

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 28, 2021

A. Cedik
shipcloud GmbH
E. Wilde
Axway
September 24, 2020

Communicating Warning Information in HTTP APIs
draft-cedik-http-warning-02

Abstract

This document defines a new HTTP field Content-Warning and a standard response format for representing warning information in HTTP APIs.

Note to Readers

This draft should be discussed on the rfc-interest mailing list
(<https://lists.w3.org/Archives/Public/ietf-http-wg/>).

Online access to all versions and files is available on GitHub
(<https://github.com/dret/I-D/tree/master/http-warning>).

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 March 28, 2021.

Copyright Notice

Copyright (c) 2020 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
2. Notational Conventions	3
3. Content-Warning Field	3
3.1. HTTP request methods	4
4. The "embedded-warning" Content-Warning Type	4
4.1. Allowed HTTP request methods for embedded-warning	4
5. JSON Warning Format	5
6. Example with HTTP Field and Embedded Warning	5
7. Cache Considerations	6
7.1. Caching the "embedded-warning" Content-Warning type	7
8. Security Considerations	7
8.1. Absence of a response body	7
8.2. Absence of warnings in the response body	8
9. IANA Considerations	8
9.1. HTTP Field Content-Warning	8
9.2. Content-Warning Type Registry	8
9.2.1. Registration Procedure	8
9.2.2. Initial Registry Content	9
10. References	9
10.1. Normative References	9
10.2. Informative References	10
Appendix A. Acknowledgements	10
Authors' Addresses	10

1. Introduction

Many current APIs are based on HTTP [RFC7230] as their application protocol. Their response handling model is based on the assumption that requests either are successful or they fail. In both cases (success and failure) an HTTP status code [RFC7231] is returned to convey either fact.

But response status is not always strictly either success or failure. For example, there are cases where an underlying system returns a response with data that cannot be defined as a clear error. API providers who are integrating such a service might want to return a success response nonetheless, but returning a HTTP status code of e.g. 200 OK without any additional information is not the only possible approach in this case.

As defined in the principles of Web architecture [W3C.REC-webarch-20041215], agents that "recover from errors by making a choice without the user's consent are not acting on the user's behalf". Therefore APIs should be able to communicate what has happened to their consumers, which then allows clients or users to make more informed decisions. Note that this specification specifically targets warnings and not errors, meaning that while it may be useful for clients to understand the warning condition and act on it, they also may choose to ignore it and treat the response as a successful one.

This document defines a warning code and a standard response structure for communicating and representing warning information in HTTP APIs. The goal is to allow HTTP providers to have a standardized way of communicating to their consumers that while the response can be considered to represent success, there is warning information available that they might want to take into account.

As a general guideline, warning information should be considered to be any information that can be safely ignored (treating the response as if it did not communicate or embed any warning information), but that might help clients and users to make better decisions.

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Content-Warning Field

The Content-Warning field can be found in the header or trailer section (see Section 4.6 of [I-D.ietf-httpbis-semantics]) of http responses and allows to represent different kinds of warning information via HTTP. It is defined as a Structured Header List [I-D.ietf-httpbis-header-structure]. Its ABNF is:

Content-Warning = sh-list

Each member of the list MUST have exactly the two parameters "type" and "date".

- o The "type" parameter represents the warning that is being signaled. Its value is defined as a sh-token and SHOULD be a type that is registered in the Content-Warning type registry Section 9.2. Clients SHOULD ignore Content-Warning types that they do not know.

- o The "date" parameter defines the last occurrence of this warning as a structured headers date as defined in [I-D.ietf-binary-structured-headers] (e.g. "1581410465").

Intermediaries of a response are not allowed to modify existing Content-Warning fields, but can add additional entries if warnings are produced while they are handling a response.

3.1. HTTP request methods

The Content-Warning Field is not tied to any specific HTTP request method, although specific values MAY only be used with a single or a subset of methods. The information as to which HTTP request methods are support for a single Content-Warning Type MUST be defined in the definition of the Content-Warning Type.

4. The "embedded-warning" Content-Warning Type

This document introduces the Content-Warning Type "embedded-warning".

As mentioned in the introduction (Section 1), HTTP requests can be successful or they can fail. They can also result in a state where the original intent was satisfied, but a side effect happened that should be conveyed back to the client.

To make it easier for clients to handle such an event, the Content-Warning type "embedded-warning" MAY be returned. In this case, the client MAY either treat the response according to its HTTP status code, or in addition the client MAY use the embedded warning information to understand the nature of the warning.

The "embedded-warning" type does not prescribe the way in which warnings are represented. The assumption is that the response will have embedded information that allows the client to learn about the nature of the warning. The following section describes a JSON structure that MAY be used to represent the warning. HTTP services are free to use this or other formats to represent the warning information they are embedding.

