be interpreted as described in [RFC2119].

- Topic: The unit to which one can subscribe to changes. The topic **SHOULD** be identified by an IRI [RFC3987]. Using an HTTPS [RFC7230] or HTTP [RFC7230] URI [RFC3986] is **RECOMMENDED**.

- Update: The message containing the updated version of the topic. An update can be marked as private, consequently, it must be dispatched only to subscribers allowed to receive it.

- Topic selector: An expression matching one or several topics.

- Publisher: An owner of a topic. Notifies the hub when the topic feed has been updated. As in almost all pubsub systems, the publisher is unaware of the subscribers, if any. Other pubsub systems might call the publisher the "source". Typically a website or a web API, but can also be a web browser.

- Subscriber: A client application that subscribes to real-time updates of topics using topic selectors. Typically a web or a mobile application, but can also be a server.

- Subscription: A topic selector used by a subscriber to receive updates. A single subscriber can have several subscriptions, when it provides several topic selectors.

- Hub: A server that handles subscription requests and distributes the content to subscribers when the corresponding topics have been updated. Any hub **MAY** implement its own policies on who can use it.

2. Discovery

The discovery mechanism aims at identifying at least 2 URLs.

1. The URL of one or more hubs designated by the publisher.

2. The canonical URL for the topic to which subscribers are expected to use for subscriptions.

The URL of the hub **MUST** be the "well-known" [RFC5785] fixed path /.well-known/mercure.

If the publisher is a server, it **SHOULD** advertise the URL of one or more hubs to the subscriber, allowing it to receive live updates when topics are updated. If more than one hub URL is specified, the

publisher **MUST** notifies each hub, so the subscriber **MAY** subscribe to one or more of them.

Note: Publishers may wish to advertise and publish to more than one hub for fault tolerance and redundancy. If one hub fails to propagate an update to the document, then using multiple independent hub is a way to increase the likelihood of delivery to subscribers. As such, subscribers may subscribe to one or more of the advertised hubs.

The publisher **SHOULD** include at least one Link Header [RFC5988] with rel=mercure (a hub link header). The target URL of these links **MUST** be a hub implementing the Mercure protocol.

The publisher **MAY** provide the following target attributes in the Link Headers:

* last-event-id: the identifier of the last event dispatched by the publisher at the time of the generation of this resource. If provided, it **MUST** be passed to the hub through a query parameter called Last-Event-ID and will be used to ensure that possible updates having been made between the resource generation by the server and the connection to the hub are not lost. See Section 7.

* content-type: the content type of the updates that will be pushed by the hub. If omitted, the subscriber **MUST** assume that the content type will be the same as that of the original resource. Setting the content-type attribute is especially useful to hint that partial updates will be pushed, using formats such as JSON Patch [RFC6902] or JSON Merge Patch [RFC7386].

* key-set: the URL of the key set to use to decrypt updates, encoded in the JWK set format (JSON Web Key Set) [RFC7517]. See Section 10. As this key set will contain a secret key, the publisher must ensure that only the subscriber can access to this URL. To do so, the authorization mechanism (see Section 6) can be reused.

All these attributes are optional.

The publisher **MAY** also include one Link Header [RFC5988] with rel=self (the self link header). It **SHOULD** contain the canonical URL for the topic to which subscribers are expected to use for subscriptions. If the Link with rel=self is omitted, the current URL of the resource **MUST** be used as a fallback.

Minimal example:

```
GET /books/foo HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-type: application/ld+json
Link: <https://example.com/.well-known/mercure>; rel="mercure"

{"@id": "/books/foo", "foo": "bar"}
```

Links embedded in HTML or XML documents as defined in the WebSub recommendation [W3C.REC-websub-20180123] **MAY** also be supported by subscribers. If both a header and an embedded link are provided, the header **MUST** be preferred.

2.1. Content Negotiation

For practical purposes, it is important that the rel=self URL only offers a single representation. As the hub has no way of knowing what Media Type ([RFC6838]) or language may have been requested by the subscriber upon discovery, it would not be able to deliver the content using the appropriate representation of the document.

It is, however, possible to perform content negotiation by returning an appropriate rel=self URL according to the HTTP headers used in the initial discovery request. For example, a request to /books/foo with an Accept header containing application/ld+json could return a rel=self value of /books/foo.jsonld.

