

HTTPAPI
Internet-Draft
Intended status: Standards Track
Expires: January 11, 2022

S. Dalal
E. Wilde
July 10, 2021

The Deprecation HTTP Header Field
draft-ietf-httpapi-deprecation-header-02

Abstract

The HTTP Deprecation Response Header Field can be used to signal to consumers of a URI-identified resource that the resource has been deprecated. Additionally, the deprecation link relation can be used to link to a resource that provides additional context for the deprecation, and possibly ways in which clients can find a replacement for the deprecated resource.

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 January 11, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. The Deprecation HTTP Response Header	3
2.1. Syntax	3
2.2. Scope	4
3. The Deprecation Link Relation Type	4
3.1. Documentation	5
4. Recommend Replacement	6
5. Sunset	6
6. Resource Behavior	7
7. IANA Considerations	7
7.1. The Deprecation HTTP Response Header	7
7.2. The Deprecation Link Relation Type	7
8. Implementation Status	8
8.1. Implementing the Deprecation Header	8
8.2. Implementing the Concept	10
9. Security Considerations	11
10. Examples	11
11. References	12
11.1. Normative References	12
11.2. Informative References	13
Appendix A. Acknowledgments	13
Authors' Addresses	13

1. Introduction

Deprecation of an HTTP resource as defined in Section 2 of [RFC7231] is a technique to communicate information about the lifecycle of a resource. It encourages applications to migrate away from the resource, discourages applications from forming new dependencies on the resource, and informs applications about the risk of continuing dependence upon the resource.

The act of deprecation does not change any behavior of the resource. It just informs client of the fact that a resource is deprecated. The Deprecation HTTP response header field MAY be used to convey this fact at runtime to clients. The header field can carry information indicating since when the deprecation is in effect.

In addition to the Deprecation header field the resource provider can use other header fields to convey additional information related to deprecation. For example, information such as where to find documentation related to the deprecation or what should be used as an

alternate and when the deprecated resource would be unreachable, etc. Alternates of a resource can be similar resource(s) or a newer version of the same resource.

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.

2. The Deprecation HTTP Response Header

The "Deprecation" HTTP response header field allows a server to communicate to a client that the resource in context of the message is or will be deprecated.

2.1. Syntax

The "Deprecation" response header field describes the deprecation. It either shows the deprecation date, which may be in the future (the resource context will be deprecated at that date) or in the past (the resource context has been deprecated at that date), or it simply flags the resource context as being deprecated:

Deprecation = IMF-fixdate / "true"

Servers MUST NOT include more than one "Deprecation" header field in the same response.

The date, if present, is the date when the resource context was or will be deprecated. It is in the form of an IMF-fixdate timestamp.

The following example shows that the resource context has been deprecated on Sunday, November 11, 2018 at 23:59:59 GMT:

Deprecation: Sun, 11 Nov 2018 23:59:59 GMT

The deprecation date can be in the future. If the value of "date" is in the future, it means that the resource will be deprecated at the given date in future.

If the deprecation date is not known, the header field can carry the simple string "true", indicating that the resource context is deprecated, without indicating when that happened:

Deprecation: true

2.2. Scope

The Deprecation header field applies to the resource that returns it, meaning that it announces the upcoming deprecation of that specific resource. However, there may be scenarios where the scope of the announced deprecation is larger than just the single resource where it appears.

Resources are free to define such an increased scope, and usually this scope will be documented by the resource so that consumers of the resource know about the increased scope and can behave accordingly. However, it is important to take into account that such increased scoping is invisible for consumers who are unaware of the increased scoping rules. This means that these consumers will not be aware of the increased scope, and they will not interpret Deprecation information different from its standard meaning (i.e., it applies to the resource only).

Using such an increased scope still may make sense, as Deprecation information is only a hint anyway; thus, it is optional information that cannot be depended on, and clients should always be implemented in ways that allow them to function without Deprecation information. Increased scope information may help clients to glean additional hints from resources and, thus, might allow them to implement behavior that allows them to make educated guesses about resources becoming deprecated.

3. The Deprecation Link Relation Type

In addition to the Deprecation HTTP header field, the server can use links with the "deprecation" link relation type to communicate to the client where to find more information about deprecation of the context. This can happen before the actual deprecation, to make a deprecation policy discoverable, or after deprecation, when there may be documentation about the deprecation, and possibly documentation of how to manage it.

This specification places no restrictions on the representation of the interlinked deprecation policy. In particular, the deprecation policy may be available as human-readable documentation or as machine-readable description.

3.1. Documentation

For a resource, deprecation could involve one or more parts of request, response or both. These parts could be one or more of the following.

- o URI - deprecation of one or more query parameter(s) or path element(s)
- o method - HTTP method for the resource is deprecated
- o request header - one or more HTTP request header(s) is deprecated
- o response header - HTTP response header(s) is deprecated
- o request body - request body contains one or more deprecated element(s)
- o response body - response body contains one or more deprecated element(s)

The purpose of the "Deprecation" header is to provide just enough "hints" about the deprecation to the client application developer. It is safe to assume that on reception of the "Deprecation" header, the client developer would look up the resource's documentation in order to find deprecation related semantics. The resource developer could provide a link to the resource documentation using a "Link" header with relation type "deprecation" as shown below.

```
Link: <https://developer.example.com/deprecation>;  
      rel="deprecation"; type="text/html"
```

In this example the interlinked content provides additional information about the deprecation of the resource context. There is no Deprecation header field in the response, and thus the resource is not deprecated. However, the resource already exposes a link where information is available how deprecation is managed for the context. This may be documentation explaining the use of the Deprecation header field, and also explaining under which circumstances and with which policies (announcement before deprecation; continued operation after deprecation) deprecation might be happening.

The following example uses the same link header, but also announces a deprecation date using a Deprecation header field.

```
Deprecation: Sun, 11 Nov 2018 23:59:59 GMT
Link: <https://developer.example.com/deprecation>;
      rel="deprecation"; type="text/html"
```

Given that the deprecation date is in the past, the linked resource may have been updated to include information about the deprecation, allowing clients to discover information about the deprecation that happened.

4. Recommend Replacement

The "Link" [RFC8288] header field can be used in addition to the "Deprecation" header field to inform the client about available alternatives to the deprecated resource. The following relation types as defined in [RFC8288] are RECOMMENDED to use for this purpose:

- o "successor-version": Points to a resource containing the successor version. [RFC5829]
- o "latest-version": Points to a resource containing the latest (e.g., current) version. [RFC5829]
- o "alternate": Designates a substitute. [W3C.REC-html401-19991224]

The following example provides link to the successor version of the requested resource that is deprecated.

```
Deprecation: Sun, 11 Nov 2018 23:59:59 GMT
Link: <https://api.example.com/v2/customers>; rel="successor-version"
```

This example provides link to an alternate resource to the requested resource that is deprecated.

```
Deprecation: Sun, 11 Nov 2018 23:59:59 GMT
Link: <https://api.example.com/v1/clients>; rel="alternate"
```

5. Sunset

In addition to the deprecation related information, if the resource provider wants to convey to the client application that the deprecated resource is expected to become unresponsive at a specific point in time, the Sunset HTTP header field [RFC8594] can be used in addition to the "Deprecation" header.

The timestamp given in the "Sunset" header field MUST be the later or the same as the one given in the "Deprecation" header field.

The following example shows that the resource in context has been deprecated since Sunday, November 11, 2018 at 23:59:59 GMT and its sunset date is Wednesday, November 11, 2020 at 23:59:59 GMT.

Deprecation: Sun, 11 Nov 2018 23:59:59 GMT
Sunset: Wed, 11 Nov 2020 23:59:59 GMT

6. Resource Behavior

The act of deprecation does not change any behavior of the resource. Deprecated resources SHOULD keep functioning as before, allowing consumers to still use the resources in the same way as they did before the resources were declared deprecated.

7. IANA Considerations

7.1. The Deprecation HTTP Response Header

The "Deprecation" response 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: Deprecation

Applicable Protocol: Hypertext Transfer Protocol (HTTP)

Status: Standard

Author: Sanjay Dalal <sanjay.dalal@cal.berkeley.edu>,
Erik Wilde <erik.wilde@dret.net>

Change controller: IETF

Specification document: this specification,
Section 2 "The Deprecation HTTP Response Header"

7.2. The Deprecation Link Relation Type

The "deprecation" link relation type should be added to the permanent registry of link relation types according to Section 4.2 of [RFC8288]:

Relation Type: deprecation

Applicable Protocol: Hypertext Transfer Protocol (HTTP)

Status: Standard

Author: Sanjay Dalal <sanjay.dalal@cal.berkeley.edu>,
Erik Wilde <erik.wilde@dret.net>

Change controller: IETF

Specification document: this specification,
Section 3 "The Deprecation Link Relation Type"

8. 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".

8.1. Implementing the Deprecation Header

This is a list of implementations that implement the deprecation header field:

Organization: Apollo

- o Description: Deprecation header is returned when deprecated functionality (as declared in the GraphQL schema) is accessed

- o Reference: <https://www.npmjs.com/package/apollo-server-tools>

Organization: Zalando

- o Description: Deprecation header is recommended as the preferred way to communicate API deprecation in Zalando API designs.
- o Reference: <https://opensource.zalando.com/restful-api-guidelines/#deprecation>

Organization: Palantir Technologies

- o Description: Deprecation header is incorporated in code generated by `conjure-java`, a CLI to generate Java POJOs and interfaces from Conjure API definitions
- o Reference: <https://github.com/palantir/conjure-java>

Organization: IETF Internet Draft, Registration Protocols Extensions

- o Description: Deprecation link relation is returned in Registration Data Access Protocol (RDAP) notices to indicate deprecation of `jCard` in favor of `JSContact`.
- o Reference: <https://tools.ietf.org/html/draft-loffredo-regext-rdap-jcard-deprecation>

Organization: E-Voyageurs Technologies

- o Description: Deprecation header is incorporated in `Hesperides`, a configuration management tool providing universal text file templating and properties editing through a REST API or a webapp.
- o Reference: https://github.com/voyages-sncf-technologies/hesperides/blob/master/documentation/lightweight-architecture-decision-records/deprecated_endpoints.md

Organization: Open-Xchange

- o Description: Deprecation header is used in Open-Xchange `appsuite-middleware`
- o Reference: <https://github.com/open-xchange/appsuite-middleware>

Organization: MediaWiki

- o Description: Core REST API of MediaWiki would use Deprecation header for endpoints that have been deprecated because a new endpoint provides the same or better functionality.
- o Reference: <https://phabricator.wikimedia.org/T232485>

8.2. Implementing the Concept

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

Organization: Zapier

- o Description: Zapier uses two custom HTTP headers named "X-API-Deprecation-Date" and "X-API-Deprecation-Info"
- o Reference: <https://zapier.com/engineering/api-geriatrics/>

Organization: IBM

- o Description: IBM uses a custom HTTP header named "Deprecated"
- o Reference:
https://www.ibm.com/support/knowledgecenter/en/SS42VS_7.3.1/com.ibm.qradar.doc/c_rest_api_getting_started.html

Organization: Ultipro

- o Description: Ultipro uses the HTTP "Warning" header as described in Section 5.5 of [RFC7234] with code "299"
- o Reference: <https://connect.ultipro.com/api-deprecation>

Organization: Clearbit

- o Description: Clearbit uses a custom HTTP header named "X-API-Warn"
- o Reference: <https://blog.clearbit.com/dealing-with-deprecation/>

Organization: PayPal

- o Description: PayPal uses a custom HTTP header named "PayPal-Deprecated"
- o Reference: <https://github.com/paypal/api-standards/blob/master/api-style-guide.md#runtime>

9. Security Considerations

The Deprecation header field SHOULD be treated as a hint, meaning that the resource is indicating (and not guaranteeing with certainty) that it is deprecated. Applications consuming the resource SHOULD check the resource documentation to verify authenticity and accuracy. Resource documentation SHOULD provide additional information about the deprecation including recommendation(s) for replacement.

In cases, where the Deprecation header field value is a date in future, it can lead to information that otherwise might not be available. Therefore, applications consuming the resource SHOULD verify the resource documentation and if possible, consult the resource developer to discuss potential impact due to deprecation and plan for possible transition to recommended resource.

In cases where "Link" header is used to provide more documentation and/or recommendation for replacement, one should assume that the content of the "Link" header field may not be secure, private or integrity-guaranteed, and due caution should be exercised when using it. Applications consuming the resource SHOULD check the referred resource documentation to verify authenticity and accuracy.

The suggested "Link" header fields make extensive use of IRIs and URIs. See [RFC3987] for security considerations relating to IRIs. See [RFC3986] for security considerations relating to URIs. See [RFC7230] for security considerations relating to HTTP headers.

Applications that take advantage of typed links should consider the attack vectors opened by automatically following, trusting, or otherwise using links gathered from the HTTP headers. In particular, Link headers that use the "successor-version", "latest-version" or "alternate" relation types should be treated with due caution. See [RFC5829] for security considerations relating to these link relation types.

10. Examples

The first example shows a deprecation header field without date information:

```
Deprecation: true
```

The second example shows a deprecation header with date information and a link to the successor version:

```
Deprecation: Sun, 11 Nov 2018 23:59:59 GMT
Link: <https://api.example.com/v2/customers>; rel="successor-version"
```

The third example shows a deprecation header field with links for the successor version and for the API's deprecation policy. In addition, it shows the sunset date for the deprecated resource:

```
Deprecation: Sun, 11 Nov 2018 23:59:59 GMT
Sunset: Wed, 11 Nov 2020 23:59:59 GMT
Link: <https://api.example.com/v2/customers>; rel="successor-version",
      <https://developer.example.com/deprecation>; rel="deprecation"
```

11. References

11.1. 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>>.
- [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>>.
- [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>>.
- [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>>.

- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.
- [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>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

11.2. Informative References

- [RFC5829] Brown, A., Clemm, G., and J. Reschke, Ed., "Link Relation Types for Simple Version Navigation between Web Resources", RFC 5829, DOI 10.17487/RFC5829, April 2010, <<https://www.rfc-editor.org/info/rfc5829>>.
- [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>>.
- [RFC8594] Wilde, E., "The Sunset HTTP Header Field", RFC 8594, DOI 10.17487/RFC8594, May 2019, <<https://www.rfc-editor.org/info/rfc8594>>.
- [W3C.REC-html401-19991224] Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, <<https://www.w3.org/TR/1999/REC-html401-19991224>>.

Appendix A. Acknowledgments

The authors would like to thank Nikhil Kolekar, Darrel Miller, Mark Nottingham, and Roberto Polli for their contributions.

The authors take all responsibility for errors and omissions.

Authors' Addresses

Sanjay Dalal

Email: sanjay.dalal@cal.berkeley.edu

URI: <https://github.com/sdatpun2>

Erik Wilde

Email: erik.wilde@dret.net

URI: <http://dret.net/netdret>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 9 November 2022

J. Jena
S. Dalal
8 May 2022

The Idempotency-Key HTTP Header Field
draft-ietf-httpapi-idempotency-key-header-01

Abstract

The HTTP Idempotency-Key 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 9 November 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. The Idempotency-Key HTTP Request Header Field	3
2.1. Syntax	3
2.2. Uniqueness of Idempotency Key	4
2.3. Idempotency Key Validity and Expiry	4
2.4. Idempotency Fingerprint	4
2.5. Responsibilities	4
2.6. Idempotency Enforcement Scenarios	5
2.7. Error Scenarios	5
3. IANA Considerations	6
3.1. The Idempotency-Key HTTP Request Header Field	6
4. Implementation Status	7
4.1. Implementing the Concept	8
5. Security Considerations	10
6. Examples	11
7. References	11
7.1. Normative References	11
7.2. Informative References	12
Appendix A. Imported ABNF	12
Acknowledgments	12
Authors' Addresses	12

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 APIs, duplicated resources are a severe problem from a 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

```
{::boilerplate bcp14-tagged}
```

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-Key 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

Idempotency-Key is an Item Structured Header [RFC8941]. Its value MUST be a String. Refer to Section 3.3.3 of [RFC8941] for ABNF of sf-string:

Idempotency-Key = sf-string

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 it in the 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.

- * Checksum of the entire request payload.
- * Checksum of selected element(s) in the request payload.
- * Field value match for each field in the request payload.
- * Field value match for selected element(s) in the request payload.
- * 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

- * 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

- * Identify idempotency key from the HTTP Idempotency-Key request header field.
- * Generate idempotency fingerprint if required.
- * Check for idempotency considering various scenarios including the ones described in section below.

2.6. Idempotency Enforcement Scenarios

- * First time request (idempotency key and fingerprint has not been seen)

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

- * Duplicate request (idempotency key and fingerprint has been seen)

Retry

The request was retried after the original request completed. The resource server SHOULD respond with the result of the previously completed operation, success or an error. See Error Scenarios for details on errors.

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 SHOULD 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 SHOULD 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 SHOULD 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"

Error scenarios above describe the status of failed idempotent requests after the resource server processes them. Clients MUST correct the requests (with the exception of 409 where no correction is required) before performing a retry operation, or the the resource server MUST fail the request and return one of the above errors.

For other 4xx/5xx errors, such as 401, 403, 500, 502, 503, 504, 429, or any other HTTP error code that is not listed here, the client SHOULD act appropriately by following the resource server's documentation.

3. IANA Considerations

3.1. The Idempotency-Key HTTP Request Header Field

The Idempotency-Key field name should be added to the "Hypertext Transfer Protocol (HTTP) Field Name Registry".

Field Name: Idempotency-Key

Status: permanent

Specification document: This specification, Section 2

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

- * Description: Stripe uses custom HTTP header named Idempotency-Key

- * Reference: <https://stripe.com/docs/idempotency>

Organization: Adyen

- * Description: Adyen uses custom HTTP header named Idempotency-Key

- * Reference: <https://docs.adyen.com/development-resources/api-idempotency/>

Organization: Dwolla

- * Description: Dwolla uses custom HTTP header named Idempotency-Key

- * Reference: <https://docs.dwolla.com/>

Organization: Interledger

- * Description: Interledger uses custom HTTP header named Idempotency-Key

- * Reference: <https://github.com/interledger/>

Organization: WorldPay

- * Description: WorldPay uses custom HTTP header named Idempotency-Key

- * Reference: <https://developer.worldpay.com/docs/wpg/idempotency>

Organization: Yandex

- * Description: Yandex uses custom HTTP header named Idempotency-Key

- * Reference: <https://cloud.yandex.com/docs/api-design-guide/concepts/idempotency>

Organization: http4s.org

- * Description: Http4s is a minimal, idiomatic Scala interface for HTTP services.

- * Reference: <https://github.com/http4s/http4s>

Organization: Finastra

- * Description: Finastra uses custom HTTP header named Idempotency-Key

- * Reference: <https://developer.fusionfabric.cloud/>

Organization: Datatrans

- * Description: Datatrans focuses on the technical processing of payments, including hosting smart payment forms and correctly routing payment information.

- * Reference: <https://docs.datatrans.ch/docs/api-endpoints>

4.1. Implementing the Concept

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

Organization: Django

- * Description: Django uses custom HTTP header named HTTP_IDEMPOTENCY_KEY

- * Reference: <https://pypi.org/project/django-idempotency-key>

Organization: Twilio

- * Description: Twilio uses custom HTTP header named I-Twilio-Idempotency-Token in webhooks
- * Reference: <https://www.twilio.com/docs/usage/webhooks/webhooks-connection-overrides>

Organization: PayPal

- * Description: PayPal uses custom HTTP header named PayPal-Request-Id
- * Reference: <https://developer.paypal.com/docs/business/develop/idempotency>

Organization: RazorPay

- * Description: RazorPay uses custom HTTP header named X-Payout-Idempotency
- * Reference: <https://razorpay.com/docs/razorpayx/api/idempotency/>

Organization: OpenBanking

- * Description: OpenBanking uses custom HTTP header called x-idempotency-key
- * Reference: <https://openbankinguk.github.io/read-write-api-site3/v3.1.6/profiles/read-write-data-api-profile.html#request-headers>

Organization: Square

- * Description: To make an idempotent API call, Square recommends adding a property named idempotency_key with a unique value in the request body.
- * Reference: <https://developer.squareup.com/docs/build-basics/using-rest-api>

Organization: Google Standard Payments

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

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

Organization: BBVA

- * Description: BBVA Open Platform uses custom HTTP header called X-Unique-Transaction-ID
- * Reference:
<https://bbvaopenplatform.com/apiReference/APIbasics/content/x-unique-transaction-id>

Organization: WebEngage

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

Resource servers that do not implement strong idempotency keys, such as UUIDs, or have appropriate controls to validate the idempotency keys, could be victim to various forms of security attacks from malicious clients:

- * Injection attacks-When the resource server does not validate the idempotency key in the client request and performs a idempotent cache lookup, there can be security attacks (primarily in the form of injection), compromising the server.
- * Data leaks-When an idempotency implementation allows low entropy keys, attackers MAY determine other keys and use them to fetch existing idempotent cache entries, belonging to other clients.