An exemplary Content-Warning field looks like this:

Content-Warning: "embedded-warning"; 1590190500

4.1. Allowed HTTP request methods for embedded-warning

The embedded-warning Content-Warning Type infers, that there is more information in the responses body. Therefore all HTTP request

methods that MAY have content in their body MAY also return embedded warnings.

HTTP request methods that do not return a body in their response SHOULD NOT return the embedded-warning Content-Warning Type.

The HTTP request method HEAD is an exception since it is allowed to return headers that are meant for being returned when sending a GET request. Therefore it MAY return the embedded-warning Content-Warning Type, although the body will be empty.

5. JSON Warning Format

The JSON warning format uses the JSON format described in [RFC8259]. It is intended to be used as a building block in the response schemas of JSON-based APIs.

In many current designs of JSON-based HTTP APIs, services represent response data as members of the returned JSON object. In order to make it easier for consumers to identify information about warnings, a top-level member is defined that contains all warning information in a representation. A "warnings" member MUST encapsulate the warnings that will be returned to the client.

When a condition occurs that can not be defined as a "hard error" (i.e., that allows clients to continue treating the resulting response as a success), additional information about this condition SHOULD be returned to the client. The "warnings" member MUST be an array that is structured with one object for each and every warning message that is returned to the client.

Entries in these individual objects follow the pattern described in [RFC7807].

When warnings are present the Content-Warning field (as defined in Section 3) SHOULD be set to indicate that warnings have been returned. This way a client will not have to parse the response body to find out whether a warnings member is present.

6. Example with HTTP Field and Embedded Warning

Since warnings do not have an effect on the returned HTTP status code, the response status code SHOULD be in the 2xx range, indicating that the intent of the client was successful.

```
POST /example HTTP/1.1
Host: example.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json
Content-Warning: "embedded-warning"; 1590190500

{
  "request_id": "2326b087-d64e-43bd-a557-42171155084f",
  "warnings": [
    {
      "detail": "Street name was too long. It has been shortened...",
      "instance": "https://example.com/shipments/3a186c51/msgs/c94d",
      "status": "200",
      "title": "Street name too long. It has been shortened.",
      "type": "https://example.com/errors/shortened_entry"
    },
    {
      "detail": "City for this zipcode unknown. Code for shipment..",
      "instance": "https://example.com/shipments/3a186c51/msgs/5927",
      "status": "200",
      "title": "City for zipcode unknown.",
      "type": "https://example.com/errors/city_unknown"
    }
  ],
  "id": "3a186c51d4281acb",
  "carrier_tracking_no": "84168117830018",
  "tracking_url": "http://example.com/3a186c51d",
  "label_url": "http://example.com/shipping_label_3a186c51d.pdf",
  "price": 3.4
}
```

This example shows that the original intent was successful. If the original request was in fact not successful, a different status code SHOULD be returned. Embedded warnings are not tied to a specific http status code. Therefore they can be combined with every status code.

7. Cache Considerations

The Content-Warning field itself does not encourage a specific handling when it comes to caching responses. It is up to the Content-Warning type to specify if caching can be used or not.

7.1. Caching the "embedded-warning" Content-Warning type

The reasons for returning the Content-Warning Type "embedded-warning" can be manifold. A system could e.g. return warnings due to circumstances in the backend that can either still exist on subsequent requests or that have been solved in the meantime.

Intermediaries can fall into the same category. When a warning occurs, it can add warnings to the response making it possible to debug what happened at the intermediary. The reason for said warning can persist or may disappear on subsequent requests.

Therefore caching embedded-warnings SHOULD NOT be done. As one can't predict if the reason for returning embedded-warnings is still persistent.

8. Security Considerations

API providers need to exercise care when reporting warnings. Malicious actors could use this information for orchestrating attacks. Social engineering can also be a factor when warning information is returned by the API.

Clients processing warning information SHOULD make sure the right type of content was transmitted by checking the content-type header as well as the content-warning field. Content in the body's warnings object SHOULD be processed accordingly. If no content-warning field was provided, clients are advised to ignore the content provided in the body's warnings object.

8.1. Absence of a response body

As described in Section 4.1 the embedded-warning Content-Warning type is expecting a body to be returned in the http response unless the HEAD method has been used for the request.

Therefore API clients SHOULD only parse a response's body when the Content-Warning type is "embedded-warning". When the body is absent, a client SHOULD stop processing the response and return an adequate error message.

If an intermediary discovers a missing response body it MAY adjust the response to return a http status code of 500 - internal server error (see Section 6.6.1 of [RFC7231]).

8.2. Absence of warnings in the response body

When the response body does not contain warnings a client MAY use appropriate ways to inform the api provider about the fact. An error message MAY be

If an intermediary discovers missing warnings in the response body it MAY adjust the response to return warnings containing this information.

