The Idempotency HTTP Header Field
draft-idempotency-header-01

Abstract

   The "HTTP" Idempotency request header field can be used to carry
   idempotency key in order to make non-idempotent "HTTP" methods such
   as "POST" or "PATCH" fault-tolerant.

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 May 27, 2021.

Table of Contents

1. Introduction

   Idempotence is the property of certain operations in mathematics and
   computer science whereby they can be applied multiple times without
   changing the result beyond the initial application. It does not
   matter if the operation is called only once, or 10s of times over.
   The result "SHOULD" be the same.

   Idempotency is important in building a fault-tolerant "HTTP API". An
   "HTTP" request method is considered "idempotent" if the intended
   effect on the server of multiple identical requests with that method
   is the same as the effect for a single such request. According to
   [RFC7231], "HTTP" methods "OPTIONS", "HEAD", "GET", "PUT" and
   "DELETE" are idempotent while methods "POST" and "PATCH" are not.

   Let's say a client of an "HTTP API" wants to create (or update) a
   resource using a "POST" method. Since "POST" is "NOT" an idempotent
   method, calling it multiple times can result in duplication or wrong
   updates. Consider a scenario where the client sent a "POST" request
   to the server, but it got a timeout. Following questions arise : Is
   the resource actually created (or updated)? Did the timeout occur
   during sending of the request, or when receiving of the response?

Can the client safely retry the request, or does it need to figure
out what happened in the first place? If "POST" had been an
idempotent method, such questions may not arise. Client would safely
retry a request until it actually gets a valid response from the
server.

For many use cases of "HTTP API", duplicate resource is a severe
problem from business perspective. For example, duplicate records
for requests involving any kind of money transfer "MUST NOT" be
allowed. In other cases, processing of duplicate webhook delivery is
not expected.

1.1. Notational Conventions

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.

This specification uses the Augmented Backus-Naur Form (ABNF)
notation of [RFC5234] and includes, by reference, the IMF-fixdate
rule as defined in Section 7.1.1.1 of [RFC7231].

The term "resource" is to be interpreted as defined in Section 2 of
   [RFC7231], that is identified by an URI. The term "resource server"
is to be interpreted as "origin server" as defined in Section 3 of
   [RFC7231].

2. The Idempotency HTTP Request Header Field

An idempotency key is a unique value generated by the client which
the resource server uses to recognize subsequent retries of the same
request. The "Idempotency-Key" HTTP request header field carries
this key.

2.1. Syntax

The "Idempotency-Key" request header field describes

```
Idempotency-Key      = idempotency-key-value

idempotency-key-value = opaque-value
opaque-value         = DQUOTE *idempotencyvalue DQUOTE
idempotencyvalue     = %x21 / %x23-7E / obs-text
      ; VCHAR except double quotes, plus obs-text
```

Clients MUST NOT include more than one "Idempotency-Key" header field in the same request.

The following example shows an idempotency key using "UUID" [RFC4122]:

Idempotency-Key: "8e03978e-40d5-43e8-bc93-6894a57f9324"

2.2. Uniqueness of Idempotency Key

The idempotency key that is supplied as part of every "POST" request MUST be unique and "MUST" not be reused with another request with a different request payload.

Uniqueness of the key "MUST" be defined by the resource owner and "MUST" be implemented by the clients of the resource server. It is "RECOMMENDED" that "UUID" [RFC4122] or a similar random identifier be used as an idempotency key.

2.3. Idempotency Key Validity and Expiry

The resource MAY enforce time based idempotency keys, thus, be able to purge or delete a key upon its expiry. The resource server SHOULD define such expiration policy and publish in related documentation.

2.4. Idempotency Fingerprint

An idempotency fingerprint MAY be used in conjunction with an idempotency key to determine the uniqueness of a request. Such a fingerprint is generated from request payload data by the resource server. An idempotency fingerprint generation algorithm MAY use one of the following or similar approaches to create a fingerprint.

o Checksum of the entire request payload.

o Checksum of selected element(s) in the request payload.

o Field value match for each field in the request payload.

o Field value match for selected element(s) in the request payload.

o Request digest/signature.

2.5. Responsibilities

Client

Clients of "HTTP API" requiring idempotency, SHOULD understand the
idempotency related requirements as published by the server and use
appropriate algorithm to generate idempotency keys.

For each request, client SHOULD

o Send a unique idempotency key in the HTTP "Idempotency-Key"
   request header field.

Resource Server

Resource server MUST publish idempotency related specification. This
specification MUST include expiration related policy if applicable.
Server is responsible for managing the lifecycle of the idempotency
key.

For each request, server SHOULD

o Identify idempotency key from the HTTP "Idempotency-Key" request
   header field.

o Generate idempotency fingerprint if required.

o Check for idempotency considering various scenarios including the
   ones described in section below.