To prevent such situations, the specification recommends the following best practices for idempotency key implementation in the resource server.

- * Establish a fixed format for the idempotency key and publish the key's specification.

- * Always validate the key as per its published specification before processing any request.
- * On the resource server, implement a unique composite key as the idempotent cache lookup key. For example, a composite key MAY be implemented by combining the idempotency key sent by the client with other client specific attributes known only to the resource server.

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

- [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>>.
- [RFC8941] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/info/rfc8941>>.

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>>.

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>

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 (<https://github.com/paypal/api-standards/blob/master/patterns.md#idempotency>) and Stripe (<https://stripe.com/docs/idempotency>) as well as Internet Draft on POST Once Exactly (<https://tools.ietf.org/html/draft-nottingham-http-poe-00>) authored by Mark Nottingham.

The authors take all responsibility for errors and omissions.

Authors' Addresses

Jayadeba Jena
Email: jayadebaj@gmail.com

Sanjay Dalal

Email: sanjay.dalal@cal.berkeley.edu

URI: <https://github.com/sdatspun2>

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 5 January 2022

E. Wilde
Axway
H. Van de Sompel
Data Archiving and Networked Services
4 July 2021

Linkset: Media Types and a Link Relation Type for Link Sets
draft-ietf-httpapi-linkset-03

Abstract

This specification defines two document formats and respective media types for representing sets of links as stand-alone resources. One format is JSON-based, the other aligned with the format for representing links in the HTTP "Link" header field. This specification also introduces a link relation type to support discovery of sets of links.

Note to Readers

Please discuss this draft on the "Building Blocks for HTTP APIs" mailing list (<https://www.ietf.org/mailman/listinfo/httpapi>).

Online access to all versions and files is available on GitHub (<https://github.com/ietf-wg-httpapi/linkset>).

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 5 January 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Scenarios	3
3.1. Third-Party Links	4
3.2. Challenges Writing to HTTP Link Header Field	4
3.3. Large Number of Links	5
4. Document Formats for Sets of Links	5
4.1. HTTP Link Document Format: application/linkset	6
4.2. JSON Document Format: application/linkset+json	6
4.2.1. Set of Links	7
4.2.2. Link Context Object	7
4.2.3. Link Target Object	8
4.2.4. Link Target Attributes	10
4.2.5. JSON Extensibility	14
5. The "profile" attribute for media types to Represent Sets of Links	15
6. The "linkset" Relation Type for Linking to a Set of Links	15
7. Examples	16
7.1. Set of Links Provided as application/linkset	16
7.2. Set of Links Provided as application/linkset+json	17
7.3. Discovering a Link Set via the "linkset" Link Relation Type	19
8. IANA Considerations	20
8.1. Link Relation Type: linkset	20
8.2. Media Type: application/linkset	20
8.3. Media Type: application/linkset+json	21
9. Security Considerations	22
10. Normative References	23
11. Informative References	24
Appendix A. JSON-LD Context	25
Appendix B. Implementation Status	30
B.1. GS1	30
B.2. FAIR Signposting Profile	31
B.3. Open Journal Systems (OJS)	31
Acknowledgements	31
Authors' Addresses	31

1. Introduction

Resources on the Web often use typed Web Links [RFC8288], either embedded in resource representations, for example using the <link> element for HTML documents, or conveyed in the HTTP "Link" header field for documents of any media type. In some cases, however, providing links in this manner is impractical or impossible and delivering a set of links as a stand-alone document is preferable.

Therefore, this specification defines two document formats and associated media types to represent sets of links. It also defines the "linkset" relation type that supports discovery of any resource that conveys a set of links as a stand-alone document.

2. Terminology

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 terms "link context" and "link target" as defined in [RFC8288].

In the examples provided in this document, links in the HTTP "Link" header field are shown on separate lines in order to improve readability. Note, however, that as per Section 5.5 of [I-D.ietf-httpbis-semantics], line breaks are deprecated in values for HTTP fields; only whitespaces and tabs are supported as separators.

3. Scenarios

The following sections outline scenarios in which providing links by means of a standalone document instead of in an HTTP "Link" header field or as links embedded in the resource representation is advantageous or necessary.

For all scenarios, links could be provided by means of a stand-alone document that is formatted according to the JSON-based serialization, the serialization aligned with the HTTP "Link" field format, or both. The former serialization is motivated by the widespread use of JSON and related tools, which suggests that handling sets of links expressed as JSON documents should be attractive to developers. The latter serialization is provided for compatibility with the existing serialization used in the HTTP "Link" field and to allow reuse of tools created to handle it.

It is important to keep in mind that when providing links by means of a standalone representation, other links can still be provided using other approaches, i.e. it is possible combine various mechanisms to convey links.

3.1. Third-Party Links

In some cases it is useful that links pertaining to a resource are provided by a server other than the one that hosts the resource. For example, this allows:

- * Providing links in which the resource is involved not just as link context but also as link target.
- * Providing links pertaining to the resource that the server hosting that resource is not aware of.
- * External management of links pertaining to the resource in a special-purpose link management service.

In such cases, links pertaining to a resource can be provided by another, specific resource. That specific resource may be managed by the same or by another custodian as the resource to which the links pertain. For clients intent on consuming links provided in that manner, it would be beneficial if the following conditions were met:

- * Links are provided in a document that uses a well-defined media type.
- * The resource to which the provided links pertain is able to link to the resource that provides these links using a well-known link relation type.

These requirements are addressed in this specification through the definition of two media types and a link relation type, respectively.

3.2. Challenges Writing to HTTP Link Header Field

In some cases, it is not straightforward to write links to the HTTP "Link" header field from an application. This can, for example, be the case because not all required link information is available to the application or because the application does not have the capability to directly write HTTP fields. In such cases, providing links by means of a standalone document can be a solution. Making the resource that provides these links discoverable can be achieved by means of a typed link.

3.3. Large Number of Links

When conveying links in an HTTP "Link" header field, it is possible for the size of the HTTP response fields to become unpredictable. This can be the case when links are determined dynamically dependent on a range of contextual factors. It is possible to statically configure a web server to correctly handle large HTTP response fields by specifying an upper bound for their size. But when the number of links is unpredictable, estimating a reliable upper bound is challenging.

Section 15 of HTTP [I-D.ietf-httpbis-semantics] defines error codes related to excess communication by the user agent ("413 Request Entity Too Large" and "414 Request-URI Too Long"), but no specific error codes are defined to indicate that response field content exceeds the upper bound that can be handled by the server, and thus it has been truncated. As a result, applications take counter measures aimed at controlling the size of the HTTP "Link" header field, for example by limiting the links they provide to those with select relation types, thereby limiting the value of the HTTP "Link" header field to clients. Providing links by means of a standalone document overcomes challenges related to the unpredictable nature of the size of HTTP "Link" header fields.

4. Document Formats for Sets of Links

This section specifies two document formats to convey a set of links. Both are based on the abstract model specified in Section 2 of Web Linking [RFC8288] that defines a link as consisting of a "link context", a "link relation type", a "link target", and optional "target attributes":

- * The format defined in Section 4.1 is identical to the payload of the HTTP "Link" header field as specified in Web Linking Section 3 of [RFC8288].
- * The format defined in Section 4.2 is based on JSON [RFC8259].

Note that Section 3.3 of [RFC8288] deprecates the "rev" construct that was provided by [RFC5988] as a means to express links with a directionality that is the inverse of direct links that use the "rel" construct. In both serializations for link sets defined here, inverse links SHOULD be represented as direct links using the "rel" construct and by switching the position of the resources involved in the link.

4.1. HTTP Link Document Format: application/linkset

This document format is identical to the payload of the HTTP "Link" header field as defined in Section 3 of [RFC8288], more specifically by its ABNF production rule for "Link" and subsequent ones. The use of non-ASCII characters in the payload of the HTTP "Link" Header field is not interoperable.

The assigned media type for this format is "application/linkset".

In order to support use cases where "application/linkset" documents are re-used outside the context of an HTTP interaction, it is RECOMMENDED to make them self-contained by adhering to the following guidelines:

- * For every link provided in the set of links, explicitly provide the link context using the "anchor" attribute.
- * For link context ("anchor" attribute) and link target ("href" attribute), use URI References that are not relative references (as defined in Section 4.1 of [RFC3986]).

If these recommendations are not followed, interpretation of links in "application/linkset" documents will depend on which URI is used as context.

It should be noted that the "application/linkset" format specified here is different than the "application/link-format" format specified in [RFC6690] in that the former fully matches the payload of the HTTP "Link" header field as defined in Section 3 of [RFC8288], whereas the latter introduces constraints on that definition to meet requirements for Constrained RESTful Environments.

4.2. JSON Document Format: application/linkset+json

This document format uses JSON [RFC8259] as the syntax to represent a set of links. The set of links follows the abstract model defined by Web Linking Section 2 of [RFC8288].

The assigned media type for this format is "application/linkset+json".

In order to support use cases where "application/linkset+json" documents are re-used outside the context of an HTTP interaction, it is RECOMMENDED to make them self-contained by adhering to the following guidelines:

- * For every link provided in the set of links, explicitly provide the link context using the "anchor" member.
- * For link context ("anchor" member) and link target ("href" member), use URI References that are not relative references (as defined in Section 4.1 of [RFC3986]).

If these recommendations are not followed, interpretation of "application/linkset+json" will depend on which URI is used as context URI.

The "application/linkset+json" serialization is designed such that it can directly be used as the content of a JSON-LD serialization by adding an appropriate context. Appendix A shows an example of a possible context that, when added to a JSON serialization, allows it to be interpreted as RDF.

4.2.1. Set of Links

In the JSON representation of a set of links:

- * A set of links MUST be represented as a JSON object which MUST have "linkset" as its sole member.
- * The "linkset" member is an array in which a distinct JSON object - the "link context object" (see Section 4.2.2) - MUST be used to represent links that have the same link context.
- * Even if there is only one link context object, it MUST be wrapped in an array. Members other than link context objects MUST NOT be included in this array.

4.2.2. Link Context Object

In the JSON representation one or more links that have the same link context are represented by a JSON object, the link context object. A link context object adheres to the following rules:

- * Each link context object MAY have an "anchor" member with a value that represents the link context. If present, this value MUST be a URI Reference and SHOULD NOT be a relative reference as per Section 4.1 of [RFC3986].

- * For each distinct relation type that the link context has with link targets, a link context object MUST have an additional member. This member is an array in which a distinct JSON object – the "link target object" (see Section 4.2.3) – MUST be used for each link target for which the relationship with the link context (value of the encompassing anchor member) applies. The name of this member expresses the relation type of the link as follows:
 - For registered relation types (Section 2.1.1 of [RFC8288]), the name of this member is the registered name of the relation type.
 - For extension relation types (Section 2.1.2 of [RFC8288]), the name of this member is the URI that uniquely represents the relation type.
- * Even if there is only one link target object it MUST be wrapped in an array. Members other than link target objects MUST NOT be included in this array.

4.2.3. Link Target Object

In the JSON representation a link target is represented by a JSON object, the link target object. A link target object adheres to the following rules:

- * Each link target object MUST have an "href" member with a value that represents the link target. This value MUST be a URI Reference and SHOULD NOT be a relative reference as per Section 4.1 of [RFC3986]. Cases where the href member is present, but no value is provided for it (i.e. the resource providing the set of links is the target of the link in the link target object) MUST be handled by providing an "href" member with an empty string ("href": "").
- * In many cases, a link target is further qualified by target attributes. Various types of attributes exist and they are conveyed as additional members of the link target object as detailed in Section 4.2.4.

The following example of a JSON-serialized set of links represents one link with its core components: link context, link relation type, and link target.

```
{
  "linkset":
  [
    { "anchor": "http://example.net/bar",
      "next": [
        { "href": "http://example.com/foo" }
      ]
    }
  ]
}
```

Figure 1

The following example of a JSON-serialized set of links represents two links that share link context and relation type but have different link targets.

```
{
  "linkset":
  [
    { "anchor": "http://example.net/bar",
      "item": [
        { "href": "http://example.com/foo1" },
        { "href": "http://example.com/foo2" }
      ]
    }
  ]
}
```

Figure 2

The following example shows a set of links that represents two links, each with a different link context, link target, and relation type. One relation type is registered, the other is an extension relation type.

```
{
  "linkset":
  [
    { "anchor": "http://example.net/bar",
      "next": [
        { "href": "http://example.com/foo1" }
      ]
    },
    { "anchor": "http://example.net/boo",
      "http://example.com/relations/baz" : [
        { "href": "http://example.com/foo2" }
      ]
    }
  ]
}
```

Figure 3

4.2.4. Link Target Attributes

A link may be further qualified by target attributes. Three types of attributes exist:

- * Attributes defined in Section 3.4.1 of Web Linking [RFC8288].
- * Extension attributes defined and used by communities as allowed by Section 3.4.2 of [RFC8288].
- * Internationalized versions of the "title" attribute defined by [RFC8288] and of extension attributes allowed by Section 3.4 of [RFC8288].

The handling of these different types of attributes is described in the sections below.

4.2.4.1. Target Attributes Defined by Web Linking

Section 3.4.1 of [RFC8288] defines the following target attributes that may be used to annotate links: "hreflang", "media", "title", "title*", and "type"; these target attributes follow different occurrence and value patterns. In the JSON representation, these attributes MUST be conveyed as additional members of the link target object as follows:

- * **"hreflang"**: The optional and repeatable **"hreflang"** target attribute MUST be represented by an array (even if there only is one value to be represented), and each value in that array MUST be a string - representing one value of the **"hreflang"** target attribute for a link - which follows the same model as in the [RFC8288] syntax.
- * **"media"**: The optional and not repeatable **"media"** target attribute MUST be represented by a **"media"** member in the link target object, and its value MUST be a string that follows the same model as in the [RFC8288] syntax.
- * **"type"**: The optional and not repeatable **"type"** target attribute MUST be represented by a **"type"** member in the link target object, and its value MUST be a string that follows the same model as in the [RFC8288] syntax.
- * **"title"**: The optional and not repeatable **"title"** target attribute MUST be represented by a **"title"** member in the link target object, and its value MUST be a string that follows the same model as in the [RFC8288] syntax.
- * **"title*"**: The optional and not repeatable **"title*"** target attribute is motivated by character encoding and language issues and follows the model defined in [RFC8187]. The details of the JSON representation that applies to **title*** are described in Section 4.2.4.2.

The following example illustrates how the repeatable **"hreflang"** and the not repeatable **"type"** target attributes are represented in a link target object.

```
{
  "linkset":
  [
    { "anchor": "http://example.net/bar",
      "next": [
        { "href": "http://example.com/foo",
          "type": "text/html",
          "hreflang": [ "en" , "de" ]
        }
      ]
    }
  ]
}
```

Figure 4

4.2.4.2. Internationalized Target Attributes

In addition to the target attributes described in Section 4.2.4.1, Section 3.4 of [RFC8288] also supports attributes that follow the content model of [RFC8187]. In [RFC8288], these target attributes are recognizable by the use of a trailing asterisk in the attribute name, such as "title*". The content model of [RFC8187] uses a string-based microsyntax that represents the character encoding, an optional language tag, and the escaped attribute value encoded according to the specified character encoding.

The JSON serialization for these target attributes MUST be as follows:

- * An internationalized target attribute is represented as a member of the link context object with the same name (including the *) of the attribute.
- * The character encoding information as prescribed by [RFC8187] is not preserved; instead, the content of the internationalized attribute is represented in the character encoding used for the JSON set of links.
- * The value of the internationalized target attribute is an array that contains one or more JSON objects. The name of one member of such JSON object is "value" and its value is the actual content (in its unescaped version) of the internationalized target attribute, i.e. the value of the attribute from which the encoding and language information are removed. The name of another, optional, member of such JSON object is "language" and its value is the language tag [RFC5646] for the language in which the attribute content is conveyed.

The following example illustrates how the "title*" target attribute defined by Section 3.4.1 of [RFC8288] is represented in a link target object.

```
{
  "linkset":
  [
    { "anchor": "http://example.net/bar",
      "next": [
        { "href":      "http://example.com/foo",
          "type":      "text/html",
          "hreflang":  [ "en" , "de" ],
          "title":     "Next chapter",
          "title*":    [ { "value": "nächstes Kapitel" ,
                           "language" : "de" } ]
        }
      ]
    }
  ]
}
```

Figure 5

The above example assumes that the German title contains an umlaut character (in the native syntax it would be encoded as `title*=UTF-8'de'n%c3%a4chstes%20Kapitel`), which gets encoded in its unescaped form in the JSON representation. Implementations MUST properly decode/encode internationalized target attributes that follow the model of [RFC8187] when transcoding between the "application/linkset" and the "application/linkset+json" formats.

4.2.4.3. Extension Target Attributes

Extension target attributes are attributes that are not defined by Section 3.4.1 of [RFC8288] (as listed in Section 4.2.4.1), but are nevertheless used to qualify links. They can be defined by communities in any way deemed necessary, and it is up to them to make sure their usage is understood by target applications. However, lacking standardization, there is no interoperable understanding of these extension attributes. One important consequence is that their cardinality is unknown to generic applications. Therefore, in the JSON serialization, all extension target attributes are treated as repeatable.

The JSON serialization for these target attributes MUST be as follows:

- * An extension target attribute is represented as a member of the link context object with the same name of the attribute, including the * if applicable.

- * The value of an extension attribute MUST be represented by an array, even if there only is one value to be represented.
- * If the extension target attribute does not have a name with a trailing asterisk, then each value in that array MUST be a string that represents one value of the attribute.
- * If the extension attribute has a name with a trailing asterisk (it follows the content model of [RFC8187]), then each value in that array MUST be a JSON object. The value of each such JSON object MUST be structured as described in Section 4.2.4.2.

The example shows a link target object with three extension target attributes. The value for each extension target attribute is an array. The two first are regular extension target attributes, with the first one ("foo") having only one value and the second one ("bar") having two. The last extension target attribute ("baz*") follows the naming rule of [RFC8187] and therefore is encoded according to the serialization described in Section 4.2.4.2.

```
{
  "linkset":
  [
    { "anchor": "http://example.net/bar",
      "next": [
        { "href": "http://example.com/foo",
          "type": "text/html",
          "foo": [ "foovalue" ],
          "bar": [ "barone", "bartwo" ],
          "baz*": [ { "value": "bazvalue" ,
                      "language" : "en" } ]
        }
      ]
    }
  ]
}
```

Figure 6

4.2.5. JSON Extensibility

The extensibility of the JSON document format for representing a set of links is restricted to the extensibility provided by [RFC8288]. The Web linking model provides for the use of extension target attributes as discussed in Section 4.2.4.3. Extensions based on the JSON syntax MUST NOT be used, and MUST be ignored when found in a JSON linkset document.

This limitation of the JSON format allows to unambiguously round trip between links provided in the HTTP "Link" header field, sets of links serialized according to the "application/linkset" format, and sets of links serialized according to the "application/linkset+json" format.

5. The "profile" attribute for media types to Represent Sets of Links

As a means to convey specific constraints or conventions (as per [RFC6906]) that apply to a link set document, the "profile" attribute MAY be used in conjunction with the media types "application/linkset" and "application/linkset+json" detailed in Section 4.1 and Section 4.2, respectively. For example, the attribute could be used to indicate that a link set uses a specific, limited set of link relation types.

The value of the "profile" attribute MUST be a non-empty list of space-separated URIs, each of which identifies specific constraints or conventions that apply to the link set document. Profile URIs MAY be registered in the IANA Profile URI Registry in the manner specified by [RFC7284].

The presence of a "profile" attribute in conjunction with the "application/linkset" and "application/linkset+json" media types does not change the semantics of a link set. As such, clients with and without knowledge of profile URIs can use the same representation.

6. The "linkset" Relation Type for Linking to a Set of Links

The target of a link with the "linkset" relation type provides a set of links, including links in which the resource that is the link context participates.

A link with the "linkset" relation type MAY be provided in the header field and/or the body of a resource's representation. It may also be discovered by other means, such as through client-side information.

A resource MAY provide more than one link with a "linkset" relation type. Multiple such links can refer to the same set of links expressed using different media types, or to different sets of links, potentially provided by different third-party services.