9. IANA Considerations

9.1. HTTP Field Content-Warning

This specification registers the following entry in the Permanent Message Field Names registry established by [RFC3864]:

- o Field name: Content-Warning
- o Applicable protocol: HTTP
- o Status: standard
- o Author/Change Controller: IETF
- o Specification document(s): [this document]
- o Related information:

9.2. Content-Warning Type Registry

The "Content-Warning Type Registry" defines the namespace for new Content-Warning types. This specification establishes a new registry according to the guidelines given in [RFC8126]. This new registry should not be included in an existing group of registries.

9.2.1. Registration Procedure

A registration MUST include the following fields:

- o Content-Warning Type: Name of the Content-Warning Type
- o Reference: Pointer to a specification text

The registration policy for this registry is "Specification Required" as defined by [RFC8126], Section 4.6. They MUST follow the "sh-token" syntax defined by [I-D.ietf-httpbis-header-structure].

9.2.2. Initial Registry Content

The registry has been populated with the registered values shown below:

Content-Warning Type	Reference
embedded-warning	this RFC, Section 4

10. References

10.1. Normative References

- [I-D.ietf-binary-structured-headers]
 Nottingham, M., "Binary Structured HTTP Headers", draft-nottingham-binary-structured-headers-02 (work in progress), March 2020.
- [I-D.ietf-httpbis-header-structure]
 Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-14 (work in progress), October 2019.
- [I-D.ietf-httpbis-semantic]
 Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", draft-ietf-httpbis-semantic-07 (work in progress), March 2020.
- [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>>.
- [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>>.
- [RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <<https://www.rfc-editor.org/info/rfc7807>>.
- [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/info/rfc8126>>.
- [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>>.

10.2. Informative References

- [W3C.REC-webarch-20041215]
Jacobs, I. and N. Walsh, "Architecture of the World Wide Web, Volume One", World Wide Web Consortium Recommendation REC-webarch-20041215, December 2004, <<http://www.w3.org/TR/2004/REC-webarch-20041215>>.

Appendix A. Acknowledgements

Thanks for comments and suggestions provided by Roy Fielding, Mark Nottingham, and Roberto Polli.

Authors' Addresses

Andre Cedik
shipcloud GmbH

Email: andre.cedik@gmail.com

Internet-DraftCommunicating Warning Information in HTTP APSeptember 2020

Erik Wilde
Axway

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2020

S. Dalal
E. Wilde
June 13, 2020

The Deprecation HTTP Header Field
draft-dalal-deprecation-header-03

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 December 15, 2020.

Copyright Notice

Copyright (c) 2020 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 Field	3
2.1. Syntax	3
3. The Deprecation Link Relation Type	4
3.1. Documentation	4
4. Recommend Replacement	5
5. Sunset	6
6. Resource Behavior	6
7. IANA Considerations	6
7.1. The Deprecation HTTP Response Header Field	6
7.2. The Deprecation Link Relation Type	7
8. Implementation Status	7
8.1. Implementing the Deprecation Header Field	8
8.2. Implementing the Concept	9
9. Security Considerations	10
10. Examples	11
11. References	11
11.1. Normative References	11
11.2. Informative References	12
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 Field

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 Friday, 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

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

"Link" [RFC8288] header could be used in addition to the "Deprecation" header to recommend the client application about available alternates to the deprecated resource. Following relation types as defined in [RFC8288] are RECOMMENDED to use for the 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 Field

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 Field"

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 Field

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>>.

Appendix A. Acknowledgments

The authors would like to thank Nikhil Kolekar, 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/sdatspun2>

Erik Wilde

Email: erik.wilde@dret.net

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

HTTP
Internet-Draft
Intended status: Standards Track
Expires: 30 May 2021

R. Polli
Team Digitale, Italian Government
A. Martinez
Red Hat
26 November 2020

RateLimit Header Fields for HTTP
draft-polli-ratelimit-headers-05

Abstract

This document defines the RateLimit-Limit, RateLimit-Remaining, RateLimit-Reset fields for HTTP, thus allowing servers to publish current request quotas 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 (ietf-http-wg@w3.org), which is archived at <https://lists.w3.org/Archives/Public/ietf-http-wg/> (<https://lists.w3.org/Archives/Public/ietf-http-wg/>).

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

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 30 May 2021.