The example below illustrates how a topic URL can return different Link headers depending on the Accept header that was sent.

```
GET /books/foo HTTP/1.1
Host: example.com
Accept: application/ld+json

HTTP/1.1 200 OK
Content-type: application/ld+json
Link: </books/foo.jsonld>; rel="self"
Link: <https://example.com/.well-known/mercure>; rel="mercure"

{"@id": "/books/foo", "foo": "bar"}
```

```
GET /books/foo HTTP/1.1
Host: example.com
Accept: text/html

HTTP/1.1 200 OK
Content-type: text/html
Link: </books/foo.html>; rel="self"
Link: <https://example.com/.well-known/mercure>; rel="mercure"

<!doctype html>
<title>foo: bar</title>
```

Similarly, the technique can also be used to return a different rel=self URL depending on the language requested by the Accept-Language header.

```
GET /books/foo HTTP/1.1
Host: example.com
Accept-Language: fr-FR

HTTP/1.1 200 OK
Content-type: application/ld+json
Content-Language: fr-FR
Link: </books/foo-fr-FR.jsonld>; rel="self"
Link: <https://example.com/.well-known/mercure>; rel="mercure"

{"@id": "/books/foo", "foo": "bar", "@context": {"@language": "fr-FR"}}
```

3. Topic Selectors

A topic selector is an expression intended to be matched by one or several topics. A topic selector can also be used to match other topic selectors for authorization purposes. See Section 6.

A topic selector can be any string including URI Templates [RFC6570] and the reserved string * that matches all topics. It is **RECOMMENDED** to use URI Templates or the reserved string * as topic selectors.

Note: URLs and IRIs are valid URI templates.

To determine if a string matches a selector, the following steps must be followed:

1. If the topic selector is * then the string matches the selector.

2. If the topic selector and the string are exactly the same, the string matches the selector. This characteristic allows to compare a URI Template with another one.

3. If the topic selector is a valid URI Template, and that the string matches this URI Template, the string matches the selector.

4. Otherwise the string does not match the selector.

4. Subscription

The subscriber subscribes to a URL exposed by a hub to receive updates from one or many topics. To subscribe to updates, the client opens an HTTPS connection following the Server-Sent Events specification [W3C.REC-eventsource-20150203] to the hub's subscription URL advertised by the publisher. The GET HTTP method must be used. The connection **SHOULD** use HTTP version 2 or superior to leverage multiplexing and other performance-oriented related features provided by these versions.

The subscriber specifies the list of topics to get updates from by using one or several query parameters named topic. The topic query parameters **MUST** contain topic selectors. See Section 3.

The protocol doesn't specify the maximum number of topic parameters that can be sent, but the hub **MAY** apply an arbitrary limit. A subscription is created for every provided topic parameter. See Section 8.1.

The EventSource JavaScript interface **MAY** be used to establish the connection. Any other appropriate mechanism including, but not limited to, readable streams [W3C.NOTE-streams-api-20161129] and XMLHttpRequest (used by popular polyfills) **MAY** also be used.

The hub sends to the subscriber updates for topics matching the provided topic selectors.

If an update is marked as private, the hub **MUST NOT** dispatch it to subscribers not authorized to receive it. See Section 6.

The hub **MUST** send these updates as text/event-stream compliant events [!@W3C.REC-eventsource-20150203].

The data property **MUST** contain the new version of the topic. It can be the full resource, or a partial update by using formats such as JSON Patch [RFC6902] or JSON Merge Patch [RFC7386].

All other properties defined in the Server-Sent Events specification **MAY** be used and **MUST** be supported by hubs.

The resource **SHOULD** be represented in a format with hypermedia capabilities such as JSON-LD [W3C.REC-json-ld-20140116], Atom [RFC4287], XML [W3C.REC-xml-20081126] or HTML [W3C.REC-html52-20171214].

Web Linking [RFC5988] **SHOULD** be used to indicate the IRI of the resource sent in the event. When using Atom, XML or HTML as the serialization format for the resource, the document **SHOULD** contain a link element with a self relation containing the IRI of the resource. When using JSON-LD, the document **SHOULD** contain an @id property containing the IRI of the resource.

Example:

```
// The subscriber subscribes to updates
// for the https://example.com/foo topic, the bar topic,
// and to any topic matching https://example.com/books/{name}
const url = new URL('https://example.com/.well-known/mercure');
url.searchParams.append('topic', 'https://example.com/foo');
url.searchParams.append('topic', 'bar');
url.searchParams.append('topic', 'https://example.com/bar/{id}');

const eventSource = new EventSource(url);

// The callback will be called every time an update is published
eventSource.onmessage = function ({data}) {
    console.log(data);
};
```

The hub **MAY** require subscribers and publishers to be authenticated, and **MAY** apply extra authorization rules not defined in this specification.

5. Publication

The publisher sends updates by issuing POST HTTPS requests on the hub URL. When it receives an update, the hub dispatches it to subscribers using the established server-sent events connections.

The hub **MAY** also dispatch this update using other protocols such as WebSub [W3C.REC-websub-20180123] or ActivityPub [W3C.REC-activitypub-20180123].

An application **CAN** send events directly to subscribers without using an external hub server, if it is able to do so. In this case, it **MAY NOT** implement the endpoint to publish updates.

The request **MUST** be encoded using the application/x-www-form-urlencoded format [W3C.REC-html52-20171214] and contains the following name-value tuples:

- topic: The identifiers of the updated topic. It is **RECOMMENDED** to use an IRI as identifier. If this name is present several times, the first occurrence is considered to be the canonical IRI of the topic, and other ones are considered to be alternate IRIs. The hub **MUST** dispatch this update to subscribers that are subscribed to both canonical or alternate IRIs.