A user agent that follows a "linkset" link MUST be aware that the set of links provided by the resource that is the target of the link can contain links in which the resource that is the context of the link does not participate; it MAY decide to ignore those links.

A user agent that follows a "linkset" link and obtains links for which anchors and targets are expressed as relative references (as per Section 4.1 of [RFC3986]) MUST determine what the context is for these links; it SHOULD ignore links for which it is unable to unambiguously make that determination.

7. Examples

Section 7.1 and Section 7.2 show examples whereby a set of links is provided as "application/linkset" and "application/linkset+json" documents, respectively. Section 7.3 illustrates the use of the "linkset" link relation type to support discovery of sets of links.

7.1. Set of Links Provided as application/linkset

Figure 7 shows a client issuing an HTTP GET request against resource <https://example.org/links/resource1>.

```
GET /links/resource1 HTTP/1.1
Host: example.org
```

Figure 7: Client HTTP GET request

Figure 8 shows the response to the GET request of Figure 7. The response contains a Content-Type header field specifying that the media type of the response is "application/linkset". A set of links, revealing authorship and versioning related to resource <https://example.org/resource1>, is provided in the response body. The HTTP "Link" header field indicates the availability of an alternate representation of the set of links using media type "application/linkset+json".

```
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:35:51 GMT
Server: Apache-Coyote/1.1
Content-Length: 1023
Content-Type: application/linkset
Link: <https://example.org/links/resource1>
      ; rel="alternate"
      ; type="application/linkset+json"

<https://authors.example.net/johndoe>
  ; rel="author"
  ; type="application/rdf+xml"
  ; anchor="https://example.org/resource1",
<https://example.org/resource1?version=3>
  ; rel="latest-version"
  ; type="text/html"
  ; anchor="https://example.org/resource1",
<https://example.org/resource1?version=2>
  ; rel="predecessor-version"
  ; type="text/html"
  ; anchor="https://example.org/resource1?version=3",
<https://example.org/resource1?version=1>
  ; rel="predecessor-version"
  ; type="text/html"
  ; anchor="https://example.org/resource1?version=2",
<https://example.org/resource1?version=1>
  ; rel="memento"
  ; type="text/html"
  ; datetime="Thu, 13 Jun 2019 09:34:33 GMT"
  ; anchor="https://example.org/resource1",
<https://example.org/resource1?version=2>
  ; rel="memento"
  ; type="text/html"
  ; datetime="Sun, 21 Jul 2019 12:22:04 GMT"
  ; anchor="https://example.org/resource1",
<https://authors.example.net/alice>
  ; rel="author"
  ; anchor="https://example.org/resource1#comment=1"
```

Figure 8: Response to HTTP GET includes a set of links

7.2. Set of Links Provided as application/linkset+json

Figure 9 shows the client issuing an HTTP GET request against `<https://example.org/links/resource1>`. In the request, the client uses an "Accept" header field to indicate it prefers a response in the "application/linkset+json" format.

```
GET links/resource1 HTTP/1.1
Host: example.org
Accept: application/linkset+json
```

Figure 9: Client HTTP GET request expressing preference for "application/linkset+json" response

Figure 10 shows the response to the HTTP GET request of Figure 9. The set of links is serialized according to the media type "application/linkset+json".

```
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:46:22 GMT
Server: Apache-Coyote/1.1
Content-Type: application/linkset+json
Link: <https://example.org/links/resource1>
      ; rel="alternate"
      ; type="application/linkset"
Content-Length: 1349

{
  "linkset": [
    {
      "anchor": "https://example.org/resource1",
      "author": [
        {
          "href": "https://authors.example.net/johndoe",
          "type": "application/rdf+xml"
        }
      ],
      "memento": [
        {
          "href": "https://example.org/resource1?version=1",
          "type": "text/html",
          "datetime": "Thu, 13 Jun 2019 09:34:33 GMT"
        },
        {
          "href": "https://example.org/resource1?version=2",
          "type": "text/html",
          "datetime": "Sun, 21 Jul 2019 12:22:04 GMT"
        }
      ],
      "latest-version": [
        {
          "href": "https://example.org/resource1?version=3",
          "type": "text/html"
        }
      ]
    }
  ]
}
```

```

    },
    {
      "anchor": "https://example.org/resource1?version=3",
      "predecessor-version": [
        {
          "href": "https://example.org/resource1?version=2",
          "type": "text/html"
        }
      ]
    },
    {
      "anchor": "https://example.org/resource1?version=2",
      "predecessor-version": [
        {
          "href": "https://example.org/resource1?version=1",
          "type": "text/html"
        }
      ]
    },
    {
      "anchor": "https://example.org/resource1#comment=1",
      "author": [
        {
          "href": "https://authors.example.net/alice"
        }
      ]
    }
  ]
}

```

Figure 10: Response to the client's request for the set of links

7.3. Discovering a Link Set via the "linkset" Link Relation Type

Figure 11 shows a client issuing an HTTP HEAD request against resource `<https://example.org/resource1>`.

```

HEAD resource1 HTTP/1.1
Host: example.org

```

Figure 11: Client HTTP HEAD request

Figure 12 shows the response to the HEAD request of Figure 11. The response contains an HTTP "Link" header field with a link that has the "linkset" relation type. It indicates that a set of links is provided by resource `<https://example.org/links/resource1>`, which provides a representation with media type "application/linkset+json".

```
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:45:54 GMT
Server: Apache-Coyote/1.1
Link: <https://example.org/links/resource1>
      ; rel="linkset"
      ; type="application/linkset+json"
Content-Length: 236
Content-Type: text/html;charset=utf-8
```

Figure 12: Response to HTTP HEAD request

Section 7.2 shows a client obtaining a set of links by issuing an HTTP GET on the target of the link with the "linkset" relation type, <https://example.org/links/resource1>.

8. IANA Considerations

8.1. Link Relation Type: linkset

The link relation type below should be registered by IANA per Section 6.2.1 of Web Linking [RFC8288]:

Relation Name: linkset

Description: The link target of a link with the "linkset" relation type provides a set of links, including links in which the link context of the link participates.

Reference: [[This document]]

8.2. Media Type: application/linkset

The Internet media type [RFC6838] for a natively encoded linkset is application/linkset.

Type name: application

Subtype name: linkset

Required parameters: none

Optional parameters: profile

Encoding considerations: Linksets are encoded according to the definition of [RFC8288]. The encoding of [RFC8288] is based on the general encoding rules of [I-D.ietf-httpbis-semantics], with the addition of allowing indicating character encoding and language for specific parameters as defined by [RFC8187].

Security considerations: The security considerations of [[This document]] apply.

Interoperability considerations: N/A

Published specification: [[This document]]

Applications that use this media type: This media type is not specific to any application, as it can be used by any application that wants to interchange web links.

Additional information:

Magic number(s): N/A

File extension(s): This media type does not propose a specific extension.

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Erik Wilde <erik.wilde@dret.net>

Intended usage: COMMON

Restrictions on usage: none

Author: Erik Wilde <erik.wilde@dret.net>

Change controller: IETF

8.3. Media Type: application/linkset+json

The Internet media type [RFC6838] for a JSON-encoded linkset is application/linkset+json.

Type name: application

Subtype name: linkset+json

Required parameters: none

Optional parameters: profile

Encoding considerations: The encoding considerations of [RFC8259] apply

Security considerations: The security considerations of [[This document]] apply.

Interoperability considerations: The interoperability considerations of [RFC8259] apply.

Published specification: [[This document]]

Applications that use this media type: This media type is not specific to any application, as it can be used by any application that wants to interchange web links.

Additional information:

Magic number(s): N/A

File extension(s): JSON documents often use ".json" as the file extension, and this media type does not propose a specific extension other than this generic one.

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Erik Wilde <erik.wilde@dret.net>

Intended usage: COMMON

Restrictions on usage: none

Author: Erik Wilde <erik.wilde@dret.net>

Change controller: IETF

9. Security Considerations

The security considerations of Web Linking [RFC8288] apply, as long as they are not specifically discussing the risks of exposing information in HTTP header fields.

In general, links may cause information leakage when they expose information (such as URIs) that can be sensitive or private. Links may expose "hidden URIs" that are not supposed to be openly shared, and may not be sufficiently protected. Ideally, none of the URIs

exposed in links should be supposed to be "hidden"; instead, if these URIs are supposed to be limited to certain users, then technical measures should be put in place so that accidentally exposing them does not cause any harm.

For the specific mechanisms defined in this specification, two security considerations should be taken into account:

- * The Web Linking model always has an "implicit context", which is the resource of the HTTP interaction. This original context can be lost or can change when self-contained link representations are moved. Changing the context can change the interpretation of links when they have no explicit anchor, or when they use relative URIs. Applications may choose to ignore links that have no explicit anchor or that use relative URIs when these are exchanged in stand-alone resources.
- * The model introduced in this specification supports "3rd party links", where one party can provide links that have another party's resource as an anchor. Depending on the link semantics and the application context, it is important to verify that there is sufficient trust in that 3rd party to allow it to provide these links. Applications may choose to treat 3rd party links differently than cases where a resource and the links for that resource are provided by the same party.

10. 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>>.
- [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>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

- [RFC8187] Reschke, J., "Indicating Character Encoding and Language for HTTP Header Field Parameters", RFC 8187, DOI 10.17487/RFC8187, September 2017, <<https://www.rfc-editor.org/info/rfc8187>>.
- [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>>.
- [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>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [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>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6906] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <<https://www.rfc-editor.org/info/rfc6906>>.
- [RFC7284] Lanthaler, M., "The Profile URI Registry", RFC 7284, DOI 10.17487/RFC7284, June 2014, <<https://www.rfc-editor.org/info/rfc7284>>.
- [I-D.ietf-httpbis-semantic]
Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantic-15, 30 March 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantic-15>>.

11. Informative References

- [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,
<<https://www.w3.org/TR/2014/REC-json-ld-20140116>>.
- [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>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988,
DOI 10.17487/RFC5988, October 2010,
<<https://www.rfc-editor.org/info/rfc5988>>.

Appendix A. JSON-LD Context

A set of links rendered according to the JSON serialization defined in Section 4.2 can be interpreted as RDF triples by adding a JSON-LD context [W3C.REC-json-ld-20140116] that maps the JSON keys to corresponding Linked Data terms. And, as per [W3C.REC-json-ld-20140116] section 6.8 (<https://www.w3.org/TR/2014/REC-json-ld-20140116/#interpreting-json-as-json-ld>), when delivering a link set that is rendered according to the "application/linkset+json" media type to a user agent, a server can convey the availability of such a JSON-LD context by using a link with the relation type "http://www.w3.org/ns/json-ld#context" in the HTTP "Link" header.

Using the latter approach to support discovery of a JSON-LD Context, the response to the GET request of Figure 9 against the URI of a set of links would be as shown in Figure 13.

```
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:48:22 GMT
Server: Apache-Coyote/1.1
Content-Type: application/linkset+json
Link: <https://example.org/contexts/linkset.jsonld>
      ; rel="http://www.w3.org/ns/json-ld#context"
      ; type="application/ld+json"
Content-Length: 1349
```

```
{
  "linkset": [
    {
      "anchor": "https://example.org/resource1",
      "author": [
        {
          "href": "https://authors.example.net/johndoe",
```

```
        "type": "application/rdf+xml"
      }
    ],
    "memento": [
      {
        "href": "https://example.org/resource1?version=1",
        "type": "text/html",
        "datetime": "Thu, 13 Jun 2019 09:34:33 GMT"
      },
      {
        "href": "https://example.org/resource1?version=2",
        "type": "text/html",
        "datetime": "Sun, 21 Jul 2019 12:22:04 GMT"
      }
    ],
    "latest-version": [
      {
        "href": "https://example.org/resource1?version=3",
        "type": "text/html"
      }
    ]
  },
  {
    "anchor": "https://example.org/resource1?version=3",
    "predecessor-version": [
      {
        "href": "https://example.org/resource1?version=2",
        "type": "text/html"
      }
    ]
  },
  {
    "anchor": "https://example.org/resource1?version=2",
    "predecessor-version": [
      {
        "href": "https://example.org/resource1?version=1",
        "type": "text/html"
      }
    ]
  },
  {
    "anchor": "https://example.org/resource1#comment=1",
    "author": [
      {
        "href": "https://authors.example.net/alice"
      }
    ]
  }
}
```

```
]
}
```

Figure 13: Using a typed link to support discovery of a JSON-LD Context for a Set of Links

In order to obtain the JSON-LD Context conveyed by the server, the user agent issues an HTTP GET against the link target of the link with the "http://www.w3.org/ns/json-ld#context" relation type. The response to this GET is shown in Figure 14. This particular JSON-LD context maps "application/linkset+json" representations of link sets to Dublin Core Terms. It also renders each link relation as a URI Reference, inspired by the same approach used for Atom [RFC4287] described in Appendix A.2 of [RFC8288].

```
HTTP/1.1 200 OK
Content-Type: application/ld+json
Content-Length: 708

{
  "@context": [
    {
      "@vocab": "http://www.iana.org/assignments/relation/",
      "anchor": "@id",
      "href": "@id",
      "linkset": "@graph",
      "_linkset": "@graph",
      "title": {
        "@id": "http://purl.org/dc/terms/title"
      },
      "title*": {
        "@id": "http://purl.org/dc/terms/title"
      },
      "type": {
        "@id": "http://purl.org/dc/terms/format"
      },
      "datetime": {
        "@id": "http://purl.org/dc/terms/date"
      }
    },
    {
      "language": "@language",
      "value": "@value",
      "hreflang": {
        "@id": "http://purl.org/dc/terms/language",
        "@container": "@set"
      }
    }
  ]
}
```

Figure 14: JSON-LD Context mapping to Dublin Core Terms and IANA assignments

Applying the JSON-LD context of Figure 14 to the link set of Figure 13 allows transforming the "application/linkset+json" link set to an RDF link set. Figure 15 shows the latter represented by means of the "text/turtle" RDF serialization.

```
<https://authors.example.net/johndoe>
  <http://purl.org/dc/terms/format>
    "application/rdf+xml" .
<https://example.org/resource1#comment=1>
  <http://www.iana.org/assignments/relation/author>
    <https://authors.example.net/alice> .
<https://example.org/resource1>
  <http://www.iana.org/assignments/relation/author>
    <https://authors.example.net/johndoe> .
<https://example.org/resource1>
  <http://www.iana.org/assignments/relation/latest-version>
    <https://example.org/resource1?version=3> .
<https://example.org/resource1>
  <http://www.iana.org/assignments/relation/memento>
    <https://example.org/resource1?version=1> .
<https://example.org/resource1>
  <http://www.iana.org/assignments/relation/memento>
    <https://example.org/resource1?version=2> .
<https://example.org/resource1?version=1>
  <http://purl.org/dc/terms/date>
    "Thu, 13 Jun 2019 09:34:33 GMT" .
<https://example.org/resource1?version=1>
  <http://purl.org/dc/terms/format>
    "text/html" .
<https://example.org/resource1?version=2>
  <http://purl.org/dc/terms/date>
    "Sun, 21 Jul 2019 12:22:04 GMT" .
<https://example.org/resource1?version=2>
  <http://purl.org/dc/terms/format>
    "text/html" .
<https://example.org/resource1?version=2>
  <http://www.iana.org/assignments/relation/predecessor-version>
    <https://example.org/resource1?version=1> .
<https://example.org/resource1?version=3>
  <http://purl.org/dc/terms/format>
    "text/html" .
<https://example.org/resource1?version=3>
  <http://www.iana.org/assignments/relation/predecessor-version>
    <https://example.org/resource1?version=2> .
```

Figure 15: RDF serialization of the link set resulting from applying the JSON-LD context

Note that the JSON-LD context of Figure 14 does not handle (meta)link relations of type `"linkset"` as they are in conflict with the top-level JSON key. A workaround is to rename the top-level key to `"_linkset"` in the `"application/linkset+json"` before transforming a link set to JSON-LD.

Appendix B. Implementation Status

This section is to be removed before publishing 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 RFC 6982 [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".

B.1. GS1

GS1 is a provider of barcodes (GS1 GTINs and EAN/UPC) for retail products and manages an ecology of services and standards to leverage them at a global scale. GS1 has indicated that it will implement this `"linkset"` specification as a means to allow requesting and representing links pertaining to products from various retailers. Currently, the GS1 Digital Link specification makes an informative reference to version 03 of the `"linkset"` I-D. GS1 expresses confidence that this will become a normative reference in the next iteration of that specification, likely to be ratified as a GS1 standard around February 2021.

B.2. FAIR Signposting Profile

The FAIR Signposting Profile is a community specification aimed at improving machine navigation of scholarly objects on the web through the use of typed web links pointing at e.g. web resources that are part of a specific object, persistent identifiers for the object and its authors, license information pertaining to the object. The specification encourages the use of Linksets and initial implementations are ongoing, for example, for the open source Dataverse data repository platform that was initiated by Harvard University and is meanwhile used by research institutions, worldwide.

B.3. Open Journal Systems (OJS)

Open Journal Systems (OJS) is an open-source software for the management of peer-reviewed academic journals, and is created by the Public Knowledge Project (PKP), released under the GNU General Public License. Open Journal Systems (OJS) is a journal management and publishing system that has been developed by PKP through its federally funded efforts to expand and improve access to research.

The OJS platform has implemented "linkset" support as an alternative way to provide links when there are more than a configured limit (they consider using about 10 as a good default, for testing purpose it is currently set to 8).

Acknowledgements

Thanks for comments and suggestions provided by Phil Archer, Dominique Guinard, Mark Nottingham, Julian Reschke, Stian Soiland-Reyes, and Sarven Capadisli.

Authors' Addresses

Erik Wilde
Axway

Email: erik.wilde@dret.net
URI: <http://dret.net/netdret/>

Herbert Van de Sompel
Data Archiving and Networked Services

Email: herbert.van.de.sompel@dans.knaw.nl
URI: <https://orcid.org/0000-0002-0715-6126>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 November 2022

E. Wilde
Axway
H. Van de Sompel
Data Archiving and Networked Services
5 May 2022

Linkset: Media Types and a Link Relation Type for Link Sets
draft-ietf-httpapi-linkset-10

Abstract

This specification defines two formats and respective media types for representing sets of links as stand-alone documents. One format is JSON-based, the other aligned with the format for representing links in the HTTP "Link" header field. This specification also introduces a link relation type to support discovery of sets of links.

Note to Readers

Please discuss this draft on the "Building Blocks for HTTP APIs" mailing list (<https://www.ietf.org/mailman/listinfo/httpapi>).

Online access to all versions and files is available on GitHub (<https://github.com/ietf-wg-httpapi/linkset>).

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 6 November 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Use Cases and Motivation	3
3.1. Third-Party Links	4
3.2. Challenges Writing to HTTP Link Header Field	5
3.3. Large Number of Links	5
4. Document Formats for Sets of Links	5
4.1. HTTP Link Document Format: application/linkset	7
4.2. JSON Document Format: application/linkset+json	7
4.2.1. Set of Links	8
4.2.2. Link Context Object	8
4.2.3. Link Target Object	9
4.2.4. Link Target Attributes	10
4.2.5. JSON Extensibility	15
5. The "profile" parameter for media types to Represent Sets of Links	15
6. The "linkset" Relation Type for Linking to a Set of Links . .	16
7. Examples	17
7.1. Set of Links Provided as application/linkset	17
7.2. Set of Links Provided as application/linkset+json	18
7.3. Discovering a Link Set via the "linkset" Link Relation Type	20
7.4. Link Set Profiles	21
7.4.1. Using a "profile" Attribute with a "linkset" Link . .	21
7.4.2. Using a "profile" Parameter with a Link Set Media Type	21
7.4.3. Using a Link with a "profile" Link Relation Type . .	22
8. IANA Considerations	23
8.1. Link Relation Type: linkset	23
8.2. Media Type: application/linkset	23
8.3. Media Type: application/linkset+json	24
9. Security Considerations	25
10. Normative References	26
11. Informative References	27
Appendix A. JSON-LD Context	28
Appendix B. Implementation Status	33
B.1. GS1	34

B.2. FAIR Signposting Profile	34
B.3. Open Journal Systems (OJS)	34
Acknowledgements	34
Authors' Addresses	35

1. Introduction

Resources on the Web often use typed Web Links [RFC8288], either embedded in resource representations, for example using the <link> element for HTML documents, or conveyed in the HTTP "Link" header field for documents of any media type. In some cases, however, providing links in this manner is impractical or impossible and delivering a set of links as a stand-alone document is preferable.