Copyright Notice

Copyright (c) 2020 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
1.1. Rate-limiting and quotas	3
1.2. Current landscape of rate-limiting headers	4
1.2.1. Interoperability issues	4
1.3. This proposal	5
1.4. Goals	5
1.5. Notational Conventions	6
2. Expressing rate-limit policies	6
2.1. Time window	6
2.2. Request quota	6
2.3. Quota policy	7
3. Header Specifications	8
3.1. RateLimit-Limit	8
3.2. RateLimit-Remaining	9
3.3. RateLimit-Reset	9
4. Providing RateLimit headers	10
5. Intermediaries	11
6. Caching	11
7. Receiving RateLimit headers	11
8. Examples	12
8.1. Unparameterized responses	12
8.1.1. Throttling informations in responses	12
8.1.2. Use in conjunction with custom headers	13
8.1.3. Use for limiting concurrency	13
8.1.4. Use in throttled responses	14
8.2. Parameterized responses	15
8.2.1. Throttling window specified via parameter	15
8.2.2. Dynamic limits with parameterized windows	15
8.2.3. Dynamic limits for pushing back and slowing down	16
8.3. Dynamic limits for pushing back with Retry-After and slow down	17
8.3.1. Missing Remaining informations	17

8.3.2. Use with multiple windows	18
9. Security Considerations	19
9.1. Throttling does not prevent clients from issuing requests	19
9.2. Information disclosure	19
9.3. Remaining quota-units are not granted requests	19
9.4. Reliability of RateLimit-Reset	20
9.5. Resource exhaustion	20
9.6. Denial of Service	20
10. IANA Considerations	20
10.1. RateLimit-Limit Field Registration	21
10.2. RateLimit-Remaining Field Registration	21
10.3. RateLimit-Reset Field Registration	21
11. References	21
11.1. Normative References	21
11.2. Informative References	22
Appendix A. Change Log	23
Appendix B. Acknowledgements	23
Appendix C. RateLimit headers currently used on the web	23
Appendix D. FAQ	24
Authors' Addresses	27

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 with the "Retry-After" header field defined in [SEMANTICS] to be returned in "429 Too Many Requests" or "503 Service Unavailable" responses.

Still, there is not a standard way to communicate service quotas so that the client can throttle its requests and prevent 4xx or 5xx responses.

1.1. 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 - eg. monetization.

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

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

Quotas may be enforced on different basis (eg. 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 request per minute;
- * limited to 1000 request per hour.

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

1.2. Current landscape of rate-limiting headers

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;

1.2.1. 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.

Client applications 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.

1.3. This proposal

This proposal 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 "delta-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.

1.4. Goals

The goals of this proposal are:

1. Standardizing the names and semantic of rate-limit headers;
2. Improve resiliency of HTTP infrastructures simplifying the enforcement and the adoption of rate-limit headers;
3. Simplify API documentation avoiding expliciting rate-limit fields semantic in documentation.

The goals do not include:

Authorization: The rate-limit headers described here 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 [RFC6454] section 7.

Response status code: The rate-limit headers may be returned in both

Successful and non Successful responses. This specification does not cover whether non Successful responses count on quota usage.

Throttling policy: This specification does not mandate a specific throttling policy. The values published in the headers, 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.

1.5. 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 7 of [MESSAGING].

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

The "delta-seconds" rule is defined in [CACHING] section 1.2.1.

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 = delta-seconds

Subsecond precision is not supported.

2.2. Request quota

The request-quota 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 "request-quota" is expressed in "quota-units" and has the following syntax:

```
request-quota = quota-units
quota-units = 1*DIGIT
```

The "request-quota" SHOULD match the maximum number of acceptable requests.

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

If the "request-quota" 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=Camilleri"

so that we have the following counters

```
GET /books/123                ; request-quota=4, remaining: 3, status=200
GET /books?author=Camilleri    ; request-quota=4, remaining: 1, status=200
GET /books?author=Eco         ; request-quota=4, remaining: 0, status=429
```

2.3. Quota policy

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

```
quota-policy = request-quota; "w" "=" time-window
              *( OWS ";" OWS quota-comment)
quota-comment = token "=" (token / quoted-string)
```

quota-policy parameters like "w" and quota-comment tokens MUST NOT occur multiple times within the same quota-policy.

An example policy of 100 quota-units per minute.

```
100;w=60
```

Two examples of providing further details via custom parameters in "quota-comments".

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

3. Header Specifications

The following "RateLimit" response fields are defined

3.1. RateLimit-Limit

The "RateLimit-Limit" response field indicates the "request-quota" associated to the client in the current "time-window".

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

The header value is

```
RateLimit-Limit = expiring-limit [ , 1#quota-policy ]  
expiring-limit = request-quota
```

The "expiring-limit" value MUST be set to the "request-quota" 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 (eg. 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 header MUST NOT occur multiple times and can be sent in a trailer section.

3.2. RateLimit-Remaining

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

The header value is

RateLimit-Remaining = quota-units

This header 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 of being served.

A low "RateLimit-Remaining" value is like a yellow traffic-light: the red light may arrive suddenly.

One example of "RateLimit-Remaining" use is below.

RateLimit-Remaining: 50

3.3. RateLimit-Reset

The "RateLimit-Reset" response field indicates either

- * the number of seconds until the quota resets.