- data (optional): the content of the new version of this topic.

- private (optional): if this name is set, the update **MUST NOT** be dispatched to subscribers not authorized to receive it. See Section 6. It is recommended to set the value to on but it **CAN** contain any value including an empty string.

- id (optional): the topic's revision identifier: it will be used as the SSE's id property. The provided id **MUST NOT** start with the # character. The provided id **SHOULD** be a valid IRI. If omitted, the hub **MUST** generate a valid IRI [RFC3987]. An UUID [RFC4122] or a DID **MAY** be used. Alternatively the hub **MAY** generate a relative URI composed of a fragment (starting with #). This is convenient to return an offset or a sequence that is unique for this hub. Even if provided, the hub **MAY** ignore the id provided by the client and generate its own id.

- type (optional): the SSE's event property (a specific event type).

- retry (optional): the SSE's retry property (the reconnection time).

In the event of success, the HTTP response's body **MUST** be the id associated to this update generated by the hub and a success HTTP status code **MUST** be returned. The publisher **MUST** be authorized to publish updates. See Section 6.

Example:

```
POST /.well-known/mercure HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Bearer [snip]

topic=https://example.com/foo&data=the%20content

HTTP/1.1 200 OK
Content-type: text/plain

urn:uuid:e1ee88e2-532a-4d6f-ba70-f0f8bd584022
```

6. Authorization

To ensure that they are authorized, both publishers and subscribers must present a valid JWS [RFC7515] in compact serialization to the hub. This JWS **SHOULD** be short-lived, especially if the subscriber is a web browser. A different key **MAY** be used to sign subscribers' and publishers' tokens.

Two mechanisms are defined to present the JWS to the hub:

- using an Authorization HTTP header
- using a cookie

If the publisher or the subscriber is not a web browser, it **SHOULD** use an Authorization HTTP header. This Authorization header **MUST** contain the string Bearer followed by the JWS. The hub will check that the JWS conforms to the rules (defined later) ensuring that the client is authorized to publish or subscribe to updates.

By the EventSource specification [W3C.REC-eventsource-20150203], web browsers can not set custom HTTP headers for such connections, and they can only be established using the GET HTTP method. However, cookies are supported and can be included even in cross-domain requests if the CORS credentials are set:

If the publisher or the subscriber is a web browser, it **SHOULD** send a cookie called mercureAuthorization containing the JWS when connecting to the hub.

Whenever possible, the mercureAuthorization cookie **SHOULD** be set during discovery (see Section 2) to improve the overall security. Consequently, if the cookie is set during the discovery, both the publisher and the hub have to share the same second level domain. The Domain attribute **MAY** be used to allow the publisher and the hub to use different subdomains. See Section 2.

The cookie **SHOULD** have the Secure, HttpOnly and SameSite attributes set. The cookie's Path attribute **SHOULD** also be set to the hub's URL. See Section 12.

When using authorization mechanisms, the connection **MUST** use an encryption layer such as HTTPS.

If both an Authorization HTTP header and a cookie named mercureAuthorization are presented by the client, the cookie **MUST** be ignored. If the client tries to execute an operation it is not allowed to, a 403 HTTP status code **SHOULD** be returned.

6.1. Publishers

Publishers **MUST** be authorized to dispatch updates to the hub, and **MUST** prove that they are authorized to send updates for the specified topics.

To be allowed to publish an update, the JWS presented by the publisher **MUST** contain a claim called mercure, and this claim **MUST** contain a publish key. mercure.publish contains an array of topic selectors. See Section 3.

If mercure.publish:

- is not defined, then the publisher **MUST NOT** be authorized to dispatch any update

- contains an empty array, the publisher **MUST NOT** be authorized to publish private updates, but can publish public updates for all topics.

Otherwise, the hub **MUST** check that every topics of the update to dispatch matches at least one of the topic selectors contained in mercure.publish.

If the publisher is not authorized for all the topics of an update, the hub **MUST NOT** dispatch the update (even if some topics in the list are allowed) and **MUST** return a 403 HTTP status code.

6.2. Subscribers

To receive updates marked as private, a subscriber **MUST** prove that it is authorized for at least one of the topics of this update. If the subscriber is not authorized to receive an update marked as private, it **MUST NOT** receive it.

To receive updates marked as private, the JWS presented by the subscriber **MUST** have a claim named mercure with a key named subscribe that contains an array of topic selectors. See Section 3.

The hub **MUST** check that at least one topic of the update to dispatch (*canonical* or *alternate*) matches at least one topic selector provided in mercure.subscribe.

This behavior makes it possible to subscribe to several topics using URI templates while guaranteeing that only authorized subscribers will receive updates marked as private (even if their canonical topics are matched by these templates).