Therefore, this specification defines two formats for representing sets of Web Links and their attributes as stand-alone documents. One serializes links in the same format as used in the HTTP Link header field, and the other serializes links in JSON. It also defines associated media types to represent sets of links, and the "linkset" relation type that supports discovery of any resource that conveys a set of links as a stand-alone document.

2. Terminology

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 terms "link context" and "link target" in the same manner as [RFC8288].

In the examples provided in this document, links in the HTTP "Link" header field are shown on separate lines in order to improve readability. Note, however, that as per Section 5.5 of [I-D.ietf-httpbis-semantics], line breaks are deprecated in values for HTTP fields; only whitespaces and tabs are supported as separators.

3. Use Cases and Motivation

The following sections describe use cases in which providing links by means of a standalone document instead of in an HTTP "Link" header field or as links embedded in the resource representation is advantageous or necessary.

For all scenarios, links could be provided by means of a stand-alone document that is formatted according to the JSON-based serialization, the serialization aligned with the HTTP "Link" field format, or both. The former serialization is motivated by the widespread use of JSON and related tools, which suggests that handling sets of links expressed as JSON documents should be attractive to developers. The latter serialization is provided for compatibility with the existing serialization used in the HTTP "Link" field and to allow reuse of tools created to handle it.

It is important to keep in mind that when providing links by means of a standalone representation, other links can still be provided using other approaches, i.e. it is possible to combine various mechanisms to convey links.

3.1. Third-Party Links

In some cases it is useful that links pertaining to a resource are provided by a server other than the one that hosts the resource. For example, this allows:

- * Providing links in which the resource is involved not just as link context but also as link target, with a different resource being the link context.
- * Providing links pertaining to the resource that the server hosting that resource is not aware of.
- * External management of links pertaining to the resource in a special-purpose link management service.

In such cases, links pertaining to a resource can be provided by another, specific resource. That specific resource may be managed by the same or by another custodian as the resource to which the links pertain. For clients intent on consuming links provided in that manner, it would be beneficial if the following conditions were met:

- * Links are provided in a document that uses a well-defined media type.
- * The resource to which the provided links pertain is able to link to the resource that provides these links using a well-known link relation type.

These requirements are addressed in this specification through the definition of two media types and a link relation type, respectively.

3.2. Challenges Writing to HTTP Link Header Field

In some cases, it is not straightforward to write links to the HTTP "Link" header field from an application. This can, for example, be the case because not all required link information is available to the application or because the application does not have the capability to directly write HTTP fields. In such cases, providing links by means of a standalone document can be a solution. Making the resource that provides these links discoverable can be achieved by means of a typed link.

3.3. Large Number of Links

When conveying links in an HTTP "Link" header field, it is possible for the size of the HTTP response fields to become unpredictable. This can be the case when links are determined dynamically in a manner dependent on a range of contextual factors. It is possible to statically configure a web server to correctly handle large HTTP response fields by specifying an upper bound for their size. But when the number of links is unpredictable, estimating a reliable upper bound is challenging.

Section 15 of HTTP [I-D.ietf-httpbis-semantics] defines error codes related to excess communication by the user agent ("413 Request Entity Too Large" and "414 Request-URI Too Long"), but no specific error codes are defined to indicate that response field content exceeds the upper bound that can be handled by the server and thus has been truncated. As a result, applications take counter measures aimed at controlling the size of the HTTP "Link" header field, for example by limiting the links they provide to those with select relation types, thereby limiting the value of the HTTP "Link" header field to clients. Providing links by means of a standalone document overcomes challenges related to the unpredictable (to the web server implementation) nature of the size of HTTP "Link" header fields.

4. Document Formats for Sets of Links

This section specifies two document formats to convey a set of links. Both are based on the abstract model specified in Section 2 of Web Linking [RFC8288] that defines a link as consisting of a "link context", a "link relation type", a "link target", and optional "target attributes":

- * The format defined in Section 4.1 is nearly identical to the field value of the HTTP "Link" header field as specified in Section 3 of [RFC8288].
- * The format defined in Section 4.2 is expressed in JSON [RFC8259].

Links provided in the HTTP Link header are intended to be used in the context of an HTTP interaction and contextual information that is available during an interaction is used to correctly interpret them. Links provided in link sets, however, can be re-used outside of an HTTP interaction, when no such contextual information is available. As a result, implementers of link sets should strive to make them self-contained by adhering to the following recommendations.

For links provided in the HTTP Link header that have no anchor or that use relative references, the URI of the resource that delivers the links provides the contextual information that is needed for their correct interpretation. In order to support use cases where link set documents are re-used outside the context of an HTTP interaction, it is RECOMMENDED to make them self-contained by adhering to the following guidelines:

- * For every link provided in the set of links, explicitly provide the link context using the "anchor" attribute.
- * For link context ("anchor" attribute) and link target ("href" attribute), use URI references that are not relative references (as defined in Section 4.1 of [RFC3986]).

If these recommendations are not followed, interpretation of links in link set documents will depend on which URI is used as context.

For a "title" attribute provided on a link in the HTTP Link header, the language in which the title is expressed is provided by the Content-Language header of the HTTP interaction with the resource that delivers the links. This does not apply to "title" attributes provided for links in link set documents because that would constrain all links in a link set to having a single title language and would not support determining title languages when a link set is used outside of an HTTP interaction. In order to support use cases where link set documents are re-used outside the context of an HTTP interaction, it is RECOMMENDED to make them self-contained by using the "title*" attribute instead of the "title" attribute because "title*" allows expressing the title language as part of its value by means of a language tag. With this regard, note that language tags are matched case-insensitively (see Section 2.1.1 of [RFC5646]). If this recommendation is not followed, accurately determining the language of titles provided on links in link set documents will not be possible.

Note also that Section 3.3 of [RFC8288] deprecates the "rev" construct that was provided by [RFC5988] as a means to express links with a directionality that is the inverse of direct links that use the "rel" construct. In both serializations for link sets defined

here, inverse links may be represented as direct links using the "rel" construct and by switching the roles of the resources involved in the link.

4.1. HTTP Link Document Format: application/linkset

This document format is nearly identical to the field value of the HTTP "Link" header field as defined in Section 3 of [RFC8288], more specifically by its ABNF [RFC5234] production rule for "Link" and subsequent ones. It differs from the format for field values of the HTTP "Link" header only in that not only spaces and horizontal tabs are allowed as separators but also newline characters as a means to improve readability for humans. The use of non-ASCII characters in the field value of the HTTP "Link" Header field is not allowed, and as such is also not allowed in "application/linkset" link sets.

The assigned media type for this format is "application/linkset".

When converting an "application/linkset" document to a field value for the HTTP "Link" header, newline characters MUST be removed or MUST be replaced by white space (SP) in order to comply with Section 5.5 of [I-D.ietf-httpbis-semantics].

Implementers of "application/linkset" link sets should strive to make them self-contained by following the recommendations regarding their use outside the context of an HTTP interaction provided in Section 4.

It should be noted that the "application/linkset" format specified here is different from the "application/link-format" format specified in [RFC6690] in that the former fully matches the field value of the HTTP "Link" header field as defined in Section 3 of [RFC8288], whereas the latter introduces constraints on that definition to meet requirements for Constrained RESTful Environments (CoRE).

4.2. JSON Document Format: application/linkset+json

This document format uses JSON [RFC8259] as the syntax to represent a set of links. The set of links follows the abstract model defined by Web Linking Section 2 of [RFC8288].

The assigned media type for this format is "application/linkset+json".

In the interests of interoperability "application/linkset+json" link sets MUST be encoded using UTF-8 as per Section 8.1 of [RFC8259].

Implementers of "application/linkset+json" link sets should strive to make them self-contained by following the recommendations regarding their use outside the context of an HTTP interaction provided in Section 4.

The "application/linkset+json" serialization allows for OPTIONAL support of a JSON-LD serialization. This can be achieved by adding an appropriate context to the "application/linkset+json" serialization using the approach described in Section 6.8. of [W3C.REC-json-ld-20140116]. Communities of practice can decide which context best meets their application needs. Appendix A shows an example of a possible context that, when added to a JSON serialization, allows it to be interpreted as Resource Description Framework (RDF) [W3C.REC-rdf11-concepts-20140225] data.

4.2.1. Set of Links

In the JSON representation of a set of links:

- * A set of links is represented in JSON as an object which MUST contain "linkset" as its sole member.
- * The value of the "linkset" member is an array in which a distinct JSON object – the "link context object" (see Section 4.2.2) – is used to represent links that have the same link context.
- * Even if there is only one link context object, it MUST be wrapped in an array.

4.2.2. Link Context Object

In the JSON representation one or more links that have the same link context are represented by a JSON object, the link context object. A link context object adheres to the following rules:

- * Each link context object MAY contain an "anchor" member with a value that represents the link context. If present, this value MUST be a URI reference and SHOULD NOT be a relative reference as defined in Section 4.1 of [RFC3986].
- * For each distinct relation type that the link context has with link targets, a link context object MUST contain an additional member. The value of this member is an array in which a distinct JSON object – the "link target object" (see Section 4.2.3) – MUST be used for each link target for which the relationship with the link context (value of the encompassing anchor member) applies. The name of this member expresses the relation type of the link as follows:

- For registered relation types (Section 2.1.1 of [RFC8288]), the name of this member is the registered name of the relation type.
 - For extension relation types (Section 2.1.2 of [RFC8288]), the name of this member is the URI that uniquely represents the relation type.
- * Even if there is only one link target object it MUST be wrapped in an array.

4.2.3. Link Target Object

In the JSON representation a link target is represented by a JSON object, the link target object. A link target object adheres to the following rules:

- * Each link target object MUST contain an "href" member with a value that represents the link target. This value MUST be a URI reference and SHOULD NOT be a relative reference as defined in Section 4.1 of [RFC3986]. Cases where the href member is present, but no value is provided for it (i.e. the resource providing the set of links is the target of the link in the link target object) MUST be handled by providing an "href" member with an empty string as its value ("href": "").
- * In many cases, a link target is further qualified by target attributes. Various types of attributes exist and they are conveyed as additional members of the link target object as detailed in Section 4.2.4.

The following example of a JSON-serialized set of links represents one link with its core components: link context, link relation type, and link target.

```
{ "linkset":  
  [  
    { "anchor": "https://example.net/bar",  
      "next": [  
        { "href": "https://example.com/foo" }  
      ]  
    }  
  ]  
}
```

Figure 1

The following example of a JSON-serialized set of links represents two links that share link context and relation type but have different link targets.

```
{ "linkset":
  [
    { "anchor": "https://example.net/bar",
      "item": [
        { "href": "https://example.com/foo1" },
        { "href": "https://example.com/foo2" }
      ]
    }
  ]
}
```

Figure 2

The following example shows a set of links that represents two links, each with a different link context, link target, and relation type. One relation type is registered, the other is an extension relation type.

```
{ "linkset":
  [
    { "anchor": "https://example.net/bar",
      "next": [
        { "href": "https://example.com/foo1" }
      ]
    },
    { "anchor": "https://example.net/boo",
      "https://example.com/relations/baz" : [
        { "href": "https://example.com/foo2" }
      ]
    }
  ]
}
```

Figure 3

4.2.4. Link Target Attributes

A link may be further qualified by target attributes as defined by Section 2 of Web Linking [RFC8288]. Three types of attributes exist:

- * Serialisation-defined attributes described in Section 3.4.1 of Web Linking [RFC8288].

- * Extension attributes defined and used by communities as allowed by Section 3.4.2 of [RFC8288].
- * Internationalized versions of the "title" attribute defined by [RFC8288] and of extension attributes allowed by Section 3.4 of [RFC8288].

The handling of these different types of attributes is described in the sections below.

4.2.4.1. Target Attributes Defined by Web Linking

Section 3.4.1 of [RFC8288] defines the following target attributes that may be used to annotate links: "hreflang", "media", "title", "title*", and "type"; these target attributes follow different occurrence and value patterns. In the JSON representation, these attributes MUST be conveyed as additional members of the link target object as follows:

- * "hreflang": The "hreflang" target attribute, defined as optional and repeatable by [RFC8288], MUST be represented by an "hreflang" member, and its value MUST be an array (even if there only is one value to be represented), and each value in that array MUST be a string - representing one value of the "hreflang" target attribute for a link - which follows the same model as in the [RFC8288] syntax.
- * "media": The "media" target attribute, defined as optional and not repeatable by [RFC8288], MUST be represented by a "media" member in the link target object, and its value MUST be a string that follows the same model as in the [RFC8288] syntax.
- * "type": The "type" target attribute, defined as optional and not repeatable by [RFC8288], MUST be represented by a "type" member in the link target object, and its value MUST be a string that follows the same model as in the [RFC8288] syntax.
- * "title": The "title" target attribute, defined as optional and not repeatable by [RFC8288], MUST be represented by a "title" member in the link target object, and its value MUST be a JSON string.
- * "title*": The "title*" target attribute, defined as optional and not repeatable by [RFC8288], is motivated by character encoding and language issues and follows the model defined in [RFC8187]. The details of the JSON representation that applies to title* are described in Section 4.2.4.2.

The following example illustrates how the repeatable "hreflang" and the not repeatable "type" target attributes are represented in a link target object.

```
{ "linkset":  
  [  
    { "anchor": "https://example.net/bar",  
      "next": [  
        { "href": "https://example.com/foo",  
          "type": "text/html",  
          "hreflang": [ "en" , "de" ]  
        }  
      ]  
    }  
  ]  
}
```

Figure 4

4.2.4.2. Internationalized Target Attributes

In addition to the target attributes described in Section 4.2.4.1, Section 3.4 of [RFC8288] also supports attributes that follow the content model of [RFC8187]. In [RFC8288], these target attributes are recognizable by the use of a trailing asterisk in the attribute name, such as "title*". The content model of [RFC8187] uses a string-based microsyntax that represents the character encoding, an optional language tag, and the escaped attribute value encoded according to the specified character encoding.

The JSON serialization for these target attributes MUST be as follows:

- * An internationalized target attribute is represented as a member of the link context object with the same name (including the *) as the attribute.
- * The character encoding information as prescribed by [RFC8187] is not preserved; instead, the content of the internationalized attribute is represented as a JSON string.

- * The value of the internationalized target attribute is an array that contains one or more JSON objects. The name of one member of such JSON object is "value" and its value is the actual content (in its unescaped version) of the internationalized target attribute, i.e. the value of the attribute from which the encoding and language information are removed. The name of another, optional, member of such JSON object is "language" and its value is the language tag [RFC5646] for the language in which the attribute content is conveyed.

The following example illustrates how the "title*" target attribute defined by Section 3.4.1 of [RFC8288] is represented in a link target object.

```
{ "linkset":  
  [  
    { "anchor": "https://example.net/bar",  
      "next": [  
        { "href":      "https://example.com/foo",  
          "type":      "text/html",  
          "hreflang":  [ "en" , "de" ],  
          "title":     "Next chapter",  
          "title*":    [ { "value": "nächstes Kapitel" ,  
                           "language" : "de" } ]  
        }  
      ]  
    }  
  ]  
}
```

Figure 5

The above example assumes that the German title contains an umlaut character (in the original syntax it would be encoded as title*=UTF-8'de'n%c3%a4chstes%20Kapitel), which gets encoded in its unescaped form in the JSON representation. Implementations MUST properly decode/encode internationalized target attributes that follow the model of [RFC8187] when transcoding between the "application/linkset" and the "application/linkset+json" formats.

4.2.4.3. Extension Target Attributes

Extension target attributes are attributes that are not defined by Section 3.4.1 of [RFC8288] (as listed in Section 4.2.4.1), but are nevertheless used to qualify links. They can be defined by communities in any way deemed necessary, and it is up to them to make sure their usage is understood by target applications. However, lacking standardization, there is no interoperable understanding of

these extension attributes. One important consequence is that their cardinality is unknown to generic applications. Therefore, in the JSON serialization, all extension target attributes are treated as repeatable.

The JSON serialization for these target attributes MUST be as follows:

- * An extension target attribute is represented as a member of the link target object with the same name as the attribute, including the * if applicable.
- * The value of an extension attribute MUST be represented by an array, even if there only is one value to be represented.
- * If the extension target attribute does not have a name with a trailing asterisk, then each value in that array MUST be a JSON string that represents one value of the attribute.
- * If the extension attribute has a name with a trailing asterisk (it follows the content model of [RFC8187]), then each value in that array MUST be a JSON object. The value of each such JSON object MUST be structured as described in Section 4.2.4.2.

The example shows a link target object with three extension target attributes. The value for each extension target attribute is an array. The two first are regular extension target attributes, with the first one ("foo") having only one value and the second one ("bar") having two. The last extension target attribute ("baz*") follows the naming rule of [RFC8187] and therefore is encoded according to the serialization described in Section 4.2.4.2.

```
{ "linkset":  
  [  
    { "anchor": "https://example.net/bar",  
      "next": [  
        { "href": "https://example.com/foo",  
          "type": "text/html",  
          "foo": [ "foovalue" ],  
          "bar": [ "barone", "bartwo" ],  
          "baz*": [ { "value": "bazvalue" ,  
                     "language" : "en" } ]  
        }  
      ]  
    }  
  ]  
}
```


Figure 6

4.2.5. JSON Extensibility

The Web linking model ([RFC8288]) provides for the use of extension target attributes as discussed in Section 4.2.4.3. The use of other forms of extensions is NOT RECOMMENDED. Limiting the JSON format in this way allows to unambiguously round trip between links provided in the HTTP "Link" header field, sets of links serialized according to the "application/linkset" format, and sets of links serialized according to the "application/linkset+json" format.

Cases may exist in which the use of extensions other than those of Section 4.2.4.3 may be useful. For example, when a link set publisher needs to include descriptive or technical metadata for internal consumption. In case such extensions are used they MUST NOT change the semantics of the JSON members defined in this specification. Agents that consume JSON linkset documents can safely ignore such extensions.

5. The "profile" parameter for media types to Represent Sets of Links

As a means to convey specific constraints or conventions (as per [RFC6906]) that apply to a link set document, the "profile" parameter MAY be used in conjunction with the media types "application/linkset" and "application/linkset+json" detailed in Section 4.1 and Section 4.2, respectively. For example, the parameter could be used to indicate that a link set uses a specific, limited set of link relation types.

The value of the "profile" parameter MUST be a non-empty list of space-separated URIs, each of which identifies specific constraints or conventions that apply to the link set document. When providing multiple profile URIs, care should be taken that the corresponding profiles are not conflicting. Profile URIs MAY be registered in the IANA Profile URI Registry in the manner specified by [RFC7284].

The presence of a "profile" parameter in conjunction with the "application/linkset" and "application/linkset+json" media types does not change the semantics of a link set. As such, clients with and without knowledge of profile URIs can use the same representation.

Section 7.4.2 shows an example of using the "profile" parameter in conjunction with the "application/linkset+json" media type.

6. The "linkset" Relation Type for Linking to a Set of Links

The target of a link with the "linkset" relation type provides a set of links, including links in which the resource that is the link context participates.

A link with the "linkset" relation type MAY be provided in the header field and/or the body of a resource's representation. It may also be discovered by other means, such as through client-side information.

A resource MAY provide more than one link with a "linkset" relation type. Multiple such links can refer to the same set of links expressed using different media types, or to different sets of links, potentially provided by different third-party services.

The set of links provided by the resource that is the target of a "linkset" link may contain links in which the resource that is the context of the "linkset" link does not participate. User agents MUST process each link in the link set independently, including processing of link context and link target, and MAY ignore links from the link set in which the context of the "linkset" link does not participate.

A user agent that follows a "linkset" link and obtains links for which anchors and targets are expressed as relative references (as per Section 4.1 of [RFC3986]) MUST determine what the context is for these links; it SHOULD ignore links for which it is unable to unambiguously make that determination.

As a means to convey specific constraints or conventions (as per [RFC6906]) that apply to a link set document, the "profile" attribute MAY be used in conjunction with the "linkset" link relation type. For example, the attribute could be used to indicate that a link set uses a specific, limited set of link relation types. The value of the "profile" attribute MUST be a non-empty list of space-separated URIs, each of which identifies specific constraints or conventions that apply to the link set document. Profile URIs MAY be registered in the IANA Profile URI Registry in the manner specified by [RFC7284]. Section 7.4.1 shows an example of using the "profile" attribute on a link with the "linkset" relation type, making both the link set and the profile(s) to which it complies discoverable.

7. Examples

Section 7.1 and Section 7.2 show examples whereby a set of links is provided as "application/linkset" and "application/linkset+json" documents, respectively. Section 7.3 illustrates the use of the "linkset" link relation type to support discovery of sets of links and Section 7.4 shows how to convey profile information pertaining to a link set.