The header value is

RateLimit-Reset = delta-seconds

The delta-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 [SEMANTICS] Section 4.1.1.1);
- * it mitigates the risk related to thundering herd when too many clients are serviced with the same timestamp.

This header 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 "request-quota" will be restored after the moment referenced by "RateLimit-Reset". The server MAY arbitrarily alter the "RateLimit-Reset" value between subsequent requests eg. in case of resource saturation or to implement sliding window policies.

4. Providing RateLimit headers

A server MAY use one or more "RateLimit" response fields defined in this document to communicate its quota policies.

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.

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" headers related to the window with the lower "RateLimit-Remaining" values.

Under certain conditions, a server MAY artificially lower "RateLimit" field values between subsequent requests, eg. 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.

5. Intermediaries

This section documents the considerations advised in Section 15.3.3 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.

6. 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 "window" value of a "quota-policy".

7. Receiving RateLimit headers

A client MUST process the received "RateLimit" headers.

A client MUST validate the values received in the "RateLimit" headers 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 "request-quota" too high.

Malformed "RateLimit" headers 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 informations (eg. the "Date" response header, 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, the "Retry-After" header field MUST take precedence and the "RateLimit-Reset" field MAY be ignored.

8. Examples

8.1. Unparameterized responses

8.1.1. Throttling informations in responses

The client exhausted its request-quota for the next 50 seconds. The "time-window" is communicated out-of-band or inferred by the header values.

Request:

```
GET /items/123
```

Response:

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

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

8.1.2. Use in conjunction with custom headers

The server uses two custom headers, 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-*" headers to inform the client that:

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

Request:

```
GET /items/123
```

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"}
```

8.1.3. Use for limiting concurrency

Throttling headers 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

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

Response:

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

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

8.1.4. Use in throttled responses

A client exhausted its quota and the server throttles the request sending the "Retry-After" response header field.

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

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"
}
```

8.2. Parameterized responses

8.2.1. 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
```

Response:

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

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

8.2.2. 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
```

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."
}
```

8.2.3. 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
```

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!"
}
```


8.3. 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 the Retry-After header, with the advantage that the server can now specify the policy's nominal limit and window that will apply after the reset, ie. 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
```

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 the "Retry-After" header 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.

8.3.1. Missing Remaining informations

The server does not expose "RateLimit-Remaining" values, but 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
```

Response:

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

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

Request:

```
GET /items/123
```

Response:

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

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

8.3.2. Use with multiple windows

This is a standardized way of describing the policy detailed in Section 8.1.2:

- * 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" headers 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
```

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"}
```

9. Security Considerations

9.1. 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.

9.2. Information disclosure

Servers should not disclose operational capacity informations that can be used to saturate its 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 informations of another user.

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

9.3. Remaining quota-units are not granted requests

"RateLimit-*" headers 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 anyway.

9.4. Reliability of RateLimit-Reset

Consider that "request-quota" 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 an higher "RateLimit-Reset" value to limit concurrency or implement dynamic or adaptive throttling policies.

9.5. 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 adding some jitter to the field-value.

9.6. 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.

10. IANA Considerations

10.1. RateLimit-Limit Field Registration

This section registers the "RateLimit-Limit" field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]).

Field name: "RateLimit-Limit"

Status: permanent

Specification document(s): Section 3.1 of this document

10.2. RateLimit-Remaining Field Registration

This section registers the "RateLimit-Remaining" field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]).

Field name: "RateLimit-Remaining"

Status: permanent

Specification document(s): Section 3.2 of this document

10.3. RateLimit-Reset Field Registration

This section registers the "RateLimit-Reset" field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([SEMANTICS]).

Field name: "RateLimit-Reset"

Status: permanent

Specification document(s): Section 3.3 of this document

11. References

11.1. Normative References

[CACHING] 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>>.

[MESSAGING] 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>>.

- [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>>.
- [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>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <<https://www.rfc-editor.org/info/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/info/rfc8174>>.
- [SEMANTICS] 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>>.
- [UNIX] The Open Group, ., "The Single UNIX Specification, Version 2 - 6 Vol Set for UNIX 98", February 1997.

11.2. Informative References

- [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/info/rfc3339>>.
- [RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <<https://www.rfc-editor.org/info/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/info/rfc7234>>.

Appendix A. Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

Appendix B. 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 implementors: Aapo Talvensaari, Nathan Friedly and Sanyam Dogra.

Appendix C. RateLimit headers currently used on the web

RFC EDITOR PLEASE DELETE THIS SECTION.

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
 - 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. Eg.

```
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.

Appendix D. FAQ

1. Why defining standard headers 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.

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

After a discussion on a similar thread (<https://github.com/httpwg/http-core/pull/317#issuecomment-585868767>) we will probably add a new "RateLimit-Scope" header to this spec.