Let's say that a subscriber wants to receive updates concerning all *book* resources it has access to. The subscriber can use the topic selector https://example.com/books/{id} as value of the topic query parameter. Adding this same URI template to the mercure.subscribe claim of the JWS presented by the subscriber to the hub would allow this subscriber to receive all updates for all book resources. It is not what we want here: this subscriber is only authorized to access **some** of these resources.

To solve this problem, the mercure.subscribe claim could contain a topic selector such as: https://example.com/users/foo/{?topic}.

The publisher could then take advantage of the previously described behavior by publishing a private update having https://example.com/books/1 as canonical topic and https://example.com/users/foo/?topic=https%3A%2F%2Fexample.com%2Fbooks%2F1 as alternate topic:

```
POST /.well-known/mercure HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Bearer [snip]

topic=https://example.com/books/1&topic=https://example.com/users/foo/?topic=https%3A%2F%2Fex
```

The subscriber is subscribed to https://example.com/books/{id} that is matched by the canonical topic of the update. This canonical topic isn't matched by the topic selector provided in its JWS claim mercure.subscribe. However, an alternate topic of the update, https://example.com/users/foo/?topic=https%3A%2F%2Fexample.com%2Fa-random-topic, is matched by it. Consequently, this private update will be received by this subscriber, while other updates having a canonical topic matched by the selector provided in a topic query parameter but not matched by selectors in the mercure.subscribe claim will not.

6.3. Payload

The mercure claim of the JWS **CAN** also contain user-defined values under the payload key. This JSON document will be attached to the

subscription and made available in subscription events. See Section 8.1.

For instance, mercure.payload can contain the user ID of the subscriber, a list of groups it belongs to, or its IP address. Storing data in mercure.payload is a convenient way to share data related to one subscriber to other subscribers.

7. Reconnection, State Reconciliation and Event Sourcing

The protocol allows to reconciliate states after a reconnection. It can also be used to implement an Event store.

To allow re-establishment in case of connection lost, events dispatched by the hub **MUST** include an id property. The value contained in this id property **SHOULD** be an IRI [RFC3987]. An UUID [RFC4122] or a DID **MAY** be used.

According to the server-sent events specification, in case of connection lost the subscriber will try to automatically re-connect. During the re-connection, the subscriber **MUST** send the last received event id in a Last-Event-ID HTTP header.

In order to fetch any update dispatched between the initial resource generation by the publisher and the connection to the hub, the subscriber **MUST** send the event id provided during the discovery as a Last-Event-ID header or query parameter. See Section 2.

EventSource implementations may not allow to set HTTP headers during the first connection (before a reconnection) and implementations in web browsers don't allow to set it.

To work around this problem, the hub **MUST** also allow to pass the last event id in a query parameter named Last-Event-ID.

If both the Last-Event-ID HTTP header and the query parameter are present, the HTTP header **MUST** take precedence.

If the Last-Event-ID HTTP header or query parameter exists, the hub **SHOULD** send all events published following the one bearing this identifier to the subscriber.

The reserved value earliest can be used to hint the hub to send all updates it has for the subscribed topics. According to its own policy, the hub **MAY** or **MAY NOT** fulfil this request.

The hub **MAY** discard some events for operational reasons. When the request contains a Last-Event-ID HTTP header or query parameter the hub **MUST** set a Last-Event-ID header on the HTTP response. The value of the Last-Event-ID response header **MUST** be the id of the event

preceding the first one sent to the subscriber, or the reserved value earliest if there is no preceding event (it happens when the hub history is empty, when the subscriber requests the earliest event or when the subscriber requests an event that doesn't exist).

The subscriber **SHOULD NOT** assume that no events will be lost (it may happen, for example if the hub stores only a limited number of events in its history). In some cases (for instance when sending partial updates in the JSON Patch [RFC6902] format, or when using the hub as an event store), updates lost can cause data lost.

To detect if a data lost ocurred, the subscriber **CAN** compare the value of the Last-Event-ID response HTTP header with the Last-Event-ID it requested. In case of data lost, the subscriber **SHOULD** re-fetch the original topic.

Note: Native EventSource implementations don't give access to headers associated with the HTTP response, however polyfills and server-sent events clients in most programming languages allow it.

The hub **CAN** also specify the reconnection time using the retry key, as specified in the server-sent events format.

8. Active Subscriptions

Mercure provides a mechanism to track active subscriptions. If the hub support this optional set of features, updates will be published when a subscription is created, or terminated, and a web API exposes the list of active subscriptions.

Variables are templated and expanded in accordance with [RFC6570].

8.1. Subscription Events

If the hub supports the active subscriptions feature, it **MUST** publish an update when a subscription is created or terminated. If this feature is implemented by the hub, an update **MUST** be dispatched every time a subscription is created or terminated.