7.1. Set of Links Provided as application/linkset

Figure 7 shows a client issuing an HTTP GET request against resource <https://example.org/links/resource1>.

```
GET /links/resource1 HTTP/1.1
Host: example.org
```

Figure 7: Client HTTP GET request

Figure 8 shows the response to the GET request of Figure 7. The response contains a Content-Type header field specifying that the media type of the response is "application/linkset". A set of links, revealing authorship and versioning related to resource <https://example.org/resource1>, is provided in the response body. The HTTP "Link" header field indicates the availability of an alternate representation of the set of links using media type "application/linkset+json".

```
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:35:51 GMT
Server: Apache-Coyote/1.1
Content-Length: 1023
Content-Type: application/linkset
Link: <https://example.org/links/resource1>
      ; rel="alternate"
      ; type="application/linkset+json"

<https://authors.example.net/johndoe>
  ; rel="author"
  ; type="application/rdf+xml"
  ; anchor="https://example.org/resource1",
<https://example.org/resource1?version=3>
  ; rel="latest-version"
  ; type="text/html"
  ; anchor="https://example.org/resource1",
<https://example.org/resource1?version=2>
  ; rel="predecessor-version"
  ; type="text/html"
  ; anchor="https://example.org/resource1?version=3",
<https://example.org/resource1?version=1>
  ; rel="predecessor-version"
  ; type="text/html"
  ; anchor="https://example.org/resource1?version=2",
<https://example.org/resource1?version=1>
  ; rel="memento"
  ; type="text/html"
  ; datetime="Thu, 13 Jun 2019 09:34:33 GMT"
  ; anchor="https://example.org/resource1",
<https://example.org/resource1?version=2>
  ; rel="memento"
  ; type="text/html"
  ; datetime="Sun, 21 Jul 2019 12:22:04 GMT"
  ; anchor="https://example.org/resource1",
<https://authors.example.net/alice>
  ; rel="author"
  ; anchor="https://example.org/resource1#comment=1"
```

Figure 8: Response to HTTP GET includes a set of links

7.2. Set of Links Provided as application/linkset+json

Figure 9 shows the client issuing an HTTP GET request against `<https://example.org/links/resource1>`. In the request, the client uses an "Accept" header field to indicate it prefers a response in the "application/linkset+json" format.

```
GET links/resource1 HTTP/1.1
Host: example.org
Accept: application/linkset+json
```

Figure 9: Client HTTP GET request expressing preference for "application/linkset+json" response

Figure 10 shows the response to the HTTP GET request of Figure 9. The set of links is serialized according to the media type "application/linkset+json".

```
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:46:22 GMT
Server: Apache-Coyote/1.1
Content-Type: application/linkset+json
Link: <https://example.org/links/resource1>
      ; rel="alternate"
      ; type="application/linkset"
Content-Length: 1246

{ "linkset":
  [
    { "anchor": "https://example.org/resource1",
      "author": [
        { "href": "https://authors.example.net/johndoe",
          "type": "application/rdf+xml"
        }
      ],
      "memento": [
        { "href": "https://example.org/resource1?version=1",
          "type": "text/html",
          "datetime": "Thu, 13 Jun 2019 09:34:33 GMT"
        },
        { "href": "https://example.org/resource1?version=2",
          "type": "text/html",
          "datetime": "Sun, 21 Jul 2019 12:22:04 GMT"
        }
      ],
      "latest-version": [
        { "href": "https://example.org/resource1?version=3",
          "type": "text/html"
        }
      ]
    },
    { "anchor": "https://example.org/resource1?version=3",
      "predecessor-version": [
        { "href": "https://example.org/resource1?version=2",
          "type": "text/html"
        }
      ]
    }
  ]
}
```

```

    }
  ]
},
{ "anchor": "https://example.org/resource1?version=2",
  "predecessor-version": [
    { "href": "https://example.org/resource1?version=1",
      "type": "text/html"
    }
  ]
},
{ "anchor": "https://example.org/resource1#comment=1",
  "author": [
    { "href": "https://authors.example.net/alice" }
  ]
}
]
}

```

Figure 10: Response to the client's request for the set of links

7.3. Discovering a Link Set via the "linkset" Link Relation Type

Figure 11 shows a client issuing an HTTP HEAD request against resource `<https://example.org/resource1>`.

```

HEAD resource1 HTTP/1.1
Host: example.org

```

Figure 11: Client HTTP HEAD request

Figure 12 shows the response to the HEAD request of Figure 11. The response contains an HTTP "Link" header field with a link that has the "linkset" relation type. It indicates that a set of links is provided by resource `<https://example.org/links/resource1>`, which provides a representation with media type "application/linkset+json".

```

HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:45:54 GMT
Server: Apache-Coyote/1.1
Link: <https://example.org/links/resource1>
      ; rel="linkset"
      ; type="application/linkset+json"
Content-Length: 236
Content-Type: text/html;charset=utf-8

```

Figure 12: Response to HTTP HEAD request

7.4. Link Set Profiles

The examples in this section illustrate the use of the "profile" attribute for a link with the "linkset" link relation type and the "profile" attribute for a link set media type. The examples are inspired by the implementation of link sets by GS1 (the standards body behind many of the world's barcodes).

7.4.1. Using a "profile" Attribute with a "linkset" Link

Figure 13 shows a client issuing an HTTP HEAD request against trade item 09506000134352 at <<https://id.gs1.org/01/9506000134352>>.

```
HEAD /01/9506000134352 HTTP/1.1
Host: id.gs1.org
```

Figure 13: Client HTTP HEAD request

Figure 14 shows the server's response to the request of Figure 13, including a "linkset" link with a "profile" attribute that has the Profile URI <<https://www.gs1.org/voc/?show=linktypes>> as its value. Dereferencing that URI yields a profile document that lists all the link relation types that a client can expect when requesting the link set made discoverable by the "linkset" link. The link relation types are presented in abbreviated form, e.g. <gs1:activityIdeas>, whereas the actual link relation type URIs are available as hyperlinks on the abbreviations, e.g. <<https://www.gs1.org/voc/activityIdeas>>. For posterity that profile document was saved in the Internet Archive at <<https://web.archive.org/web/20210927160406/https://www.gs1.org/voc/?show=linktypes>> on 27 September 2021.

```
HTTP/1.1 307 Temporary Redirect
Date: Mon, 27 Sep 2021 16:03:07 GMT
Server: nginx
Link: https://id.gs1.org/01/9506000134352?linkType=all
; rel="linkset"
; type="application/linkset+json"
; profile="https://www.gs1.org/voc/?show=linktypes"
Location: https://example.com/risotto-rice-with-mushrooms/
```

Figure 14: Response to the client's HEAD request including a "profile" attribute for the "linkset" link

7.4.2. Using a "profile" Parameter with a Link Set Media Type

Figure 15 shows a client issuing an HTTP HEAD request against the link set <<https://id.gs1.org/01/9506000134352?linkType=all>> that was discovered through the HTTP interactions shown in Section 7.4.1.

```
HEAD /01/9506000134352?linkType=all HTTP/1.1
Host: id.gsl.org
```

Figure 15: Client HTTP HEAD request

Figure 16 shows the server's response to the request of Figure 15. Note the "profile" parameter for the application/linkset+json media type, which has as value the same Profile URI <<https://www.gsl.org/voc/?show=linktypes>> as was used in <xref target="Response_pr_at"/>.

```
HTTP/1.1 200 OK
Date: Mon, 27 Sep 2021 16:03:33 GMT
Server: nginx
Content-Type: application/linkset+json;
    profile="https://www.gsl.org/voc/?show=linktypes"
Content-Length: 396
```

Figure 16: Response to the client's HEAD request including a "profile" parameter for the "application/linkset+json" media type

7.4.3. Using a Link with a "profile" Link Relation Type

Note that the response Figure 16 from the link set resource is equivalent to the response shown in Figure 17, which leverages the "profile" link relation type defined in [RFC6906].

```
HTTP/1.1 200 OK
Date: Mon, 27 Sep 2021 16:03:33 GMT
Server: nginx
Content-Type: application/linkset+json
Link: https://www.gsl.org/voc/?show=linktypes; rel="profile"
Content-Length: 396
```

Figure 17: Response to the client's HEAD request including a "profile" link

A link with a "profile" link relation type as shown in Figure 17 can also be conveyed in the link set document itself. This is illustrated by Figure 18. Following the recommendation that all links in a link set document should have an explicit anchor, such a link has the URI of the link set itself as anchor and the Profile URI as target. Multiple Profile URIs are handled by using multiple "href" members.


```
{ "linkset":  
  [  
    { "anchor": "https://id.gsl.org/01/9506000134352?linkType=all",  
      "profile": [  
        { "href": "https://www.gsl.org/voc/?show=linktypes"}  
      ]  
    },  
    { "anchor": "https://id.gsl.org/01/9506000134352",  
      "https://gsl.org/voc/whatsInTheBox": [  
        { "href": "https://example.com/en/packContents/GB"}  
      ]  
    }  
  ]  
}
```

Figure 18: A Link Set that declares the profile it complies with using a "profile" link

8. IANA Considerations

8.1. Link Relation Type: linkset

The link relation type below should be registered by IANA in the Link Relation Type Registry as per Section 4.2 of Web Linking [RFC8288]:

Relation Name: linkset

Description: The link target of a link with the "linkset" relation type provides a set of links, including links in which the link context of the link participates.

Reference: [[This document]]

8.2. Media Type: application/linkset

The Internet media type application/linkset for a linkset encoded as described in Section 4.1 should be registered by IANA in the Media Type Registry as per [RFC6838].

Type name: application

Subtype name: linkset

Required parameters: N/A

Optional parameters: profile

Encoding considerations: Linksets are encoded according to the definition of [RFC8288]. The encoding of [RFC8288] is based on the general encoding rules of [I-D.ietf-httpbis-semantics], with the addition of allowing indicating character encoding and language for specific parameters as defined by [RFC8187].

Security considerations: The security considerations of [[This document]] apply.

Interoperability considerations: N/A

Published specification: [[This document]]

Applications that use this media type: This media type is not specific to any application, as it can be used by any application that wants to interchange web links.

Additional information:

Magic number(s): N/A

File extension(s): This media type does not propose a specific extension.

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Erik Wilde <erik.wilde@dret.net>

Intended usage: COMMON

Restrictions on usage: none

Author: Erik Wilde <erik.wilde@dret.net>

Change controller: IETF

8.3. Media Type: application/linkset+json

The Internet media type application/linkset+json for a linkset encoded as described in Section 4.2 should be registered by IANA in the Media Type Registry as per [RFC6838].

Type name: application

Subtype name: linkset+json

Required parameters: N/A

Optional parameters: profile

Encoding considerations: The encoding considerations of [RFC8259] apply

Security considerations: The security considerations of [[This document]] apply.

Interoperability considerations: The interoperability considerations of [RFC8259] apply.

Published specification: [[This document]]

Applications that use this media type: This media type is not specific to any application, as it can be used by any application that wants to interchange web links.

Additional information:

Magic number(s): N/A

File extension(s): JSON documents often use ".json" as the file extension, and this media type does not propose a specific extension other than this generic one.

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Erik Wilde <erik.wilde@dret.net>

Intended usage: COMMON

Restrictions on usage: none

Author: Erik Wilde <erik.wilde@dret.net>

Change controller: IETF

9. Security Considerations

The security considerations of Section 7 of [RFC3986] apply, as well as those of Web Linking [RFC8288] as long as the latter are not specifically discussing the risks of exposing information in HTTP header fields.

In general, links may cause information leakage when they expose information (such as URIs) that can be sensitive or private. Links may expose "hidden URIs" that are not supposed to be openly shared,

and may not be sufficiently protected. Ideally, none of the URIs exposed in links should be supposed to be "hidden"; instead, if these URIs are supposed to be limited to certain users, then technical measures should be put in place so that accidentally exposing them does not cause any harm.

For the specific mechanisms defined in this specification, two security considerations should be taken into account:

- * The Web Linking model always has an "implicit context", which is the resource of the HTTP interaction. This original context can be lost or can change when self-contained link representations are moved. Changing the context can change the interpretation of links when they have no explicit anchor, or when they use relative URIs. Applications may choose to ignore links that have no explicit anchor or that use relative URIs when these are exchanged in stand-alone resources.
- * The model introduced in this specification supports "3rd party links", where one party can provide links that have another party's resource as an anchor. Depending on the link semantics and the application context, it is important to verify that there is sufficient trust in that 3rd party to allow it to provide these links. Applications may choose to treat 3rd party links differently than cases where a resource and the links for that resource are provided by the same party.

10. 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>>.
- [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>>.
- [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>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.

- [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,
<<https://www.w3.org/TR/2014/REC-json-ld-20140116>>.
- [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>>.
- [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>>.
- [RFC8187] Reschke, J., "Indicating Character Encoding and Language
for HTTP Header Field Parameters", RFC 8187,
DOI 10.17487/RFC8187, September 2017,
<<https://www.rfc-editor.org/info/rfc8187>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
Interchange Format", STD 90, RFC 8259,
DOI 10.17487/RFC8259, December 2017,
<<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288,
DOI 10.17487/RFC8288, October 2017,
<<https://www.rfc-editor.org/info/rfc8288>>.
- [I-D.ietf-httpbis-semantic]
Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP
Semantics", Work in Progress, Internet-Draft, draft-ietf-
httpbis-semantic-19, 12 September 2021,
<[https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-
semantic-19](https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantic-19)>.

11. Informative References

- [W3C.REC-rdf11-concepts-20140225]
Cyganiak, R., Wood, D., and M. Lanthaler, "RDF 1.1
Concepts and Abstract Syntax", World Wide Web Consortium
Recommendation REC-rdf11-concepts-20140225, 25 February
2014,
<<https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988,
DOI 10.17487/RFC5988, October 2010,
<<https://www.rfc-editor.org/info/rfc5988>>.

- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6906] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <<https://www.rfc-editor.org/info/rfc6906>>.
- [RFC7284] Lanthaler, M., "The Profile URI Registry", RFC 7284, DOI 10.17487/RFC7284, June 2014, <<https://www.rfc-editor.org/info/rfc7284>>.
- [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>>.
- [DCMI-TERMS] Initiative, D. C. M., "DCMI Metadata Terms", 2020, <<https://www.dublincore.org/specifications/dublin-core/dcmi-terms/>>.

Appendix A. JSON-LD Context

A set of links rendered according to the JSON serialization defined in Section 4.2 can be interpreted as RDF triples by adding a JSON-LD context [W3C.REC-json-ld-20140116] that maps the JSON keys to corresponding Linked Data terms. And, as per [W3C.REC-json-ld-20140116] section 6.8 (<https://www.w3.org/TR/2014/REC-json-ld-20140116/#interpreting-json-as-json-ld>), when delivering a link set that is rendered according to the "application/linkset+json" media type to a user agent, a server can convey the availability of such a JSON-LD context by using a link with the relation type "http://www.w3.org/ns/json-ld#context" in the HTTP "Link" header.

Figure 19 shows the response to an HTTP GET against the URI of a link set resource and illustrates this approach to support discovery of a JSON-LD Context. The example is inspired by the GS1 implementation and shows a link set that uses relation types from the GS1 vocabulary at <<https://www.gs1.org/voc/>> that are expressed as HTTP URIs.

```
HTTP/1.1 200 OK
Date: Mon, 11 Oct 2021 10:48:22 GMT
Server: Apache-Coyote/1.1
Content-Type: application/linkset+json
Link: <https://example.org/contexts/linkset.jsonld>
      ; rel="http://www.w3.org/ns/json-ld#context"
```

```
; type="application/ld+json"
Content-Length: 1532

{
  "linkset": [
    {
      "anchor": "https://id.gsl.org/01/09506000149301",
      "https://gsl.org/voc/pip": [
        {
          "href": "https://example.com/en/defaultPage",
          "hreflang": [
            "en"
          ],
          "type": "text/html",
          "title": "Product information"
        },
        {
          "href": "https://example.com/fr/defaultPage",
          "hreflang": [
            "fr"
          ],
          "title": "Information produit"
        }
      ],
      "https://gsl.org/voc/whatsInTheBox": [
        {
          "href": "https://example.com/en/packContents/GB",
          "hreflang": [
            "en"
          ],
          "title": "What's in the box?"
        },
        {
          "href": "https://example.com/fr/packContents/FR",
          "hreflang": [
            "fr"
          ],
          "title": "Qu'y a-t-il dans la boite?"
        },
        {
          "href": "https://example.com/fr/packContents/CH",
          "hreflang": [
            "fr"
          ],
          "title": "Qu'y a-t-il dans la boite?"
        }
      ],
      "https://gsl.org/voc/relatedVideo": [
```

```
{
  "href": "https://video.example",
  "hreflang": [
    "en",
    "fr"
  ],
  "title*": [
    {
      "value": "See it in action!",
      "language": "en"
    },
    {
      "value": "Voyez-le en action!",
      "language": "fr"
    }
  ]
}
]
```

Figure 19: Using a typed link to support discovery of a JSON-LD Context for a Set of Links

In order to obtain the JSON-LD Context conveyed by the server, the user agent issues an HTTP GET against the link target of the link with the "http://www.w3.org/ns/json-ld#context" relation type. The response to this GET is shown in Figure 20. This particular JSON-LD context maps "application/linkset+json" representations of link sets to Dublin Core Terms [DCMI-TERMS]. Note that the "linkset" entry in the JSON-LD context is introduced to support links with the "linkset" relation type in link sets.


```

HTTP/1.1 200 OK
Content-Type: application/ld+json
Content-Length: 658

{
  "@context": [
    {
      "@version": 1.1,
      "@vocab": "https://gs1.org/voc/",
      "anchor": "@id",
      "href": "@id",
      "linkset": {
        "@id": "@graph",
        "@context": {
          "linkset": "linkset"
        }
      },
      "title": {
        "@id": "http://purl.org/dc/terms/title"
      },
      "title*": {
        "@id": "http://purl.org/dc/terms/title"
      },
      "type": {
        "@id": "http://purl.org/dc/terms/format"
      }
    },
    {
      "language": "@language",
      "value": "@value",
      "hreflang": {
        "@id": "http://purl.org/dc/terms/language",
        "@container": "@set"
      }
    }
  ]
}

```

Figure 20: JSON-LD Context mapping to Dublin Core Terms

Applying the JSON-LD context of Figure 20 to the link set of Figure 19 allows transforming the "application/linkset+json" link set to an RDF link set. Figure 21 shows the latter represented by means of the "text/turtle" RDF serialization.

```
<https://example.com/en/defaultPage>
  <http://purl.org/dc/terms/format>
    "text/html" .
<https://example.com/en/defaultPage>
  <http://purl.org/dc/terms/language>
    "en" .
<https://example.com/en/defaultPage>
  <http://purl.org/dc/terms/title>
    "Product information" .
<https://example.com/en/packContents/GB>
  <http://purl.org/dc/terms/language>
    "en" .
<https://example.com/en/packContents/GB>
  <http://purl.org/dc/terms/title>
    "What's in the box?" .
<https://example.com/fr/defaultPage>
  <http://purl.org/dc/terms/language>
    "fr" .
<https://example.com/fr/defaultPage>
  <http://purl.org/dc/terms/title>
    "Information produit" .
<https://example.com/fr/packContents/CH>
  <http://purl.org/dc/terms/language>
    "fr" .
<https://example.com/fr/packContents/CH>
  <http://purl.org/dc/terms/title>
    "Qu'y a-t-il dans la boîte?" .
<https://example.com/fr/packContents/FR>
  <http://purl.org/dc/terms/language>
    "fr" .
<https://example.com/fr/packContents/FR>
  <http://purl.org/dc/terms/title>
    "Qu'y a-t-il dans la boîte?" .
<https://id.gsl.org/01/09506000149301>
  <https://gsl.org/voc/pip>
    <https://example.com/en/defaultPage> .
<https://id.gsl.org/01/09506000149301>
  <https://gsl.org/voc/pip>
    <https://example.com/fr/defaultPage> .
<https://id.gsl.org/01/09506000149301>
  <https://gsl.org/voc/relatedVideo>
    <https://video.example> .
<https://id.gsl.org/01/09506000149301>
  <https://gsl.org/voc/whatsInTheBox>
    <https://example.com/en/packContents/GB> .
<https://id.gsl.org/01/09506000149301>
  <https://gsl.org/voc/whatsInTheBox>
    <https://example.com/fr/packContents/CH> .
```

```
<https://id.gsl.org/01/09506000149301>
  <https://gsl.org/voc/whatsInTheBox>
  <https://example.com/fr/packContents/FR> .
<https://video.example>
  <http://purl.org/dc/terms/language>
  "en" .
<https://video.example>
  <http://purl.org/dc/terms/language>
  "fr" .
<https://video.example>
  <http://purl.org/dc/terms/title>
  "See it in action!"@en .
<https://video.example>
  <http://purl.org/dc/terms/title>
  "Voyez-le en action!"@fr .
```

Figure 21: RDF serialization of the link set resulting from applying the JSON-LD context

Appendix B. Implementation Status

This section is to be removed before publishing 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 RFC 7942 [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".