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

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

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

delta-seconds as defined in [CACHING] section 1.2.1 clarifies some parsing rules too.

Timestamps require a clock synchronization protocol (see [SEMANTICS] section 4.1.1.1). This may be problematic (eg. 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-http-wg/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 (eg. 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 (eg. 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. The returned values were used to decide whether to serve or not the current request and do not imply any guarantee that future requests will be successful.

Instead they help to understand when future requests will probably be throttled. A low value for "RateLimit-Remaining" should be interpreted as a yellow traffic-light for either the number of requests issued in the "time-window" or the request throughput.

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

You can always return the simplest form of the 3 headers

```
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 header:

```
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 header?

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 headers is easily parseable, even by implementors processing responses using technologies like dynamic interpreter with limited syntax.

Using a single header 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 9.2) 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 9.1). RateLimit headers enable sending on the same connection different limit values on each response, depending on the policy scope (eg. 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.

Authors' Addresses

Roberto Polli
Team Digitale, Italian Government

Email: robipolli@gmail.com

Alejandro Martinez Ruiz
Red Hat

Email: amr@redhat.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 19, 2021

E. Wilde
Axway
H. Van de Sompel
Data Archiving and Networked Services
October 16, 2020

Linkset: Media Types and a Link Relation Type for Link Sets
draft-wilde-linkset-07

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 ART mailing list
(<https://www.ietf.org/mailman/listinfo/art>).

Online access to all versions and files is available on GitHub
(<https://github.com/dret/I-D/tree/master/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 April 19, 2021.

Copyright Notice

Copyright (c) 2020 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	9
5. The "linkset" Relation Type for Linking to a Set of Links . .	13
6. Examples	14
6.1. Set of Links Provided as application/linkset	14
6.2. Set of Links Provided as application/linkset+json	15
6.3. Discovering a Link Set via the "linkset" Link Relation Type	18
7. Implementation Status	18
7.1. GS1	19
7.2. Open Journal Systems (OJS)	19
8. IANA Considerations	19
8.1. Link Relation Type: linkset	19
8.2. Media Type: application/linkset	20
8.2.1. IANA Considerations	20
8.3. Media Type: application/linkset+json	21
9. Security Considerations	22
10. References	23
10.1. Normative References	23
10.2. Informative References	24

Appendix A. Acknowledgements	24
Appendix B. JSON-LD Context	24
Authors' Addresses	29

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 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.

Therefor, 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]. These terms respectively correspond with "Context IRI" and "Target IRI" as used in [RFC5988]. Although defined as IRIs, in common scenarios they are also URIs.

In the examples provided in this document, links in the HTTP "Link" header are shown on separate lines in order to improve readability. Note, however, that as per Section 3.2 of [RFC7230], line breaks are not allowed in values for HTTP headers; 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" header 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" header 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:

- o Providing links in which the resource is involved not just as link context but also as link target.
- o Providing links pertaining to the resource that the server hosting that resource is not aware of.
- o 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:

- o Links are provided in a document that uses a well-defined media type.
- o 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 headers. 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 header 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 headers by specifying an upper bound for their size. But when the number of links is unpredictable, estimating a reliable upper bound is challenging.

HTTP [RFC7231] 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 header 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":

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

Note that [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 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:

- o For every link provided in the set of links, explicitly provide the link context using the "anchor" attribute.
- o For link context ("anchor" attribute) and link target ("href" attribute), use absolute URIs (as defined in Section 4.3 of [RFC3986]).

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

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 [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:

- o For every link provided in the set of links, explicitly provide the link context using the "anchor" member.
- o For link context ("anchor" member) and link target ("href" member), use absolute URIs (as defined in Section 4.3 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 B 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:

- o A set of links MUST be represented as a JSON object which MUST have "linkset" as its sole member.
- o 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.
- o If necessary, the "linkset" member MAY contain information in addition to link context objects, in which case that information MUST NOT change the semantics of the links provided by those link context objects.
- o 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:

- o Each link context object MUST have an "anchor" member with a value that represents the link context. This value SHOULD be an absolute URI as defined in Section 4.3 of [RFC3986]. Cases whereby no value is to be provided for the "anchor" member (i.e. the resource providing the set of links is the link context for each link in the link context object) MUST be handled by providing an "anchor" member with null value ("anchor": null).
- o 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:

- o
 - * For registered relation types [RFC8288], the name of this member is the registered name of the relation type.
 - * For extension relation types [RFC8288], the name of this member is the URI that uniquely represents the relation type.
- o 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:

- o Each link target object MUST have an "href" member with a value that represents the link target. This value SHOULD be an absolute URI as defined in Section 4.3 of [RFC3986]. Cases whereby no value is to be provided for the "href" member (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 null value ("href": "").
- o 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" }
      ]
    }
  ]
}
```

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"}
      ]
    }
  ]
}
```