The topic of these updates **MUST** be an expansion of /.well-known/mercure/subscriptions/{topic}/{subscriber}. {topic} is the topic selector used for this subscription and {subscriber} is an unique identifier for the subscriber.

Note: Because it is recommended to use URI Templates and IRIs for the {topic} and {subscriber} variables, values will usually contain the :, /, { and } characters. Per [RFC6570], these characters are reserved. They **MUST** be percent encoded during the expansion process.

If a subscriber has several subscriptions, it **SHOULD** be identified by the same identifier. {subscriber} **SHOULD** be an IRI [RFC3987]. An UUID [RFC4122] or a DID **MAY** be used.

The content of the update **MUST** be a JSON-LD [W3C.REC-json-ld-20140116] document containing at least the following properties:

- *@context: the fixed value https://mercure.rocks/. @context can be omitted if already defined in a parent node. See Section 9.
- *id: the identifier of this update, it **MUST** be the same value as the subscription update's topic
- *type: the fixed value Subscription
- *topic: the topic selector used of this subscription
- *subscriber: the topic identifier of the subscriber. It **SHOULD** be an IRI.
- *active: true when the subscription is active, and false when it is terminated
- *payload (optional): the content of mercure.payload in the subscriber's JWS (see Section 6)

The JSON-LD document **MAY** contain other properties.

In order to only allow authorized subscribers to receive subscription events, the subscription update **MUST** be marked as private.

Example:

```
{
   "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D/urn%3
   "type": "Subscription",
   "topic": "https://example.com/{selector}",
   "subscriber": "urn:uuid:bb3de268-05b0-4c65-b44e-8f9acefc29d6",
   "active": true,
   "payload": {"foo": "bar"}
}
```

8.2. Subscription API

If the hub supports subscription events (see Section 8.1), it **SHOULD** also expose active subscriptions through a web API.

For instance, subscribers interested in maintaining a list of active subscriptions can call the web API to retrieve them, and then use subscription events (see Section 8.1) to keep it up to date.

The web API **MUST** expose endpoints following these patterns:

* /.well-known/mercure/subscriptions: the collection of subscriptions

* /.well-known/mercure/subscriptions/{topic}: the collection of subscriptions for the given topic selector

* /.well-known/mercure/subscriptions/{topic}/{subscriber}: a specific subscription

To access to the URLs exposed by the web API, clients **MUST** be authorized according to the rules defined in Section 6. The requested URL **MUST** match at least one of the topic selectors provided in the mercure.subscribe key of the JWS.

The web API **MUST** set the Content-Type HTTP header to application/ld+json.

URLs returning a single subscription (following the pattern /.well-known/mercure/subscriptions/{topic}/{subscriber}) **MUST** expose the same JSON-LD document as described in Section 8.1. If the requested subscription does not exist, a 404 status code **MUST** be returned.

If the requested subscription isn't active anymore, the hub can either return the JSON-LD document with the active property set to false or return a 404 status code. Accordingly, collection endpoints **CAN** return terminated connections with the active property set to false or omit them.

Collection endpoints **MUST** return JSON-LD documents containing at least the following properties:

* @context: the fixed value https://mercure.rocks/. @context can be omitted if already defined in a parent node. See Section 9.

* id: the URL used to retrieve the document

* type: the fixed value Subscriptions

* subscriptions: an array of subscription documents as described in Section 8.1

In addition, all endpoints **MUST** set the lastEventID property at the root of the returned JSON-LD document:

* lastEventID: the identifier of the last event dispatched by the hub at the time of this request (see Section 7). The value **MUST** be earliest if no events have been dispatched yet. The value of this property **SHOULD** be passed back to the hub when subscribing to subscription events to prevent data loss.

As data returned by this web API is volatile, clients **SHOULD** validate that a response coming from cache is still valid before using it.