B.1. GS1

GS1 is a provider of identifiers, most famously seen in EAN/UPC barcodes for retail and healthcare products, and manages an ecology of services and standards to leverage them at a global scale. GS1 has indicated that it will fully implement this "linkset" specification as a means to allow requesting and representing links pertaining to products, shipments, assets and locations. The current GS1 Digital Link specification makes an informative reference to version 03 of the "linkset" I-D, mentions the formal adoption of that I-D by the IETF HTTPAPI Working Group, and indicates it being on track to achieve RFC status. The GS1 Digital Link specification adopts the JSON format specified by the I-D and mentions future plans to also support the Link header format as well as their respective media types, neither of which have changed since version 03.

B.2. FAIR Signposting Profile

The FAIR Signposting Profile is a community specification aimed at improving machine navigation of scholarly objects on the web through the use of typed web links pointing at e.g. web resources that are part of a specific object, persistent identifiers for the object and its authors, license information pertaining to the object. The specification encourages the use of Linksets and initial implementations are ongoing, for example, for the open source Dataverse data repository platform that was initiated by Harvard University and is meanwhile used by research institutions, worldwide.

B.3. Open Journal Systems (OJS)

Open Journal Systems (OJS) is an open-source software for the management of peer-reviewed academic journals, and is created by the Public Knowledge Project (PKP), released under the GNU General Public License. Open Journal Systems (OJS) is a journal management and publishing system that has been developed by PKP through its federally funded efforts to expand and improve access to research.

The OJS platform has implemented "linkset" support as an alternative way to provide links when there are more than a configured limit (they consider using about 10 as a good default, for testing purpose it is currently set to 8).

Acknowledgements

Thanks for comments and suggestions provided by Phil Archer, Dominique Guinard, Mark Nottingham, Julian Reschke, Rob Sanderson, Stian Soiland-Reyes, Sarven Capadisli, and Addison Phillips.

Authors' Addresses

Erik Wilde
Axway
Email: erik.wilde@dret.net
URI: <http://dret.net/netdret/>

Herbert Van de Sompel
Data Archiving and Networked Services
Email: herbert.van.de.sompel@dans.knaw.nl
URI: <https://orcid.org/0000-0002-0715-6126>

HTTPAPI
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

R. Polli
Team Digitale, Italian Government
A. Martinez
Red Hat
7 March 2022

RateLimit Fields for HTTP
draft-ietf-httpapi-ratelimit-headers-03

Abstract

This document defines the RateLimit-Limit, RateLimit-Remaining, RateLimit-Reset fields for HTTP, thus allowing servers to publish current service limits and clients to shape their request policy and avoid being throttled out.

Note to Readers

RFC EDITOR: please remove this section before publication

Discussion of this draft takes place on the HTTP working group mailing list (httpapi@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/httpapi/> (<https://mailarchive.ietf.org/arch/browse/httpapi/>).

The source code and issues list for this draft can be found at <https://github.com/ietf-wg-httpapi/ratelimit-headers> (<https://github.com/ietf-wg-httpapi/ratelimit-headers>).

References to ThisRFC in the IANA Considerations section would be replaced with the RFC number when assigned.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Goals	4
1.2. Notational Conventions	5
2. Expressing rate-limit policies	5
2.1. Time window	5
2.2. Service limit	5
2.3. Quota policy	6
3. Providing RateLimit fields	7
3.1. Performance considerations	8
4. Receiving RateLimit fields	9
4.1. Intermediaries	10
4.2. Caching	10
5. Fields definition	10
5.1. RateLimit-Limit	10
5.2. RateLimit-Remaining	11
5.3. RateLimit-Reset	12
6. Security Considerations	13
6.1. Throttling does not prevent clients from issuing requests	13
6.2. Information disclosure	13
6.3. Remaining quota-units are not granted requests	13
6.4. Reliability of RateLimit-Reset	13
6.5. Resource exhaustion	14
6.6. Denial of Service	14
7. IANA Considerations	15
7.1. RateLimit Parameters Registration	15
8. References	16
8.1. Normative References	16
8.2. Informative References	17
Appendix A. Rate-limiting and quotas	17

A.1. Interoperability issues	18
Appendix B. Examples	19
B.1. Unparameterized responses	19
B.1.1. Throttling information in responses	19
B.1.2. Use in conjunction with custom fields	20
B.1.3. Use for limiting concurrency	21
B.1.4. Use in throttled responses	22
B.2. Parameterized responses	22
B.2.1. Throttling window specified via parameter	22
B.2.2. Dynamic limits with parameterized windows	23
B.2.3. Dynamic limits for pushing back and slowing down	23
B.3. Dynamic limits for pushing back with Retry-After and slow down	24
B.3.1. Missing Remaining information	25
B.3.2. Use with multiple windows	26
FAQ	27
RateLimit fields currently used on the web	30
Acknowledgements	31
Changes	31
Since draft-ietf-httpapi-ratelimit-headers-01	31
Since draft-ietf-httpapi-ratelimit-headers-00	32
Authors' Addresses	32

1. Introduction

The widespreading of HTTP as a distributed computation protocol requires an explicit way of communicating service status and usage quotas.

This was partially addressed by the Retry-After header field defined in [SEMANTICS] to be returned in 429 Too Many Requests (see [STATUS429]) or 503 Service Unavailable responses.

Widely deployed quota mechanisms limit the number of acceptable requests in a given time window, e.g. 10 requests per second; currently, there is no standard way to communicate service quotas so that the client can throttle its requests and prevent 4xx or 5xx responses. See Appendix A for further information on the current usage of rate limiting in HTTP.

This document defines syntax and semantics for the following fields:

- * RateLimit-Limit: containing the requests quota in the time window;
- * RateLimit-Remaining: containing the remaining requests quota in the current window;

- * **RateLimit-Reset:** containing the time remaining in the current window, specified in seconds.

The behavior of **RateLimit-Reset** is compatible with the **delay-seconds** notation of **Retry-After**.

The fields definition allows to describe complex policies, including the ones using multiple and variable time windows and dynamic quotas, or implementing concurrency limits.

1.1. Goals

The goals of the **RateLimit** fields are:

Interoperability: Standardization of the names and semantics of **rate-limit** headers to ease their enforcement and adoption;

Resiliency: Improve resiliency of HTTP infrastructure by providing clients with information useful to throttle their requests and prevent 4xx or 5xx responses;

Documentation: Simplify API documentation by eliminating the need to include detailed quota limits and related fields in API documentation.

The following features are out of the scope of this document:

Authorization: **RateLimit** fields are not meant to support authorization or other kinds of access controls.

Throttling scope: This specification does not cover the throttling scope, that may be the given resource-target, its parent path or the whole Origin (see Section 7 of [RFC6454]). This can be addressed using extensibility mechanisms such as the parameter registry Section 7.1.

Response status code: **RateLimit** fields may be returned in both successful (see Section 15.3 of [SEMANTICS]) and non-successful responses. This specification does not cover whether non Successful responses count on quota usage, nor it mandates any correlation between the **RateLimit** values and the returned status code.

Throttling policy: This specification does not mandate a specific throttling policy. The values published in the fields, including the window size, can be statically or dynamically evaluated.

Service Level Agreement: Conveyed quota hints do not imply any

service guarantee. Server is free to throttle respectful clients under certain circumstances.

1.2. 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 document uses the Augmented BNF defined in [RFC5234] and updated by [RFC7405] along with the "#rule" extension defined in Section 5.6.1 of [SEMANTICS].

The term Origin is to be interpreted as described in Section 7 of [RFC6454].

This specification uses Structured Fields [SF] to specify syntax.

The terms sf-list, sf-item, sf-string, sf-token, sf-integer, bare-item and key refer to the structured types defined therein.

2. Expressing rate-limit policies

2.1. Time window

Rate limit policies limit the number of acceptable requests in a given time window.

A time window is expressed in seconds, using the following syntax:

```
time-window = delay-seconds  
delay-seconds = sf-integer
```

Where delay-seconds is a non-negative sf-integer compatible with the "delay-seconds" rule defined in Section 10.2.3 of [SEMANTICS].

Subsecond precision is not supported.

2.2. Service limit

The service-limit is a value associated to the maximum number of requests that the server is willing to accept from one or more clients on a given basis (originating IP, authenticated user, geographical, ..) during a time-window as defined in Section 2.1.

The service-limit is expressed in quota-units and has the following syntax:

```
service-limit = quota-units
quota-units = sf-integer
```

where quota-units is a non-negative sf-integer.

The service-limit SHOULD match the maximum number of acceptable requests.

The service-limit MAY differ from the total number of acceptable requests when weight mechanisms, bursts, or other server policies are implemented.

If the service-limit does not match the maximum number of acceptable requests the relation with that SHOULD be communicated out-of-band.

Example: A server could

- * count once requests like /books/{id}
- * count twice search requests like /books?author=WuMing

so that we have the following counters

```
GET /books/123           ; service-limit=4, remaining: 3, status=200
GET /books?author=WuMing ; service-limit=4, remaining: 1, status=200
GET /books?author=Eco    ; service-limit=4, remaining: 0, status=429
```

2.3. Quota policy

This specification allows describing a quota policy with the following syntax:

```
quota-policy = sf-item
```

where the associated bare-item is a service-limit and parameters are supported.

The following parameters are defined:

- w: The REQUIRED "w" parameter specifies a time window. Its syntax is a "time-window" defined in Section 2.1.

Other parameters are allowed and can be regarded as comments. They ought to be registered within the "Hypertext Transfer Protocol (HTTP) RateLimit Parameters Registry", as described in Section 7.1.

An example policy of 100 quota-units per minute.

```
100;w=60
```

The definition of a quota-policy does not imply any specific distribution of quota-units over time. Such service specific details can be conveyed as parameters.

Two policy examples containing further details via custom parameters

```
100;w=60;comment="fixed window"  
12;w=1;burst=1000;policy="leaky bucket"
```

To avoid clashes, implementers SHOULD prefix unregistered parameters with an x-`<vendor>` identifier, e.g. x-acme-policy, x-acme-burst. While it is useful to define a clear syntax and semantics even for custom parameters, it is important to note that user agents are not required to process quota policy information.

3. Providing RateLimit fields

A server uses the RateLimit response fields defined in this document to communicate its quota policies according to the following rules:

- * RateLimit-Limit and RateLimit-Reset are REQUIRED;
- * RateLimit-Remaining is RECOMMENDED.

The returned values refers to the metrics used to evaluate if the current request respects the quota policy and MAY not apply to subsequent requests.

Example: a successful response with the following fields

```
RateLimit-Limit: 10  
RateLimit-Remaining: 1  
RateLimit-Reset: 7
```

does not guarantee that the next request will be successful. Server metrics may be subject to other conditions like the one shown in the example from Section 2.2.

A server MAY return RateLimit response fields independently of the response status code. This includes throttled responses.

This document does not mandate any correlation between the RateLimit values and the returned status code.

Servers should be careful in returning RateLimit fields in redirection responses (e.g. 3xx status codes) because a low RateLimit-Remaining value could prevent the client from issuing requests. For example, given the rate limiting fields below, a client could decide to wait 10 seconds before following the Location header, because RateLimit-Remaining is 0.

```
HTTP/1.1 301 Moved Permanently
Location: /foo/123
RateLimit-Remaining: 0
RateLimit-Limit: 10
RateLimit-Reset: 10
```

If a response contains both the Retry-After and the RateLimit-Reset fields, the value of RateLimit-Reset SHOULD reference the same point in time as Retry-After.

When using a policy involving more than one time-window, the server MUST reply with the RateLimit fields related to the window with the lower RateLimit-Remaining values.

A service returning RateLimit fields MUST NOT convey values exposing an unwanted volume of requests and SHOULD implement mechanisms to cap the ratio between RateLimit-Remaining and RateLimit-Reset (see Section 6.5); this is especially important when quota-policies use a large time-window.

Under certain conditions, a server MAY artificially lower RateLimit field values between subsequent requests, e.g. to respond to Denial of Service attacks or in case of resource saturation.

Servers usually establish whether the request is in-quota before creating a response, so the RateLimit field values should be already available in that moment. Nonetheless servers MAY decide to send the RateLimit fields in a trailer section.

To ease the migration from existing rate limit headers, a server SHOULD be able to provide the RateLimit-Limit field even without the optional quota-policy section.

3.1. Performance considerations

Servers are not required to return RateLimit fields in every response, and clients need to take this into account. For example, an implementer concerned with performance might provide RateLimit fields only when a given quota is going to expire.

Implementers concerned with response fields' size, might take into account their ratio with respect to the payload data, or use header-compression http features such as [HPACK].

4. Receiving RateLimit fields

A client **MUST** process the received RateLimit fields.

A client **MUST** validate the values received in the RateLimit fields before using them and check if there are significant discrepancies with the expected ones. This includes a RateLimit-Reset moment too far in the future or a service-limit too high.

A client receiving RateLimit fields **MUST NOT** assume that subsequent responses contain the same RateLimit fields, or any RateLimit fields at all.

Malformed RateLimit fields **MAY** be ignored.

A client **SHOULD NOT** exceed the quota-units expressed in RateLimit-Remaining before the time-window expressed in RateLimit-Reset.

A client **MAY** still probe the server if the RateLimit-Reset is considered too high.

The value of RateLimit-Reset is generated at response time: a client aware of a significant network latency **MAY** behave accordingly and use other information (e.g. the Date response header field, or otherwise gathered metrics) to better estimate the RateLimit-Reset moment intended by the server.

The quota-policy values and comments provided in RateLimit-Limit are informative and **MAY** be ignored.

If a response contains both the RateLimit-Reset and Retry-After fields, Retry-After **MUST** take precedence and RateLimit-Reset **MAY** be ignored.

This specification does not mandate a specific throttling behavior and implementers can adopt their preferred policies, including:

- * slowing down or preemptively back-off their request rate when approaching quota limits;
- * consuming all the quota according to the exposed limits and then wait.

4.1. Intermediaries

This section documents the considerations advised in Section 16.3.2 of [SEMANTICS].

An intermediary that is not part of the originating service infrastructure and is not aware of the quota-policy semantic used by the Origin Server SHOULD NOT alter the RateLimit fields' values in such a way as to communicate a more permissive quota-policy; this includes removing the RateLimit fields.

An intermediary MAY alter the RateLimit fields in such a way as to communicate a more restrictive quota-policy when:

- * it is aware of the quota-unit semantic used by the Origin Server;
- * it implements this specification and enforces a quota-policy which is more restrictive than the one conveyed in the fields.

An intermediary SHOULD forward a request even when presuming that it might not be serviced; the service returning the RateLimit fields is the sole responsible of enforcing the communicated quota-policy, and it is always free to service incoming requests.

This specification does not mandate any behavior on intermediaries respect to retries, nor requires that intermediaries have any role in respecting quota-policies. For example, it is legitimate for a proxy to retransmit a request without notifying the client, and thus consuming quota-units.

4.2. Caching

As is the ordinary case for HTTP caching ([RFC7234]), a response with RateLimit fields might be cached and re-used for subsequent requests. A cached RateLimit response does not modify quota counters but could contain stale information. Clients interested in determining the freshness of the RateLimit fields could rely on fields such as Date and on the time-window of a quota-policy.

5. Fields definition

The following RateLimit response fields are defined

5.1. RateLimit-Limit

The RateLimit-Limit response field indicates the service-limit associated to the client in the current time-window.

If the client exceeds that limit, it MAY not be served.

The field is a List Structured Field of positive length. The first member is named `expiring-limit` and its syntax is `service-limit`, while the syntax of the other optional members is `quota-policy`

RateLimit-Limit = sf-list

The `expiring-limit` value MUST be set to the `service-limit` that is closer to reach its limit.

The `quota-policy` is defined in Section 2.3, and its values are informative.

RateLimit-Limit: 100

A time-window associated to `expiring-limit` can be communicated via an optional `quota-policy` value, like shown in the following example

RateLimit-Limit: 100, 100;w=10

If the `expiring-limit` is not associated to a time-window, the time-window MUST either be:

- * inferred by the value of `RateLimit-Reset` at the moment of the reset, or
- * communicated out-of-band (e.g. in the documentation).

Policies using multiple quota limits MAY be returned using multiple `quota-policy` items, like shown in the following two examples:

RateLimit-Limit: 10, 10;w=1, 50;w=60, 1000;w=3600, 5000;w=86400
RateLimit-Limit: 10, 10;w=1;burst=1000, 1000;w=3600

This field MUST NOT occur multiple times and can be sent in a trailer section.

5.2. RateLimit-Remaining

The `RateLimit-Remaining` response field indicates the remaining quota-units defined in Section 2.2 associated to the client.

The field is an Integer Structured Field and its value is

RateLimit-Remaining = quota-units

This field MUST NOT occur multiple times and can be sent in a trailer section.

Clients MUST NOT assume that a positive RateLimit-Remaining value is a guarantee that further requests will be served.

A low RateLimit-Remaining value is like a yellow traffic-light for either the number of requests issued in the time-window or the request throughput: the red light may arrive suddenly (see Section 3).

One example of RateLimit-Remaining use is below.

RateLimit-Remaining: 50

5.3. RateLimit-Reset

The RateLimit-Reset response field indicates either

- * the number of seconds until the quota resets.

The field is an Integer Structured Field and its value is

RateLimit-Reset = delay-seconds

The delay-seconds format is used because:

- * it does not rely on clock synchronization and is resilient to clock adjustment and clock skew between client and server (see Section 5.6.7 of [SEMANTICS]);
- * it mitigates the risk related to thundering herd when too many clients are serviced with the same timestamp.

This field MUST NOT occur multiple times and can be sent in a trailer section.

An example of RateLimit-Reset use is below.

RateLimit-Reset: 50

The client MUST NOT assume that all its service-limit will be restored after the moment referenced by RateLimit-Reset. The server MAY arbitrarily alter the RateLimit-Reset value between subsequent requests e.g. in case of resource saturation or to implement sliding window policies.

6. Security Considerations

6.1. Throttling does not prevent clients from issuing requests

This specification does not prevent clients to make over-quota requests.

Servers should always implement mechanisms to prevent resource exhaustion.

6.2. Information disclosure

Servers should not disclose to untrusted parties operational capacity information that can be used to saturate its infrastructural resources.

While this specification does not mandate whether non 2xx responses consume quota, if 401 and 403 responses count on quota a malicious client could probe the endpoint to get traffic information of another user.

As intermediaries might retransmit requests and consume quota-units without prior knowledge of the User Agent, RateLimit fields might reveal the existence of an intermediary to the User Agent.

6.3. Remaining quota-units are not granted requests

RateLimit-* fields convey hints from the server to the clients in order to avoid being throttled out.

Clients MUST NOT consider the quota-units returned in RateLimit-Remaining as a service level agreement.

In case of resource saturation, the server MAY artificially lower the returned values or not serve the request regardless of the advertised quotas.

6.4. Reliability of RateLimit-Reset

Consider that service-limit may not be restored after the moment referenced by RateLimit-Reset, and the RateLimit-Reset value should not be considered fixed nor constant.

Subsequent requests may return a higher RateLimit-Reset value to limit concurrency or implement dynamic or adaptive throttling policies.

6.5. Resource exhaustion

When returning RateLimit-Reset you must be aware that many throttled clients may come back at the very moment specified.

This is true for Retry-After too.

For example, if the quota resets every day at 18:00:00 and your server returns the RateLimit-Reset accordingly

```
Date: Tue, 15 Nov 1994 08:00:00 GMT
RateLimit-Reset: 36000
```

there's a high probability that all clients will show up at 18:00:00.

This could be mitigated by adding some jitter to the field-value.

Resource exhaustion issues can be associated with quota policies using a large time-window, because a user agent by chance or on purpose might consume most of its quota-units in a significantly shorter interval.

