

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/>

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

S. Dalal
E. Wilde
July 10, 2021

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 11, 2022.

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

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

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

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

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

2. The Deprecation HTTP Response Header

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

2.1. Syntax

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

Deprecation = IMF-fixdate / "true"

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

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

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

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

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

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

Deprecation: true

2.2. Scope

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

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

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

3. The Deprecation Link Relation Type

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

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

3.1. Documentation

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

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

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

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

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

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

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

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

4. Recommend Replacement

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

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

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

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

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

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

5. Sunset

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

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

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

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

6. Resource Behavior

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

7. IANA Considerations

7.1. The Deprecation HTTP Response Header

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

Header Field Name: Deprecation

Applicable Protocol: Hypertext Transfer Protocol (HTTP)

Status: Standard

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

Change controller: IETF

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

7.2. The Deprecation Link Relation Type

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

Relation Type: deprecation

Applicable Protocol: Hypertext Transfer Protocol (HTTP)

Status: Standard

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

Change controller: IETF

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

8. Implementation Status

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

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

8.1. Implementing the Deprecation Header

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

Organization: Apollo

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

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

Organization: Zalando

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

Organization: Palantir Technologies

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

Organization: IETF Internet Draft, Registration Protocols Extensions

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

Organization: E-Voyageurs Technologies

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

Organization: Open-Xchange

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

Organization: MediaWiki

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

8.2. Implementing the Concept

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

Organization: Zapier

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

Organization: IBM

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

Organization: Ultipro

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

Organization: Clearbit

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

Organization: PayPal

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

9. Security Considerations

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

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

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

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

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

10. Examples

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

```
Deprecation: true
```

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

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

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

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

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <<https://www.rfc-editor.org/info/rfc3864>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <<https://www.rfc-editor.org/info/rfc3987>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.

- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

11.2. Informative References

- [RFC5829] Brown, A., Clemm, G., and J. Reschke, Ed., "Link Relation Types for Simple Version Navigation between Web Resources", RFC 5829, DOI 10.17487/RFC5829, April 2010, <<https://www.rfc-editor.org/info/rfc5829>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [RFC8594] Wilde, E., "The Sunset HTTP Header Field", RFC 8594, DOI 10.17487/RFC8594, May 2019, <<https://www.rfc-editor.org/info/rfc8594>>.
- [W3C.REC-html401-19991224] Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, <<https://www.w3.org/TR/1999/REC-html401-19991224>>.

Appendix A. Acknowledgments

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

The authors take all responsibility for errors and omissions.

Authors' Addresses

Sanjay Dalal

Email: sanjay.dalal@cal.berkeley.edu

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

Erik Wilde

Email: erik.wilde@dret.net

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

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

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

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

Abstract

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

Note to Readers

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 6 November 2022.

Copyright Notice

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

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

Table of Contents

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

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

1. Introduction

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

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

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

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

3. Use Cases and Motivation

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

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

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

3.1. Third-Party Links

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

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

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

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

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

3.2. Challenges Writing to HTTP Link Header Field

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

3.3. Large Number of Links

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

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

4. Document Formats for Sets of Links

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

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

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

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

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

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

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

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

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

4.1. HTTP Link Document Format: application/linkset

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

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

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

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

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

4.2. JSON Document Format: application/linkset+json

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

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

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

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

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

4.2.1. Set of Links

In the JSON representation of a set of links:

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

4.2.2. Link Context Object

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

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

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

4.2.3. Link Target Object

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

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

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

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

Figure 1

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

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

Figure 2

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

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

Figure 3

4.2.4. Link Target Attributes

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

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

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

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

4.2.4.1. Target Attributes Defined by Web Linking

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

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

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

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

Figure 4

4.2.4.2. Internationalized Target Attributes

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

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

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

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

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

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

Figure 5

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

4.2.4.3. Extension Target Attributes

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

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

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

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

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

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

Figure 6

4.2.5. JSON Extensibility

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7. Examples

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

7.1. Set of Links Provided as application/linkset

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

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

Figure 7: Client HTTP GET request

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

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

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

```

HEAD resource1 HTTP/1.1
Host: example.org

```

Figure 11: Client HTTP HEAD request

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

```

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

```

Figure 12: Response to HTTP HEAD request

7.4. Link Set Profiles

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

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

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

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

Figure 13: Client HTTP HEAD request

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

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

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

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

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

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

Figure 15: Client HTTP HEAD request

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

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

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

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

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

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

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

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

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

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

8. IANA Considerations

8.1. Link Relation Type: linkset

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

Relation Name: linkset

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

Reference: [[This document]]

8.2. Media Type: application/linkset

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

Type name: application

Subtype name: linkset

Required parameters: N/A

Optional parameters: profile

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

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

Interoperability considerations: N/A

Published specification: [[This document]]

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

Additional information:

Magic number(s): N/A

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

Macintosh file type code(s): TEXT

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

Intended usage: COMMON

Restrictions on usage: none

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

Change controller: IETF

8.3. Media Type: application/linkset+json

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

Type name: application

Subtype name: linkset+json

Required parameters: N/A

Optional parameters: profile

Encoding considerations: The encoding considerations of [RFC8259] apply

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

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

Published specification: [[This document]]

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

Additional information:

Magic number(s): N/A

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

Macintosh file type code(s): TEXT

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

Intended usage: COMMON

Restrictions on usage: none

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

Change controller: IETF

9. Security Considerations

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

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

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

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

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

10. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.

- [W3C.REC-json-ld-20140116]
Sporny, M., Kellogg, G., and M. Lanthaler, "JSON-LD 1.0",
World Wide Web Consortium Recommendation REC-json-ld-
20140116, 16 January 2014,
<<https://www.w3.org/TR/2014/REC-json-ld-20140116>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type
Specifications and Registration Procedures", BCP 13,
RFC 6838, DOI 10.17487/RFC6838, January 2013,
<<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8187] Reschke, J., "Indicating Character Encoding and Language
for HTTP Header Field Parameters", RFC 8187,
DOI 10.17487/RFC8187, September 2017,
<<https://www.rfc-editor.org/info/rfc8187>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
Interchange Format", STD 90, RFC 8259,
DOI 10.17487/RFC8259, December 2017,
<<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288,
DOI 10.17487/RFC8288, October 2017,
<<https://www.rfc-editor.org/info/rfc8288>>.
- [I-D.ietf-httpbis-semantics]
Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP
Semantics", Work in Progress, Internet-Draft, draft-ietf-
httpbis-semantics-19, 12 September 2021,
<[https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-
semantics-19](https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19)>.

11. Informative References

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

- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6906] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <<https://www.rfc-editor.org/info/rfc6906>>.
- [RFC7284] Lanthaler, M., "The Profile URI Registry", RFC 7284, DOI 10.17487/RFC7284, June 2014, <<https://www.rfc-editor.org/info/rfc7284>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [DCMI-TERMS] Initiative, D. C. M., "DCMI Metadata Terms", 2020, <<https://www.dublincore.org/specifications/dublin-core/dcmi-terms/>>.

Appendix A. JSON-LD Context

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

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

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

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

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

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

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

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

```

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

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

```

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

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


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

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

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

Appendix B. Implementation Status

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

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942 [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

B.1. GS1

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

B.2. FAIR Signposting Profile

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

B.3. Open Journal Systems (OJS)

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

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

Acknowledgements

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

Authors' Addresses

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

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