Examples:

```
GET /.well-known/mercure/subscriptions HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-type: application/ld+json
Link: <https://example.com/.well-known/mercure>; rel="mercure"
ETag: urn:uuid:5e94c686-2c0b-4f9b-958c-92ccc3bbb4eb
Cache-control: must-revalidate

{
   "@context": "https://mercure.rocks/",
   "id": "/.well-known/mercure/subscriptions",
   "type": "Subscriptions",
   "lastEventID": "urn:uuid:5e94c686-2c0b-4f9b-958c-92ccc3bbb4eb",
   "subscriptions": [
      {
         "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D
         "type": "Subscription",
         "topic": "https://example.com/{selector}",
         "subscriber": "urn:uuid:bb3de268-05b0-4c65-b44e-8f9acefc29d6",
         "active": true,
         "payload": {"foo": "bar"}
      },
      {
         "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2Fa-topic/urn%3A
         "type": "Subscription",
         "topic": "https://example.com/a-topic",
         "subscriber": "urn:uuid:1e0cba4c-4bcd-44f0-ae8a-7b76f7ef1280",
         "active": true,
         "payload": {"baz": "bat"}
      },
      {
         "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D
         "type": "Subscription",
         "topic": "https://example.com/{selector}",
         "subscriber": "urn:uuid:a6c49794-5f74-4723-999c-3a7e33e51d49",
         "active": true,
         "payload": {"foo": "bap"}
      }
   ]
}
```

```
GET /.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-type: application/ld+json
Link: <https://example.com/.well-known/mercure>; rel="mercure"
ETag: urn:uuid:5e94c686-2c0b-4f9b-958c-92ccc3bbb4eb
Cache-control: must-revalidate

{
   "@context": "https://mercure.rocks/",
   "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D",
   "type": "Subscriptions",
   "lastEventID": "urn:uuid:5e94c686-2c0b-4f9b-958c-92ccc3bbb4eb",
   "subscriptions": [
      {
         "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D
         "type": "Subscription",
         "topic": "https://example.com/{selector}",
         "subscriber": "urn:uuid:bb3de268-05b0-4c65-b44e-8f9acefc29d6",
         "active": true,
         "payload": {"foo": "bar"}
      },
      {
         "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D
         "type": "Subscription",
         "topic": "https://example.com/{selector}",
         "subscriber": "urn:uuid:a6c49794-5f74-4723-999c-3a7e33e51d49",
         "active": true,
         "payload": {"foo": "bap"}
      }
   ]
}
```

```
GET /.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D/urn%3Auuid%
Host: example.com

HTTP/1.1 200 OK
Content-type: application/ld+json
Link: <https://example.com/.well-known/mercure>; rel="mercure"
ETag: urn:uuid:5e94c686-2c0b-4f9b-958c-92ccc3bbb4eb
Cache-control: must-revalidate

{
   "@context": "https://mercure.rocks/",
   "id": "/.well-known/mercure/subscriptions/https%3A%2F%2Fexample.com%2F%7Bselector%7D/urn%3
   "type": "Subscription",
   "topic": "https://example.com/{selector}",
   "subscriber": "urn:uuid:bb3de268-05b0-4c65-b44e-8f9acefc29d6",
   "active": true,
   "payload": {"foo": "bar"},
   "lastEventID": "urn:uuid:5e94c686-2c0b-4f9b-958c-92ccc3bbb4eb"
}
```

9. JSON-LD Context

The JSON-LD context available at https://mercure.rocks is the following:

```
{
"@context": {
   "@vocab": "_:",
   "mercure": "https://mercure.rocks/",
   "id": "@id",
   "type": "@type",
   "Subscription": "mercure:Subscription",
   "Subscriptions": "mercure:Subscriptions",
   "subscriptions": "mercure:subscriptions",
   "topic": "mercure:topic",
   "subscriber": "mercure:subscriber",
   "active": "mercure:active",
   "payload": "mercure:payload",
   "lastEventID": "mercure:lastEventID"
}
```

10. Encryption

Using HTTPS does not prevent the hub from accessing the update's content. Depending of the intended privacy of information contained

in the update, it **MAY** be necessary to prevent eavesdropping by the hub.

To make sure that the message content can not be read by the hub, the publisher **MAY** encrypt the message before sending it to the hub. The publisher **SHOULD** use JSON Web Encryption [RFC7516] to encrypt the update content. The publisher **MAY** provide the URL pointing to the relevant encryption key(s) in the key-set attribute of the Link HTTP header during the discovery. See Section 2. The key-set attribute **MUST** link to a key encoded using the JSON Web Key Set [RFC7517] format. Any other out-of-band mechanism **MAY** be used instead to share the key between the publisher and the subscriber.

Update encryption is considered a best practice to prevent mass surveillance. This is especially relevant if the hub is managed by an external provider.

11. IANA Considerations

11.1. Well-Known URIs Registry

A new "well-known" URI as described in Section 2 has been registered in the "Well-Known URIs" registry as described below:

- *URI Suffix: mercure
- *Change Controller: IETF
- *Specification document(s): This specification, Section 2
- *Related information: N/A

11.2. Link Relation Types Registry

A new "Link Relation Type" as described in Section 2 has been registered in the "Link Relation Type" registry with the following entry:

- *Relation Name: mercure
- *Description: The Mercure Hub to use to subscribe to updates of this resource.
- *Reference: This specification, Section 2

11.3. JSON Web Token (JWT) Registry

A new "JSON Web Token Claim" as described in Section 6 **will be** registered in the "JSON Web Token (JWT)" with the following entry:

*Claim Name: mercure

*Description: Mercure data.

*Reference: This specification, Section 6

12. Security Considerations

The confidentiality of the secret key(s) used to generate the JWSs is a primary concern. The secret key(s) **MUST** be stored securely. They **MUST** be revoked immediately in the event of compromission.

Possessing valid JWSs allows any client to subscribe, or to publish to the hub. Their confidentiality **MUST** therefore be ensured. To do so, JWSs **MUST** only be transmitted over secure connections.