This behavior can be even triggered by the provided RateLimit fields. The following example describes a service with an unconsumed quota-policy of 10000 quota-units per 1000 seconds.

```
RateLimit-Limit: 10000, 10000;w=1000
RateLimit-Remaining: 10000
RateLimit-Reset: 10
```

A client implementing a simple ratio between RateLimit-Remaining and RateLimit-Reset could infer an average throughput of 1000 quota-units per second, while RateLimit-Limit conveys a quota-policy with an average of 10 quota-units per second. If the service cannot handle such load, it should return either a lower RateLimit-Remaining value or an higher RateLimit-Reset value. Moreover, complementing large time-window quota-policies with a short time-window one mitigates those risks.

6.6. Denial of Service

RateLimit fields may assume unexpected values by chance or purpose. For example, an excessively high RateLimit-Remaining value may be:

- * used by a malicious intermediary to trigger a Denial of Service attack or consume client resources boosting its requests;
- * passed by a misconfigured server;

or an high RateLimit-Reset value could inhibit clients to contact the server.

Clients MUST validate the received values to mitigate those risks.

7. IANA Considerations

IANA is requested to update one registry and create one new registry.

Please add the following entries to the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]):

Field Name	Status	Specification
RateLimit-Limit	permanent	Section 5.1 of ThisRFC
RateLimit-Remaining	permanent	Section 5.2 of ThisRFC
RateLimit-Reset	permanent	Section 5.3 of ThisRFC

Table 1

7.1. RateLimit Parameters Registration

IANA is requested to create a new registry to be called "Hypertext Transfer Protocol (HTTP) RateLimit Parameters Registry", to be located at <https://www.iana.org/assignments/http-ratelimit-parameters> (<https://www.iana.org/assignments/http-ratelimit-parameters>). Registration is done on the advice of a Designated Expert, appointed by the IESG or their delegate. All entries are Specification Required ([IANA], Section 4.6).

Registration requests consist of the following information:

- * Parameter name: The parameter name, conforming to [SF].
- * Field name: The RateLimit field for which the parameter is registered. If a parameter is intended to be used with multiple fields, it has to be registered for each one.
- * Description: A brief description of the parameter.
- * Specification document: A reference to the document that specifies the parameter, preferably including a URI that can be used to retrieve a copy of the document.

- * Comments (optional): Any additional information that can be useful.

The initial contents of this registry should be:

Field Name	Parameter name	Description	Specification	Comments (optional)
RateLimit-Limit	w	Time window	Section 2.3 of ThisRFC	

Table 2

8. References

8.1. Normative References

- [IANA] 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>>.
- [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>>.
- [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/rfc/rfc5234>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/rfc/rfc6454>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <<https://www.rfc-editor.org/rfc/rfc7405>>.
- [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>>.

[SEMANTICS]

Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>>.

[SF]

Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/rfc/rfc8941>>.

8.2. Informative References

[HPACK]

Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <<https://www.rfc-editor.org/rfc/rfc7541>>.

[RFC3339]

Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/rfc/rfc3339>>.

[RFC6585]

Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <<https://www.rfc-editor.org/rfc/rfc6585>>.

[RFC7234]

Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/rfc/rfc7234>>.

[STATUS429]

Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, DOI 10.17487/RFC6525, February 2012, <<https://www.rfc-editor.org/rfc/rfc6525>>.

[UNIX]

The Open Group, "The Single UNIX Specification, Version 2 - 6 Vol Set for UNIX 98", February 1997.

Appendix A. Rate-limiting and quotas

Servers use quota mechanisms to avoid systems overload, to ensure an equitable distribution of computational resources or to enforce other policies - e.g. monetization.

A basic quota mechanism limits the number of acceptable requests in a given time window, e.g. 10 requests per second.

When quota is exceeded, servers usually do not serve the request replying instead with a 4xx HTTP status code (e.g. 429 or 403) or adopt more aggressive policies like dropping connections.

Quotas may be enforced on different basis (e.g. per user, per IP, per geographic area, ..) and at different levels. For example, an user may be allowed to issue:

- * 10 requests per second;
- * limited to 60 requests per minute;
- * limited to 1000 requests per hour.

Moreover system metrics, statistics and heuristics can be used to implement more complex policies, where the number of acceptable requests and the time window are computed dynamically.

To help clients throttling their requests, servers may expose the counters used to evaluate quota policies via HTTP header fields.

Those response headers may be added by HTTP intermediaries such as API gateways and reverse proxies.

On the web we can find many different rate-limit headers, usually containing the number of allowed requests in a given time window, and when the window is reset.

The common choice is to return three headers containing:

- * the maximum number of allowed requests in the time window;
- * the number of remaining requests in the current window;
- * the time remaining in the current window expressed in seconds or as a timestamp;

A.1. Interoperability issues

A major interoperability issue in throttling is the lack of standard headers, because:

- * each implementation associates different semantics to the same header field names;
- * header field names proliferates.

User Agents interfacing with different servers may thus need to process different headers, or the very same application interface that sits behind different reverse proxies may reply with different throttling headers.

Appendix B. Examples

B.1. Unparameterized responses

B.1.1. Throttling information in responses

The client exhausted its service-limit for the next 50 seconds. The time-window is communicated out-of-band or inferred by the field values.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 100
Ratelimit-Remaining: 0
Ratelimit-Reset: 50
```

```
{"hello": "world"}
```

Since the field values are not necessarily correlated with the response status code, a subsequent request is not required to fail. The example below shows that the server decided to serve the request even if RateLimit-Remaining is 0. Another server, or the same server under other load conditions, could have decided to throttle the request instead.

Request:

```
GET /items/456 HTTP/1.1
Host: api.example
```

Response:


```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 100
Ratelimit-Remaining: 0
Ratelimit-Reset: 48
```

```
{"still": "successful"}
```

B.1.2. Use in conjunction with custom fields

The server uses two custom fields, namely `acme-RateLimit-DayLimit` and `acme-RateLimit-HourLimit` to expose the following policy:

- * 5000 daily quota-units;
- * 1000 hourly quota-units.

The client consumed 4900 quota-units in the first 14 hours.

Despite the next hourly limit of 1000 quota-units, the closest limit to reach is the daily one.

The server then exposes the `RateLimit-*` fields to inform the client that:

- * it has only 100 quota-units left;
- * the window will reset in 10 hours.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
acme-RateLimit-DayLimit: 5000
acme-RateLimit-HourLimit: 1000
RateLimit-Limit: 5000
RateLimit-Remaining: 100
RateLimit-Reset: 36000
```

```
{"hello": "world"}
```

B.1.3. Use for limiting concurrency

Throttling fields may be used to limit concurrency, advertising limits that are lower than the usual ones in case of saturation, thus increasing availability.

The server adopted a basic policy of 100 quota-units per minute, and in case of resource exhaustion adapts the returned values reducing both RateLimit-Limit and RateLimit-Remaining.

After 2 seconds the client consumed 40 quota-units

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 100
RateLimit-Remaining: 60
RateLimit-Reset: 58
```

```
{"elapsed": 2, "issued": 40}
```

At the subsequent request - due to resource exhaustion - the server advertises only RateLimit-Remaining: 20.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 100
RateLimit-Remaining: 20
RateLimit-Reset: 56
```

```
{"elapsed": 4, "issued": 41}
```

B.1.4. Use in throttled responses

A client exhausted its quota and the server throttles it sending Retry-After.

In this example, the values of Retry-After and RateLimit-Reset reference the same moment, but this is not a requirement.

The 429 Too Many Requests HTTP status code is just used as an example.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
Date: Mon, 05 Aug 2019 09:27:00 GMT
Retry-After: Mon, 05 Aug 2019 09:27:05 GMT
RateLimit-Reset: 5
RateLimit-Limit: 100
Ratelimit-Remaining: 0
```

```
{
  "title": "Too Many Requests",
  "status": 429,
  "detail": "You have exceeded your quota"
}
```

B.2. Parameterized responses

B.2.1. Throttling window specified via parameter

The client has 99 quota-units left for the next 50 seconds. The time-window is communicated by the w parameter, so we know the throughput is 100 quota-units per minute.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 100, 100;w=60
Ratelimit-Remaining: 99
Ratelimit-Reset: 50
```

```
{"hello": "world"}
```

B.2.2. Dynamic limits with parameterized windows

The policy conveyed by `RateLimit-Limit` states that the server accepts 100 quota-units per minute.

To avoid resource exhaustion, the server artificially lowers the actual limits returned in the throttling headers.

The `RateLimit-Remaining` then advertises only 9 quota-units for the next 50 seconds to slow down the client.

Note that the server could have lowered even the other values in `RateLimit-Limit`: this specification does not mandate any relation between the field values contained in subsequent responses.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 10, 100;w=60
Ratelimit-Remaining: 9
Ratelimit-Reset: 50
```

```
{
  "status": 200,
  "detail": "Just slow down without waiting."
}
```

B.2.3. Dynamic limits for pushing back and slowing down

Continuing the previous example, let's say the client waits 10 seconds and performs a new request which, due to resource exhaustion, the server rejects and pushes back, advertising `RateLimit-Remaining: 0` for the next 20 seconds.

The server advertises a smaller window with a lower limit to slow down the client for the rest of its original window after the 20 seconds elapse.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
RateLimit-Limit: 0, 15;w=20
Ratelimit-Remaining: 0
Ratelimit-Reset: 20
```

```
{
  "status": 429,
  "detail": "Wait 20 seconds, then slow down!"
}
```

B.3. Dynamic limits for pushing back with Retry-After and slow down

Alternatively, given the same context where the previous example starts, we can convey the same information to the client via Retry-After, with the advantage that the server can now specify the policy's nominal limit and window that will apply after the reset, e.g. assuming the resource exhaustion is likely to be gone by then, so the advertised policy does not need to be adjusted, yet we managed to stop requests for a while and slow down the rest of the current window.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
Retry-After: 20
RateLimit-Limit: 15, 100;w=60
Ratelimit-Remaining: 15
Ratelimit-Reset: 40
```

```
{
  "status": 429,
  "detail": "Wait 20 seconds, then slow down!"
}
```

Note that in this last response the client is expected to honor `Retry-After` and perform no requests for the specified amount of time, whereas the previous example would not force the client to stop requests before the reset time is elapsed, as it would still be free to query again the server even if it is likely to have the request rejected.

B.3.1. Missing Remaining information

The server does not expose `RateLimit-Remaining` values (for example, because the underlying counters are not available). Instead, it resets the limit counter every second.

It communicates to the client the limit of 10 quota-units per second always returning the couple `RateLimit-Limit` and `RateLimit-Reset`.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 10
Ratelimit-Reset: 1
```

```
{"first": "request"}
```

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 Ok
Content-Type: application/json
RateLimit-Limit: 10
Ratelimit-Reset: 1

{"second": "request"}
```

B.3.2. Use with multiple windows

This is a standardized way of describing the policy detailed in Appendix B.1.2:

- * 5000 daily quota-units;
- * 1000 hourly quota-units.

The client consumed 4900 quota-units in the first 14 hours.

Despite the next hourly limit of 1000 quota-units, the closest limit to reach is the daily one.

The server then exposes the RateLimit fields to inform the client that:

- * it has only 100 quota-units left;
- * the window will reset in 10 hours;
- * the expiring-limit is 5000.

Request:

```
GET /items/123 HTTP/1.1
Host: api.example
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
RateLimit-Limit: 5000, 1000;w=3600, 5000;w=86400
RateLimit-Remaining: 100
RateLimit-Reset: 36000

{"hello": "world"}
```

FAQ

RFC Editor: Please remove this section before publication.

1. Why defining standard fields for throttling?

To simplify enforcement of throttling policies.

2. Can I use RateLimit-* in throttled responses (eg with status code 429)?

Yes, you can.

3. Are those specs tied to RFC 6585?

No. [RFC6585] defines the 429 status code and we use it just as an example of a throttled request, that could instead use even 403 or whatever status code. The goal of this specification is to standardize the name and semantic of three ratelimit fields widely used on the internet. Stricter relations with status codes or error response payloads would impose behaviors to all the existing implementations making the adoption more complex.

4. Why don't pass the throttling scope as a parameter?

The word "scope" can have different meanings: for example it can be an URL, or an authorization scope. Since authorization is out of the scope of this document (see Section 1.1), and that we rely only on [SEMANTICS], in Section 1.1 we defined "scope" in terms of URL.

Since clients are not required to process quota policies (see Section 4), we could add a new "RateLimit-Scope" field to this spec. See this discussion on a similar thread (<https://github.com/httpwg/http-core/pull/317#issuecomment-585868767>)

Specific ecosystems can still bake their own prefixed parameters, such as acme-auth-scope or acme-url-scope and ensure that clients process them. This behavior cannot be relied upon when communicating between different ecosystems.

We are open to suggestions: comment on this issue (<https://github.com/ioggstream/draft-polli-ratelimit-headers/issues/70>)

5. Why using delay-seconds instead of a UNIX Timestamp? Why not using subsecond precision?

Using `delay-seconds` aligns with `Retry-After`, which is returned in similar contexts, eg on 429 responses.

Timestamps require a clock synchronization protocol (see Section 5.6.7 of [SEMANTICS]). This may be problematic (e.g. clock adjustment, clock skew, failure of hardcoded clock synchronization servers, IoT devices, ..). Moreover timestamps may not be monotonically increasing due to clock adjustment. See Another NTP client failure story (<https://community.ntppool.org/t/another-ntp-client-failure-story/1014/>)

We did not use subsecond precision because:

- * that is more subject to system clock correction like the one implemented via the `adjtimex()` Linux system call;
- * response-time latency may not make it worth. A brief discussion on the subject is on the `httpwg` ml (<https://lists.w3.org/Archives/Public/ietf-httpwg/2019JulSep/0202.html>)
- * almost all rate-limit headers implementations do not use it.

6. Why not support multiple quota remaining?

While this might be of some value, my experience suggests that overly-complex quota implementations results in lower effectiveness of this policy. This spec allows the client to easily focusing on `RateLimit-Remaining` and `RateLimit-Reset`.

7. Shouldn't I limit concurrency instead of request rate?

You can use this specification to limit concurrency at the HTTP level (see `{#use-for-limiting-concurrency}`) and help clients to shape their requests avoiding being throttled out.

A problematic way to limit concurrency is connection dropping, especially when connections are multiplexed (e.g. HTTP/2) because this results in unserved client requests, which is something we want to avoid.

A semantic way to limit concurrency is to return 503 + `Retry-After` in case of resource saturation (e.g. thrashing, connection queues too long, Service Level Objectives not meet, ..). Saturation conditions can be either dynamic or static: all this is out of the scope for the current document.

8. Do a positive value of RateLimit-Remaining imply any service guarantee for my future requests to be served?

No. FAQ integrated in Section 5.2.

9. Is the quota-policy definition Section 2.3 too complex?

You can always return the simplest form of the 3 fields

```
RateLimit-Limit: 100
RateLimit-Remaining: 50
RateLimit-Reset: 60
```

The key runtime value is the first element of the list: expiring-limit, the others quota-policy are informative. So for the following field:

```
RateLimit-Limit: 100, 100;w=60;burst=1000;comment="sliding window", 5000;w=3600;burst=0;comment="fixed window"
```

the key value is the one referencing the lowest limit: 100

1. Can we use shorter names? Why don't put everything in one field?

The most common syntax we found on the web is X-RateLimit-* and when starting this I-D we opted for it (<https://github.com/ioggstream/draft-polli-ratelimit-headers/issues/34#issuecomment-519366481>)

The basic form of those fields is easily parseable, even by implementers processing responses using technologies like dynamic interpreter with limited syntax.

Using a single field complicates parsing and takes a significantly different approach from the existing ones: this can limit adoption.

1. Why don't mention connections?

Beware of the term "connection": - it is just one possible saturation cause. Once you go that path you will expose other infrastructural details (bandwidth, CPU, .. see Section 6.2) and complicate client compliance; - it is an infrastructural detail defined in terms of server and network rather than the consumed service. This specification protects the services first, and then the infrastructures through client cooperation (see Section 6.1). RateLimit fields enable sending on the same connection different limit values on each response, depending on the policy scope (e.g. per-user, per-custom-key, ..)

2. Can intermediaries alter RateLimit fields?

Generally, they should not because it might result in unserved requests. There are reasonable use cases for intermediaries mangling RateLimit fields though, e.g. when they enforce stricter quota-policies, or when they are an active component of the service. In those case we will consider them as part of the originating infrastructure.

3. Why the w parameter is just informative? Could it be used by a client to determine the request rate?

A non-informative w parameter might be fine in an environment where clients and servers are tightly coupled. Conveying policies with this detail on a large scale would be very complex and implementations would be likely not interoperable. We thus decided to leave w as an informational parameter and only rely on RateLimit-Limit, RateLimit-Remaining and RateLimit-Reset for defining the throttling behavior.

RateLimit fields currently used on the web

RFC Editor: Please remove this section before publication.

Commonly used header field names are:

- * X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset;
- * X-Rate-Limit-Limit, X-Rate-Limit-Remaining, X-Rate-Limit-Reset.

There are variants too, where the window is specified in the header field name, eg:

- * x-ratelimit-limit-minute, x-ratelimit-limit-hour, x-ratelimit-limit-day
- * x-ratelimit-remaining-minute, x-ratelimit-remaining-hour, x-ratelimit-remaining-day

Here are some interoperability issues:

- * X-RateLimit-Remaining references different values, depending on the implementation:
 - seconds remaining to the window expiration
 - milliseconds remaining to the window expiration

- seconds since UTC, in UNIX Timestamp [UNIX]
- a datetime, either IMF-fixdate [SEMANTICS] or [RFC3339]
- * different headers, with the same semantic, are used by different implementers:
 - X-RateLimit-Limit and X-Rate-Limit-Limit
 - X-RateLimit-Remaining and X-Rate-Limit-Remaining
 - X-RateLimit-Reset and X-Rate-Limit-Reset

The semantic of RateLimit-Remaining depends on the windowing algorithm. A sliding window policy for example may result in having a RateLimit-Remaining value related to the ratio between the current and the maximum throughput. e.g.

```
RateLimit-Limit: 12, 12;w=1
RateLimit-Remaining: 6           ; using 50% of throughput, that is 6 units/s
RateLimit-Reset: 1
```

If this is the case, the optimal solution is to achieve

```
RateLimit-Limit: 12, 12;w=1
RateLimit-Remaining: 1         ; using 100% of throughput, that is 12 units/s
RateLimit-Reset: 1
```

At this point you should stop increasing your request rate.

Acknowledgements

Thanks to Willi Schoenborn, Alejandro Martinez Ruiz, Alessandro Ranellucci, Amos Jeffries, Martin Thomson, Erik Wilde and Mark Nottingham for being the initial contributors of these specifications. Kudos to the first community implementers: Aapo Talvensaari, Nathan Friedly and Sanyam Dogra.

In addition to the people above, this document owes a lot to the extensive discussion in the HTTPAPI workgroup, including Rich Salz, Darrel Miller and Julian Reschke.

Changes

__RFC Editor: Please remove this section before publication.__

Since draft-ietf-httpapi-ratelimit-headers-01

- * Update IANA considerations #60
- * Use Structured fields #58
- * Reorganize document #67

Since draft-ietf-httpapi-ratelimit-headers-00

- * Use I-D.httpbis-semantics, which includes referencing delay-seconds instead of delta-seconds. #5

Authors' Addresses

Roberto Polli
Team Digitale, Italian Government
Italy
Email: robipolli@gmail.com

Alejandro Martinez Ruiz
Red Hat
Email: amr@redhat.com

HTTPAPI
Internet-Draft
Obsoletes: 7807 (if approved)
Intended status: Standards Track
Expires: 18 October 2022

M. Nottingham
E. Wilde
S. Dalal
16 April 2022

Problem Details for HTTP APIs
draft-ietf-httpapi-rfc7807bis-02

Abstract

This document defines a "problem detail" to carry machine-readable details of errors in a HTTP response to avoid the need to define new error response formats for HTTP APIs.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at
<https://github.com/ietf-wg-httpapi/rfc7807bis>.

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 18 October 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
2. Requirements	3
3. The Problem Details JSON Object	4
3.1. Members of a Problem Details Object	5
3.1.1. "type"	5
3.1.2. "status"	6
3.1.3. "title"	7
3.1.4. "detail"	7
3.1.5. "instance"	7
3.2. Extension Members	8
4. Defining New Problem Types	8
4.1. Example	10
4.2. Registered Problem Types	10
4.2.1. about:blank	11
5. Security Considerations	11
6. IANA Considerations	12
7. References	12
7.1. Normative References	12
7.2. Informative References	13
Appendix A. JSON Schema for HTTP Problems	14
Appendix B. HTTP Problems and XML	15
Appendix C. Using Problem Details with Other Formats	17
Acknowledgements	18
Authors' Addresses	18

1. Introduction

HTTP status codes (Section 15 of [HTTP]) cannot always convey enough information about errors to be helpful. While humans using Web browsers can often understand an HTML [HTML5] response body, non-human consumers of HTTP APIs have difficulty doing so.

To address that shortcoming, this specification defines simple JSON [RFC8259] and XML [XML] document formats to describe the specifics of problem(s) encountered -- "problem details".

For example, consider a response indicating that the client's account doesn't have enough credit. The API's designer might decide to use the 403 Forbidden status code to inform HTTP-generic software (such as client libraries, caches, and proxies) of the response's general semantics. API-specific problem details (such as the why the server refused the request and the applicable account balance) can be carried in the response content, so that the client can act upon them appropriately (for example, triggering a transfer of more credit into the account).

This specification identifies the specific "problem type" (e.g., "out of credit") with a URI [RFC3986]. HTTP APIs can use URIs under their control to identify problems specific to them, or can reuse existing ones to facilitate interoperability and leverage common semantics (see Section 4.2).

Problem details can contain other information, such as a URI identifying the problem's specific occurrence (effectively giving an identifier to the concept "The time Joe didn't have enough credit last Thursday"), which can be useful for support or forensic purposes.

The data model for problem details is a JSON [RFC8259] object; when serialized as a JSON document, it uses the "application/problem+json" media type. Appendix B defines an equivalent XML format, which uses the "application/problem+xml" media type.

Note that problem details are (naturally) not the only way to convey the details of a problem in HTTP. If the response is still a representation of a resource, for example, it's often preferable to describe the relevant details in that application's format. Likewise, defined HTTP status codes cover many situations with no need to convey extra detail.

This specification's aim is to define common error formats for applications that need one so that they aren't required to define their own, or worse, tempted to redefine the semantics of existing HTTP status codes. Even if an application chooses not to use it to convey errors, reviewing its design can help guide the design decisions faced when conveying errors in an existing format.

2. Requirements

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.

3. The Problem Details JSON Object

The canonical model for problem details is a JSON [RFC8259] object.

When serialized as a JSON document, that format is identified with the "application/problem+json" media type.

For example, an HTTP response carrying JSON problem details:

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json
Content-Language: en
```

```
{
  "type": "https://example.com/probs/out-of-credit",
  "title": "You do not have enough credit.",
  "detail": "Your current balance is 30, but that costs 50.",
  "instance": "/account/12345/mgs/abc",
  "balance": 30,
  "accounts": ["/account/12345",
               "/account/67890"]
}
```

Here, the out-of-credit problem (identified by its type) indicates the reason for the 403 in "title", identifies the specific problem occurrence with "instance", gives occurrence-specific details in "detail", and adds two extensions; "balance" conveys the account's balance, and "accounts" lists links where the account can be topped up.