2.6. Idempotency Enforcement Scenarios

o First time request (idempotency key or fingerprint has not been
   seen)

   The resource server SHOULD process the request normally and
   respond with an appropriate response and status code.

o Duplicate request (idempotency key or fingerprint has been seen)

   Retry

   The request was retried after the original request completed. The
   resource server MUST respond with the result of the previously
   completed operation, success or an error.

   Concurrent Request

   The request was retried before the original request completed.
   The resource server MUST respond with a resource conflict error.
   See Error Scenarios for details.

2.7. Error Scenarios

   If the "Idempotency-Key" request header is missing for a documented
   idempotent operation requiring this header, the resource server MUST
   reply with an "HTTP" "400" status code with body containing a link
   pointing to relevant documentation. Alternately, using the "HTTP"
   header "Link", the client can be informed about the error as shown
   below.

   HTTP/1.1 400 Bad Request
   Link: <https://developer.example.com/idempotency>;
     rel="describedby"; type="text/html"

   If there is an attempt to reuse an idempotency key with a different
   request payload, the resource server MUST reply with a "HTTP" "422"
   status code with body containing a link pointing to relevant
   documentation. The status code "422" is defined in Section 11.2 of
   [RFC4918]. The server can also inform the client by using the "HTTP"
   header "Link" as shown below.

   HTTP/1.1 422 Unprocessable Entity
   Link: <https://developer.example.com/idempotency>;
   rel="describedby"; type="text/html"

   If the request is retried, while the original request is still being
   processed, the resource server MUST reply with an "HTTP" "409" status
   code with body containing a link or the "HTTP" header "Link" pointing
   to the relevant documentation.

   HTTP/1.1 409 Conflict
   Link: <https://developer.example.com/idempotency>;
   rel="describedby"; type="text/html"

   For other errors, the resource MUST return the appropriate status
   code and error message.

3. IANA Considerations

3.1. The Idempotency-Key HTTP Request Header Field

   The "Idempotency-Key" request header should be added to the permanent
   registry of message header fields (see [RFC3864]), taking into
   account the guidelines given by HTTP/1.1 [RFC7231].

Header Field Name: Idempotency-Key

Applicable Protocol: Hypertext Transfer Protocol (HTTP)

Status: Standard

Authors:
        Jayadeba Jena
        Email: jjena@paypal.com

        Sanjay Dalal
        Email: sanjay.dalal@cal.berkeley.edu

Change controller: IETF

Specification document: this specification,
            Section 2 "The Idempotency HTTP Request Header Field"

4. Implementation Status

Note to RFC Editor: Please remove this section before publication.

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 [RFC7942].
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 7942, "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".

Organization: Stripe

o Description: Stripe uses custom HTTP header named "Idempotency-
   Key"

o   Reference: https://stripe.com/docs/idempotency

Organization: Adyen

o   Description: Adyen uses custom HTTP header named "Idempotency-Key"

o   Reference: https://docs.adyen.com/development-resources/api-idempotency/

Organization: Dwolla

o   Description: Dwolla uses custom HTTP header named "Idempotency-Key"

o   Reference: https://docs.dwolla.com/

Organization: Interledger

o   Description: Interledger uses custom HTTP header named "Idempotency-Key"

o   Reference: https://github.com/interledger/

Organization: WorldPay

o   Description: WorldPay uses custom HTTP header named "Idempotency-Key"

o   Reference: https://developer.worldpay.com/docs/wpg/idempotency

Organization: Yandex

o   Description: Yandex uses custom HTTP header named "Idempotency-Key"

o   Reference: https://cloud.yandex.com/docs/api-design-guide/concepts/idempotency

4.1. Implementing the Concept

This is a list of implementations that implement the general concept, but do so using different mechanisms:

Organization: Django

o   Description: Django uses custom HTTP header named "HTTP_IDEMPOTENCY_KEY"

o   Reference: https://pypi.org/project/django-idempotency-key

Organization: Twilio

o   Description: Twilio uses custom HTTP header named "I-Twilio-
    Idempotency-Token" in webhooks

o   Reference: https://www.twilio.com/docs/usage/webhooks/webhooks-
    connection-overrides

Organization: PayPal

o   Description: PayPal uses custom HTTP header named "PayPal-Request-
    Id"

o   Reference: https://developer.paypal.com/docs/business/develop/
    idempotency

Organization: RazorPay

o   Description: RazorPay uses custom HTTP header named "X-Payout-
    Idempotency"

o   Reference: https://razorpay.com/docs/razorpayx/api/idempotency/

Organization: OpenBanking

o   Description: OpenBanking uses custom HTTP header called "x-
    idempotency-key"

o   Reference: https://openbankinguk.github.io/read-write-api-
    site3/v3.1.6/profiles/read-write-data-api-profile.html#request-
    headers