Also, when the client is a web browser, the JWS **SHOULD** not be made accessible to JavaScript scripts for resilience against Cross-site Scripting (XSS) attacks. It's the main reason why, when the client is a web browser, using HttpOnly cookies as the authorization mechanism **SHOULD** always be preferred.

In the event of compromission, revoking JWSs before their expiration is often difficult. To that end, using short-lived tokens is strongly **RECOMMENDED**.

The publish endpoint of the hub may be targeted by Cross-Site Request Forgery (CSRF) attacks when the cookie-based authorization mechanism is used. Therefore, implementations supporting this mechanism **MUST** mitigate such attacks.

The first prevention method to implement is to set the mercureAuthorization cookie's SameSite attribute. However, some web browsers still not support this attribute and will remain vulnerable. Additionally, hub implementations **SHOULD** use the Origin and Referer HTTP headers set by web browsers to verify that the source origin matches the target origin. If none of these headers are available, the hub **SHOULD** discard the request.

CSRF prevention techniques, including those previously mentioned, are described in depth in OWASP's Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet.

13. Implementation Status

[RFC Editor Note: Please remove this entire seciton prior to publication as an RFC.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist. According to RFC 6982, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit."

13.1. Mercure.rocks Hub

Organization responsible for the implementation:

Dunglas Services SAS Les-Tilleuls.coop

Implementation Name and Details:

Mercure.rocks, available at https://mercure.rocks

Brief Description:

This is the reference implementation of the Mercure hub. It is written in Go and is optimized for performance.

Level of Maturity:

Widely used.

Coverage:

All the features of the protocol.

Version compatibility:

The implementation follows the latest draft.

Licensing:

All code is covered under the GNU Affero Public License version 3 or later.

Implementation Experience:

Used in production.

Contact Information:

Kévin Dunglas, kevin+mercure@dunglas.fr https://mercure.rocks

Interoperability:

Reported compatible with all major browsers and server-side tools.

13.2. Ilshidur/node-mercure

Implementation Name and Details:

Ilshidur/node-mercure, https://github.com/Ilshidur/node-mercure

Brief Description:

Hub and Publisher implemented in Node.

Level of Maturity:

Beta, not suitable for production.

Coverage:

All the features of the protocol except the subscription events.

Version compatibility:

The implementation currently follows the revision 5 of the draft.

Licensing:

All code is covered under the GNU Public License version 3 or later.

Contact Information:

https://github.com/Ilshidur/node-mercure

Interoperability:

Reported compatible with all major browsers and server-side tools.

13.3. Symfony

Implementation Name and Details:

Symfony Mercure Component, available at https://symfony.com/doc/current/components/mercure.html

Brief Description:

This a publisher library written in PHP. It also provides support for Mercure in the Symfony web framework.

Level of Maturity:

Widely used.

Coverage:

All the publisher features of the protocol.

Version compatibility:

The implementation follows the latest draft.

Licensing:

All code is covered under the MIT license.

Implementation Experience:

Used in production.

Contact Information:

https://symfony.com

Interoperability:

Reported compatible with the Mercure.rocks Hub.

13.4. API Platform

Implementation Name and Details:

API Platform, available at https://api-platform.com/docs/core/mercure/

Brief Description:

The API Platform framework, allows to create async APIs implementing the Mercure protocol and to generate clients for these APIs.

Level of Maturity:

Widely used.

Coverage:

All the publisher and consumer features of the protocol.

Version compatibility:

The implementation follows the latest draft.

Licensing:

All code is covered under the MIT license.

Implementation Experience:

Used in production.

Contact Information:

https://api-platform.com

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.5. Laravel Mercure Broadcaster

Implementation Name and Details:

Laravel Mercure Broadcaster, available at https://github.com/mvanduijker/laravel-mercure-broadcaster

Brief Description:

Laravel broadcaster for Mercure. Use the Mercure protocol as transport for Laravel Broadcast.

Level of Maturity:

Production

Coverage:

All the publisher features of the protocol.

Version compatibility:

The implementation currently follows the revision 5 of the draft. An open Pull Request adds support for the latest version of the draft.

Licensing:

All code is covered under the MIT license.

Implementation Experience:

Used in production.

Contact Information:

https://github.com/mvanduijker/laravel-mercure-broadcaster

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.6. dart_mercure

Implementation Name and Details:

dart*mercure, available at https://github.com/wallforfry/dart*mercure

Brief Description:

Publisher and Subscriber library for Dart / Flutter.

Level of Maturity:

Stable

Coverage:

All the publisher and subscriber features of the protocol.

Version compatibility:

The implementation follows the latest draft.

Licensing:

All code is covered under the BSD 2-Clause "Simplified" License.

Contact Information:

https://github.com/wallforfry/dart_mercure

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.7. pymercure

Implementation Name and Details:

pymercure, available at https://github.com/vitorluis/python-mercure

Brief Description:

Publisher and Subscriber library for Python.