When designed to accommodate it, problem-specific extensions can allow more than one instance of the same problem type to be conveyed. For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json
Content-Language: en

{
  "type": "https://example.net/validation-error",
  "title": "Your request is not valid.",
  "causes": [
    {
      "detail": "must be a positive integer",
      "problem-pointer": "#/age"
    },
    {
      "detail": "must be 'green', 'red' or 'blue'",
      "problem-pointer": "#/profile/color"
    }
  ]
}
```

The fictional problem type here defines the "causes" extension, an array that describes the details of multiple occurrences. Each member is an object containing "detail" to describe the issue, and "problem-pointer" to locate the problem within the request's content using a JSON Pointer [RFC6901].

When an API encounters multiple problems that do not share the same type, it is RECOMMENDED that the most relevant or urgent problem be represented in the response. While it is possible to create generic "batch" problem types that convey multiple, disparate types, they do not map well into HTTP semantics.

3.1. Members of a Problem Details Object

Problem detail objects can have the following members. If a member's value type does not match the specified type, the member MUST be ignored -- i.e., processing will continue as if the member had not been present.

3.1.1. "type"

The "type" member is a JSON string containing a URI reference [RFC3986] that identifies the problem type. Consumers MUST use the "type" URI (after resolution, if necessary) problem's primary identifier.

When this member is not present, its value is assumed to be "about:blank".

If the type URI is a locator (e.g., those with a "http" or "https" scheme), dereferencing it SHOULD provide human-readable documentation for the problem type (e.g., using HTML [HTML5]). However, consumers SHOULD NOT automatically dereference the type URI, unless they do so when providing information to developers (e.g., when a debugging tool is in use).

When "type" contains a relative URI, it is resolved relative to the document's base URI, as per [RFC3986], Section 5. However, using relative URIs can cause confusion, and they might not be handled correctly by all implementations.

For example, if the two resources "https://api.example.org/foo/bar/123" and "https://api.example.org/widget/456" both respond with a "type" equal to the relative URI reference "example-problem", when resolved they will identify different resources ("https://api.example.org/foo/bar/example-problem" and "https://api.example.org/widget/example-problem" respectively). As a result, it is RECOMMENDED that absolute URIs be used in "type" when possible, and that when relative URIs are used, they include the full path (e.g., "/types/123").

The type URI can also be a non-resolvable URI. For example, the tag URI scheme [RFC4151] can be used to uniquely identify problem types:

```
tag:mnot@mnot.net,2021-09-17:OutOfLuck
```

Non-resolvable URIs ought not be used when there is some future possibility that it might become desirable to do so. For example, if an API designer used the URI above and later adopted a tool that resolves type URIs to discover information about the error, taking advantage of that capability would require switching to a resolvable URI, creating a new identity for the problem type and thus introducing a breaking change.

3.1.2. "status"

The "status" member is a JSON number indicating the HTTP status code ([HTTP], Section 15) generated by the origin server for this occurrence of the problem.

The "status" member, if present, is only advisory; it conveys the HTTP status code used for the convenience of the consumer. Generators MUST use the same status code in the actual HTTP response, to assure that generic HTTP software that does not understand this format still behaves correctly. See Section 5 for further caveats regarding its use.

Consumers can use the status member to determine what the original status code used by the generator was, in cases where it has been changed (e.g., by an intermediary or cache), and when message bodies persist without HTTP information. Generic HTTP software will still use the HTTP status code.

3.1.3. "title"

The "title" member is a JSON string containing a short, human-readable summary of the problem type.

It SHOULD NOT change from occurrence to occurrence of the problem, except for localization (e.g., using proactive content negotiation; see [HTTP], Section 12.1).

The "title" string is advisory and included only for users who are not aware of the semantics of the URI and can not discover them (e.g., during offline log analysis).

3.1.4. "detail"

The "detail" member is a JSON string containing a human-readable explanation specific to this occurrence of the problem.

The "detail" member, if present, ought to focus on helping the client correct the problem, rather than giving debugging information.

Consumers SHOULD NOT parse the "detail" member for information; extensions are more suitable and less error-prone ways to obtain such information.

3.1.5. "instance"

The "instance" member is a JSON string containing a URI reference that identifies the specific occurrence of the problem.

When the "instance" URI is dereferenceable, the problem details object can be fetched from it. It might also return information about the problem occurrence in other formats through use of proactive content negotiation (see [HTTP], Section 12.5.1).

When the "instance" URI is not dereferenceable, it serves as a unique identifier for the problem occurrence that may be of significance to the server, but is opaque to the client.

When "instance" contains a relative URI, it is resolved relative to the document's base URI, as per [RFC3986], Section 5. However, using relative URIs can cause confusion, and they might not be handled correctly by all implementations.

For example, if the two resources "https://api.example.org/foo/bar/123" and "https://api.example.org/widget/456" both respond with an "instance" equal to the relative URI reference "example-instance", when resolved they will identify different resources ("https://api.example.org/foo/bar/example-instance" and "https://api.example.org/widget/example-instance" respectively). As a result, it is RECOMMENDED that absolute URIs be used in "instance" when possible, and that when relative URIs are used, they include the full path (e.g., "/instances/123").

3.2. Extension Members

Problem type definitions MAY extend the problem details object with additional members.

For example, our "out of credit" problem above defines two such extensions -- "balance" and "accounts" to convey additional, problem-specific information.

Similarly, the "Multi-Status" example defines two extensions -- "causes" and "problem-pointer". Extensions like "problem-pointer" are more appropriate to use for problems associated with client side errors 4xx only.

Clients consuming problem details MUST ignore any such extensions that they don't recognize; this allows problem types to evolve and include additional information in the future.

Note that because extensions are effectively put into a namespace by the problem type, it is not possible to define new "standard" members without defining a new media type.

4. Defining New Problem Types

When an HTTP API needs to define a response that indicates an error condition, it might be appropriate to do so by defining a new problem type.

Before doing so, it's important to understand what they are good for, and what's better left to other mechanisms.

Problem details are not a debugging tool for the underlying implementation; rather, they are a way to expose greater detail about the HTTP interface itself. Designers of new problem types need to carefully consider the Security Considerations (Section 5), in particular, the risk of exposing attack vectors by exposing implementation internals through error messages.

Likewise, truly generic problems -- i.e., conditions that might apply to any resource on the Web -- are usually better expressed as plain status codes. For example, a "write access disallowed" problem is probably unnecessary, since a 403 Forbidden status code in response to a PUT request is self-explanatory.

Finally, an application might have a more appropriate way to carry an error in a format that it already defines. Problem details are intended to avoid the necessity of establishing new "fault" or "error" document formats, not to replace existing domain-specific formats.

That said, it is possible to add support for problem details to existing HTTP APIs using HTTP content negotiation (e.g., using the Accept request header to indicate a preference for this format; see [HTTP], Section 12.5.1).

New problem type definitions MUST document:

1. a type URI (typically, with the "http" or "https" scheme),
2. a title that appropriately describes it (think short), and
3. the HTTP status code for it to be used with.

Problem type definitions MAY specify the use of the Retry-After response header ([HTTP], Section 10.2.3) in appropriate circumstances.

A problem's type URI SHOULD resolve to HTML [HTML5] documentation that explains how to resolve the problem.

A problem type definition MAY specify additional members on the problem details object. For example, an extension might use typed links [RFC8288] to another resource that machines can use to resolve the problem.

If such additional members are defined, their names SHOULD start with a letter (ALPHA, as per [RFC5234], Appendix B.1) and SHOULD comprise characters from ALPHA, DIGIT ([RFC5234], Appendix B.1), and "_" (so that it can be serialized in formats other than JSON), and they SHOULD be three characters or longer.

4.1. Example

For example, if you are publishing an HTTP API to your online shopping cart, you might need to indicate that the user is out of credit (our example from above), and therefore cannot make the purchase.

If you already have an application-specific format that can accommodate this information, it's probably best to do that. However, if you don't, you might use one of the problem details formats -- JSON if your API is JSON-based, or XML if it uses that format.

To do so, you might look in the registry (Section 4.2) for an already-defined type URI that suits your purposes. If one is available, you can reuse that URI.

If one isn't available, you could mint and document a new type URI (which ought to be under your control and stable over time), an appropriate title and the HTTP status code that it will be used with, along with what it means and how it should be handled.

4.2. Registered Problem Types

This specification defines the HTTP Problem Type registry for common, widely-used problem type URIs, to promote reuse.

The policy for this registry is Specification Required, per [RFC8126], Section 4.5.

When evaluating requests, the Expert(s) should consider community feedback, how well-defined the problem type is, and this specification's requirements. Vendor-specific, application-specific, and deployment-specific values are not registrable. Specification documents should be published in a stable, freely available manner (ideally located with a URL), but need not be standards.

Registrations MAY use the prefix "https://iana.org/assignments/http-problem-types#" for the type URI.

Registration requests should use the following template:

- * Type URI: [a URI for the problem type]
- * Title: [a short description of the problem type]
- * Recommended HTTP status code: [what status code is most appropriate to use with the type]
- * Reference: [to a specification defining the type]

See the registry at <https://iana.org/assignments/http-problem-types> (<https://iana.org/assignments/http-problem-types>) for details on where to send registration requests.

4.2.1. about:blank

This specification registers one Problem Type, "about:blank".

- * Type URI: about:blank
- * Title: See HTTP Status Code
- * Recommended HTTP status code: N/A
- * Reference: [this document]

The "about:blank" URI [RFC6694], when used as a problem type, indicates that the problem has no additional semantics beyond that of the HTTP status code.

When "about:blank" is used, the title SHOULD be the same as the recommended HTTP status phrase for that code (e.g., "Not Found" for 404, and so on), although it MAY be localized to suit client preferences (expressed with the Accept-Language request header).

Please note that according to how the "type" member is defined (Section 3.1), the "about:blank" URI is the default value for that member. Consequently, any problem details object not carrying an explicit "type" member implicitly uses this URI.

5. Security Considerations

When defining a new problem type, the information included must be carefully vetted. Likewise, when actually generating a problem -- however it is serialized -- the details given must also be scrutinized.

Risks include leaking information that can be exploited to compromise the system, access to the system, or the privacy of users of the system.

Generators providing links to occurrence information are encouraged to avoid making implementation details such as a stack dump available through the HTTP interface, since this can expose sensitive details of the server implementation, its data, and so on.

The "status" member duplicates the information available in the HTTP status code itself, bringing the possibility of disagreement between the two. Their relative precedence is not clear, since a disagreement might indicate that (for example) an intermediary has changed the HTTP status code in transit (e.g., by a proxy or cache). Generic HTTP software (such as proxies, load balancers, firewalls, and virus scanners) are unlikely to know of or respect the status code conveyed in this member.

6. IANA Considerations

Please update the "application/problem+json" and "application/problem+xml" registrations in the Internet media types registry [RFC6838]. to refer to this document.

Please create the HTTP Problem Types Registry, as specified in Section 4.2, and populate it with "about:blank" as per Section 4.2.1.

7. References

7.1. Normative References

- [HTTP] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>>.
- [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>>.
- [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/rfc/rfc3986>>.

- [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/rfc/rfc5234>>.
- [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>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/rfc/rfc8259>>.
- [XML] 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, <<https://www.w3.org/TR/2008/REC-xml-20081126>>.

7.2. Informative References

- [HTML5] WHATWG, "HTML - Living Standard", n.d., <<https://html.spec.whatwg.org>>.
- [I-D.draft-bhutton-json-schema-00] Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents", Work in Progress, Internet-Draft, draft-bhutton-json-schema-00, 8 December 2020, <<https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-00>>.
- [ISO-19757-2] International Organization for Standardization, "Information Technology -- Document Schema Definition Languages (DSDL) -- Part 2: Grammar-based Validation -- RELAX NG", ISO/IEC 19757-2, 2003.
- [RDFa] Adida, B., Birbeck, M., McCarron, S., and I. Herman, "RDFa Core 1.1 - Third Edition", World Wide Web Consortium Recommendation REC-rdfa-core-20150317, 17 March 2015, <<https://www.w3.org/TR/2015/REC-rdfa-core-20150317>>.

- [RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, DOI 10.17487/RFC4151, October 2005, <<https://www.rfc-editor.org/rfc/rfc4151>>.
- [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/rfc/rfc4918>>.
- [RFC6694] Moonesamy, S., Ed., "The "about" URI Scheme", RFC 6694, DOI 10.17487/RFC6694, August 2012, <<https://www.rfc-editor.org/rfc/rfc6694>>.
- [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/rfc/rfc6838>>.
- [RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <<https://www.rfc-editor.org/rfc/rfc6901>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/rfc/rfc8288>>.
- [XSLT] Clark, J., Pieters, S., and H. Thompson, "Associating Style Sheets with XML documents 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xml-stylesheet-20101028, 28 October 2010, <<https://www.w3.org/TR/2010/REC-xml-stylesheet-20101028>>.

Appendix A. JSON Schema for HTTP Problems

This section presents a non-normative JSON Schema [I-D.draft-bhutton-json-schema-00] for HTTP Problem Details. If there is any disagreement between it and the text of the specification, the latter prevails.

```
# NOTE: '\ ' line wrapping per RFC 8792
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "A problem object RFC 7807bis",
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "format": "uri-reference",
      "description": "A URI reference RFC3986 that identifies the \
problem type."
    },
    "title": {
      "type": "string",
      "description": "A short, human-readable summary of the \
problem type. It SHOULD NOT change from occurrence to occurrence \
of the problem, except for purposes of localization (e.g., using \
proactive content negotiation; see RFC7231, Section 3.4)"
    },
    "status": {
      "type": "integer",
      "description": "The HTTP status code (RFC7231, Section 6) \
generated by the origin server for this occurrence of the problem.",
      "minimum": 100,
      "maximum": 599
    },
    "detail": {
      "type": "string",
      "description": "A human-readable explanation specific to \
this occurrence of the problem."
    },
    "instance": {
      "type": "string",
      "format": "uri-reference",
      "description": "A URI reference that identifies the \
specific occurrence of the problem. It may or may not yield \
further information if dereferenced."
    }
  }
}
```

Appendix B. HTTP Problems and XML

HTTP-based APIs that use XML [XML] can express problem details using the format defined in this appendix.

The RELAX NG schema [ISO-19757-2] for the XML format is:

```

default namespace ns = "urn:ietf:rfc:7807"

start = problem

problem =
  element problem {
    ( element type           { xsd:anyURI }?
      & element title        { xsd:string }?
      & element detail        { xsd:string }?
      & element status        { xsd:positiveInteger }?
      & element instance      { xsd:anyURI }? ),
    anyNsElement
  }

anyNsElement =
  ( element ns:* { anyNsElement | text }
    | attribute * { text } ) *

```

Note that this schema is only intended as documentation, and not as a normative schema that captures all constraints of the XML format. It is possible to use other XML schema languages to define a similar set of constraints (depending on the features of the chosen schema language).

The media type for this format is "application/problem+xml".

Extension arrays and objects are serialized into the XML format by considering an element containing a child or children to represent an object, except for elements that contain only child element(s) named 'i', which are considered arrays. For example, the example above appears in XML as follows:

```

HTTP/1.1 403 Forbidden
Content-Type: application/problem+xml
Content-Language: en

```

```

<?xml version="1.0" encoding="UTF-8"?>
<problem xmlns="urn:ietf:rfc:7807">
  <type>https://example.com/probs/out-of-credit</type>
  <title>You do not have enough credit.</title>
  <detail>Your current balance is 30, but that costs 50.</detail>
  <instance>https://example.net/account/12345/msgs/abc</instance>
  <balance>30</balance>
  <accounts>
    <i>https://example.net/account/12345</i>
    <i>https://example.net/account/67890</i>
  </accounts>
</problem>

```

This format uses an XML namespace, primarily to allow embedding it into other XML-based formats; it does not imply that it can or should be extended with elements or attributes in other namespaces. The RELAX NG schema explicitly only allows elements from the one namespace used in the XML format. Any extension arrays and objects MUST be serialized into XML markup using only that namespace.

When using the XML format, it is possible to embed an XML processing instruction in the XML that instructs clients to transform the XML, using the referenced XSLT code [XSLT]. If this code is transforming the XML into (X)HTML, then it is possible to serve the XML format, and yet have clients capable of performing the transformation display human-friendly (X)HTML that is rendered and displayed at the client. Note that when using this method, it is advisable to use XSLT 1.0 in order to maximize the number of clients capable of executing the XSLT code.

Appendix C. Using Problem Details with Other Formats

In some situations, it can be advantageous to embed problem details in formats other than those described here. For example, an API that uses HTML [HTML5] might want to also use HTML for expressing its problem details.

Problem details can be embedded in other formats either by encapsulating one of the existing serializations (JSON or XML) into that format or by translating the model of a problem detail (as specified in Section 3) into the format's conventions.

For example, in HTML, a problem could be embedded by encapsulating JSON in a script tag:

```
<script type="application/problem+json">
  {
    "type": "https://example.com/probs/out-of-credit",
    "title": "You do not have enough credit.",
    "detail": "Your current balance is 30, but that costs 50.",
    "instance": "/account/12345/msgs/abc",
    "balance": 30,
    "accounts": ["/account/12345",
                  "/account/67890"]
  }
</script>
```

or by inventing a mapping into RDFa [RDFa].

This specification does not make specific recommendations regarding embedding problem details in other formats; the appropriate way to embed them depends both upon the format in use and application of that format.

Acknowledgements

The authors would like to thank Jan Algermissen, Subbu Allamaraju, Mike Amundsen, Roy Fielding, Eran Hammer, Sam Johnston, Mike McCall, Julian Reschke, and James Snell for review of this specification.

Authors' Addresses

Mark Nottingham
Pahran
Australia
Email: mnot@mnot.net
URI: <https://www.mnot.net/>

Erik Wilde
Email: erik.wilde@dret.net
URI: <http://dret.net/netdret/>

Sanjay Dalal
United States of America
Email: sanjay.dalal@cal.berkeley.edu
URI: <https://github.com/sdatspun2>