Organization: Square

o   Description: To make an idempotent API call, Square recommends
    adding a property named "idempotency_key" with a unique value in
    the request body.

o   Reference: https://developer.squareup.com/docs/build-basics/using-
    rest-api

Organization: Google Standard Payments

o   Description: Google Standard Payments API uses a property named
    "requestId" in request body in order to provider idempotency in
    various use cases.

   o Reference: https://developers.google.com/standard-payments/
      payment-processor-service-api/rest/v1/TopLevel/capture

   Organization: BBVA

   o Description: BBVA Open Platform uses custom HTTP header called "X-
      Unique-Transaction-ID"

   o Reference:
      https://bbvaopenplatform.com/apiReference/APIbasics/content/x-
      unique-transaction-id

   Organization: WebEngage

   o Description: WebEngage uses custom HTTP header called "x-request-
      id" to identify webhook POST requests uniquely to achieve events
      idempotency.

   o Reference: https://docs.webengage.com/docs/webhooks

5. Security Considerations

   This section is meant to inform developers, information providers,
   and users of known security concerns specific to the idempotency
   keys.

   For idempotent request handling, the resources MAY make use of the
   value in the idempotency key to look up a cache or a persistent store
   for duplicate requests matching the key. If the resource does not
   validate the value of the idempotency key prior to performing such a
   lookup, it MAY lead to various forms of security attacks and
   compromise. To avoid such situations, the resource SHOULD publish
   the expected format of the idempotency key, algorithm used to
   generate it and always validate the key value as per the published
   specification before processing any request.

6. Examples

   The first example shows an idempotency-key header field with key
   value using UUID version 4 scheme:

   Idempotency-Key: "8e03978e-40d5-43e8-bc93-6894a57f9324"

   Second example shows an idempotency-key header field with key value
   using some random string generator:

   Idempotency-Key: "clkyoesmbgybucifusbbtdsbohtyuuwz"

7. References

7.1. Normative References

[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>.

[RFC3864]   Klyne, G., Nottingham, M., and J. Mogul, "Registration
            Procedures for Message Header Fields", BCP 90, RFC 3864,
            DOI 10.17487/RFC3864, September 2004,
            <https://www.rfc-editor.org/info/rfc3864>.

[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>.

[RFC4918]   Dusseault, L., Ed., "HTTP Extensions for Web Distributed
            Authoring and Versioning (WebDAV)", RFC 4918,
            DOI 10.17487/RFC4918, June 2007,
            <https://www.rfc-editor.org/info/rfc4918>.

[RFC5234]   Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
            Specifications: ABNF", STD 68, RFC 5234,
            DOI 10.17487/RFC5234, January 2008,
            <https://www.rfc-editor.org/info/rfc5234>.

[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>.

[RFC7231]   Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
            Protocol (HTTP/1.1): Semantics and Content", RFC 7231,
            DOI 10.17487/RFC7231, June 2014,
            <https://www.rfc-editor.org/info/rfc7231>.

[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/info/rfc8174>.

7.2. Informative References

   [RFC7942]   Sheffer, Y. and A. Farrel, "Improving Awareness of Running
               Code: The Implementation Status Section", BCP 205,
               RFC 7942, DOI 10.17487/RFC7942, July 2016,
               <https://www.rfc-editor.org/info/rfc7942>.

7.3. URIs

   [1] https://github.com/paypal/api-standards/blob/master/
        patterns.md#idempotency

   [2] https://stripe.com/docs/idempotency

   [3] https://tools.ietf.org/html/draft-nottingham-http-poe-00

Appendix A. Acknowledgments

   The authors would like to thank Mark Nottingham for his support for
   this Internet Draft. We would like to acknowledge that this draft is
   inspired by Idempotency related patterns described in API
   documentation of PayPal [1] and Stripe [2] as well as Internet Draft
   on POST Once Exactly [3] authored by Mark Nottingham.

   The authors take all responsibility for errors and omissions.

Appendix B. Appendix

B.1. Appendix A. Imported ABNF

   The following core rules are included by reference, as defined in
   Appendix B.1 of [RFC5234]: ALPHA (letters), CR (carriage return),
   CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double
   quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any
   8-bit sequence of data), SP (space), and VCHAR (any visible US-ASCII
   character).

   The rules below are defined in [RFC7230]:

     obs-text   = <obs-text, see [RFC7230], Section 3.2.6>

Authors' Addresses

   Jayadeba Jena
   PayPal, Inc.

   Email: jjena@paypal.com

Sanjay Dalal

Email: sanjay.dalal@cal.berkeley.edu
URI: https://github.com/sdatspun2