Level of Maturity:

Alpha

Coverage:

All the publisher and subscriber features of the protocol.

Version compatibility:

The implementation currently follows the revision 5 of the draft. An open Pull Request adds support for the latest version of the draft.

Licensing:

All code is covered under the BSD 2-Clause "Simplified" License.

Contact Information:

https://github.com/vitorluis/python-mercure

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.8. Amphp Mercure Publisher

Implementation Name and Details:

Amphp Mercure Publisher, available at https://github.com/eislambey/amp-mercure-publisher

Brief Description:

Async Mercure publisher based on Amphp.

Level of Maturity:

Stable

Coverage:

All the publisher features of the protocol.

Version compatibility:

The implementation currently follows the revision 5 of the draft. An open Pull Request adds support for the latest version of the draft.

Licensing:

All code is covered under the MIT license.

Contact Information:

https://github.com/eislambey/amp-mercure-publisher

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.9. Java Library for Mercure

Implementation Name and Details:

Java Library for Mercure, available at https://github.com/vitorluis/java-mercure

Brief Description:

Java library to publish messages to a Mercure Hub!

Level of Maturity:

Alpha

Coverage:

All the publisher features of the protocol.

Version compatibility:

The implementation currently follows the revision 5 of the draft. An open Pull Request adds support for the latest version of the draft.

Licensing:

All code is covered under the MIT license.

Contact Information:

https://github.com/vitorluis/java-mercure

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.10. Yii 2 Mercure behavior

Implementation Name and Details:

Yii 2 Mercure behavior, available at https://github.com/bizley/mercure-behavior

Brief Description:

Yii 2 behavior to automatically publish updates to a Mercure hub.

Level of Maturity:

Stable

Coverage:

All the publisher features of the protocol.

Version compatibility:

The implementation currently follows the revision 5 of the draft.

Licensing:

All code is covered under the Apache License 2.0.

Contact Information:

https://github.com/bizley/mercure-behavior

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.11. GitHub Action for Mercure

Implementation Name and Details:

GitHub Action for Mercure, available at https://github.com/marketplace/actions/github-action-for-mercure

Brief Description:

Send a Mercure update when a GitHub event occurs.

Level of Maturity:

Stable

Coverage:

All the publisher features of the protocol.

Version compatibility:

The implementation currently follows the latest version of the draft.

Licensing:

All code is covered under the GNU Public License version 3 or later.

Contact Information:

https://github.com/Ilshidur/action-mercure

Interoperability:

Reported compatible with the reference implementation of the Mercure Hub.

13.12. Other Implementations

Other implementations can be found on GitHub: https://github.com/topics/mercure

14. Acknowledgements

Parts of this specification, especially Section 2 have been adapted from the WebSub recommendation [W3C.REC-websub-20180123]. The editor wish to thanks all the authors of this specification.

15. Normative References

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[W3C.REC-json-ld-20140116] Sporny, M., Kellogg, G., and M. Lanthaler, "JSON-LD 1.0", World Wide Web Consortium Recommendation REC-json-ld-20140116, 16 January 2014, <http://www.w3.org/TR/2014/REC-json-ld-20140116>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[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/info/rfc2119>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://www.rfc-editor.org/info/rfc5988>.

16. Informative References

[W3C.REC-eventsource-20150203] Hickson, I., "Server-Sent Events", World Wide Web Consortium Recommendation REC-eventsource-20150203, 3 February 2015, <http://www.w3.org/TR/2015/REC-eventsource-20150203>.

[W3C.NOTE-streams-api-20161129]
Moussa, F. and T. Yoshino, "Streams API", World Wide Web Consortium NOTE NOTE-streams-api-20161129, 29 November 2016, <https://www.w3.org/TR/2016/NOTE-streams-api-20161129>.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <https://www.rfc-editor.org/info/rfc4287>.

[RFC7386] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7386, DOI 10.17487/RFC7386, October 2014, <https://www.rfc-editor.org/info/rfc7386>.

[W3C.REC-websub-20180123] Genestoux, J. and A. Parecki, "WebSub", World Wide Web Consortium Recommendation REC-websub-20180123, 23 January 2018, <https://www.w3.org/TR/2018/REC-websub-20180123>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, 26 November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[W3C.REC-html52-20171214] Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, 14 December 2017, <https://www.w3.org/TR/2017/REC-html52-20171214>.

[W3C.REC-activitypub-20180123]
Webber, C. and J. Tallon, "ActivityPub", World Wide Web Consortium Recommendation REC-activitypub-20180123, 23 January 2018, <https://www.w3.org/TR/2018/REC-activitypub-20180123>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <https://www.rfc-editor.org/info/rfc6982>.

[RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <https://www.rfc-editor.org/info/rfc6902>.

[RFC4122]
Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

Author's Address

Kévin Dunglas
Les-Tilleuls.coop
82 rue Winston Churchill
59160 Lille
France

Email: kevin@les-tilleuls.coop