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"}
      ]
    }
  ]
}
```

4.2.4. Link Target Attributes

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

- o Attributes defined by the serialization of Web Linking [RFC8288].
- o Extension attributes defined and used by communities as allowed by [RFC8288].

- o Internationalized versions of the "title" attribute defined by [RFC8288] and of extension attributes allowed by [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

RFC 8288 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:

- o "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.
- o "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.
- o "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.
- o "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.
- o "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" ]
        }
      ]
    }
  ]
}
```

4.2.4.2. Internationalized Target Attributes

In addition to the target attributes described in Section 4.2.4.1, [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:

- o An internationalized target attribute is represented as a member of the link context object with the same name (including the *) of the attribute.
- o 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.
- o The value of the internationalized target attribute is an array that contains one or more JSON objects. The name of the first 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 the optional second 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 [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": "nachstes Kapitel" , "language" : "de" }
        ]
      ]
    }
  ]
}
```

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. This is not shown in the above example due to the limitations of RFC publication. 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 RFC 8288 (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:

- o 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.

- o The value of an extension attribute MUST be represented by an array, even if there only is one value to be represented.
- o 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.
- o 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" } ]
        }
      ]
    }
  ]
}
```

5. 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 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 not expressed as absolute URIs MUST properly determine what the context is for these links; it SHOULD ignore links for which it is unable to unambiguously make that determination.

6. Examples

Section 6.1 and Section 6.2 show examples whereby the set of links are provided as "application/linkset" and "application/linkset+json" documents, respectively. Section 6.3 illustrates the use of the "linkset" link relation type to support discovery of sets of links.

6.1. Set of Links Provided as application/linkset

Figure 1 shows a client issuing an HTTP GET request against resource <http://example.org/resource1>.

```
GET /resource1 HTTP/1.1
Host: example.org
Connection: close
```

Figure 1: Client HTTP GET request

Figure 2 shows the response to the GET request of Figure 1. The response contains a Content-Type header specifying that the media type of the response is "application/linkset". A set of links, including links that pertain to the responding resource, is provided in the response body.

```
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2019 10:35:51 GMT
Server: Apache-Coyote/1.1
Content-Length: 729
Content-Type: application/linkset
Connection: close

<http://authors.example.net/johndoe>
  ; rel="author"
  ; type="application/rdf+xml"
  ; anchor="http://example.org/resource1",
<http://authors.example.net/janedoe>
  ; rel="author"
  ; type="application/rdf+xml"
  ; anchor="http://example.org/resource1",
<http://example.org/resource1/items/AF48EF.pdf>
  ; rel="item"
  ; type="application/pdf"
  ; anchor="http://example.org/resource1",
<http://example.org/resource1/items/CB63DA.html>
  ; rel="item"
  ; type="text/html"
  ; anchor="http://example.org/resource1",
<http://example.org/resource1>
  ; rel="latest-version"
  ; anchor="http://example.org/resource41/",
<http://example.org/resource40>
  ; rel="prev"
  ; anchor="http://example.org/resource41/"
```

Figure 2: Response to HTTP GET includes a set of links

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

Figure 3 shows the client issuing an HTTP GET request against `<http://example.com/links/article/7507>`. In the request, the client uses an "Accept" header to indicate it prefers a response in the "application/linkset+json" format.

```
GET links/article/7507 HTTP/1.1
Host: example.com
Accept: application/linkset+json
Connection: close
```

Figure 3: Client HTTP GET request expressing preference for "application/linkset+json" response

Figure 4 shows the response to the HTTP GET request of Figure 3. 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
Content-Length: 802

{
  "linkset": [
    {
      "anchor": "https://example.org/article/view/7507",
      "author": [
        {
          "href": "https://orcid.org/0000-0002-1825-0097",
        }
      ],
      "item": [
        {
          "href": "https://example.org/article/7507/item/1",
          "type": "application/pdf"
        },
        {
          "href": "https://example.org/article/7507/item/2",
          "type": "text/csv"
        }
      ],
      "cite-as": [
        {
          "href": "https://doi.org/10.5555/12345680",
          "title": "A Methodology for the Emulation of Architecture"
        }
      ]
    },
    {
      "anchor": "https://example.com/links/article/7507",
      "alternate": [
        {
          "href": "https://mirror.example.com/links/article/7507",
          "type": "application/linkset"
        }
      ]
    }
  ]
}
```

Figure 4: Response to the client's request for the set of links

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

Figure 5 shows a client issuing an HTTP HEAD request against resource `<http://example.org/article/view/7507>`.

```
HEAD article/view/7507 HTTP/1.1
Host: example.org
Connection: close
```

Figure 5: Client HTTP HEAD request

Figure 6 shows the response to the HEAD request of Figure 5. The response contains a "Link" header with a link that has the "linkset" relation type. It indicates that a set of links is provided by resource `<http://example.com/links/article/7507>`, 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: <http://example.com/links/article/7507>
      ; rel="linkset"
      ; type="application/linkset+json"
Content-Length: 236
Content-Type: text/html; charset=utf-8
Connection: close
```

Figure 6: Response to HTTP HEAD request

Section 6.2 shows a client obtaining a set of links by issuing an HTTP GET on the target of the link with the "linkset" relation type, `<http://example.com/links/article/7507>`.

7. 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 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".

7.1. 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.

7.2. 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).

8. IANA Considerations

8.1. Link Relation Type: linkset

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

Relation Name: linkset

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

Reference: [[This document]]

8.2. Media Type: application/linkset

8.2.1. IANA Considerations

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

Type name: application

Subtype name: linkset

Required parameters: none

Optional parameters: none

Encoding considerations: Linksets are encoded according to the definition of [RFC8288]. The encoding of [RFC8288] is based on the general encoding rules of [RFC7230], 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: The interoperability considerations of [RFC7230] 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): 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: none

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:

- o 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.
- o 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. References

10.1. 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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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>>.

10.2. Informative References

- [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>>.
- [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, January 2014, <<https://www.w3.org/TR/2014/REC-json-ld-20140116>>.

10.3. URIs

- [1] <https://www.w3.org/TR/2014/REC-json-ld-20140116/#interpreting-json-as-json-ld>
- [2] <https://tools.ietf.org/html/rfc8288#appendix-A.2>

Appendix A. Acknowledgements

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

Appendix B. 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 [1], 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 3 against the URI of a set of links would be as shown in Figure 7.

```
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: 846

{
  "linkset": [
    {
      "anchor": "https://example.org/article/view/7507",
      "author": [
        {
          "href": "https://orcid.org/0000-0002-1825-0097"
        }
      ],
      "item": [
        {
          "href": "https://example.org/article/7507/item/1",
          "type": "application/pdf"
        },
        {
          "href": "https://example.org/article/7507/item/2",
          "type": "text/csv"
        }
      ],
      "cite-as": [
        {
          "href": "https://doi.org/10.5555/12345680",
          "title": "A Methodology for the Emulation of Architecture"
        }
      ]
    },
    {
      "anchor": "https://example.com/links/article/7507",
      "alternate": [
        {
          "href": "https://mirror.example.com/links/article/7507",
          "type": "application/linkset"
        }
      ]
    }
  ]
}
```

```
    }  
  ]  
}  
]
```

Figure 7: 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 8. This particular JSON-LD context maps "application/linkset+json" representations of link sets to Dublin Core Terms. It also renders each link relation as an absolute URI, inspired by the same approach used for Atom [RFC4287] described in [RFC8288] appendix A.2 [2].

```
HTTP/1.1 200 OK
Content-Type: application/ld+json
Content-Length: 638

{
  "@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"
      }
    },
    {
      "language": "@language",
      "value": "@value",
      "hreflang": {
        "@id": "http://purl.org/dc/terms/language",
        "@container": "@set"
      }
    }
  ]
}
```

Figure 8: JSON-LD Context mapping to schema.org and IANA assignments

Applying the JSON-LD context of Figure 8 to the link set of Figure 7 allows transforming the "application/linkset+json" link set to an RDF link set. Figure 9 shows the latter represented by means of the "text/turtle" RDF serialization.

```
<https://example.org/article/view/7507>
  <http://www.iana.org/assignments/relation/author>
  <https://orcid.org/0000-0002-1825-0097> .

<https://example.org/article/view/7507>
  <http://www.iana.org/assignments/relation/item>
  <https://example.org/article/7507/item/1> .

<https://example.org/article/7507/item/1>
  <http://purl.org/dc/terms/format>
  "application/pdf" .

<https://example.org/article/view/7507>
  <http://www.iana.org/assignments/relation/item>
  <https://example.org/article/7507/item/2> .

<https://example.org/article/7507/item/2>
  <http://purl.org/dc/terms/format>
  "text/csv" .

<https://example.org/article/view/7507>
  <http://www.iana.org/assignments/relation/cite-as>
  <https://doi.org/10.5555/12345680> .

<https://doi.org/10.5555/12345680>
  <http://purl.org/dc/terms/title>
  "A Methodology for the Emulation of Architecture" .

<https://example.com/links/article/7507>
  <http://www.iana.org/assignments/relation/alternate>
  <https://mirror.example.com/links/article/7507> .

<https://mirror.example.com/links/article/7507>
  <http://purl.org/dc/terms/format>
  "application/linkset" .
```

Figure 9: RDF serialization of the link set resulting from applying the JSON-LD context

Note that the JSON-LD context of Figure 8 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.

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>