Network Working Group
Internet-Draft
Intended status: Informational
Expires: 15 November 2026

M. F. Callec
Independent

14 May 2026

A Well-Known URI for API Change Advisories
draft-callec-api-advisory-00

Abstract

This document defines a well-known URI, /.well-known/api-advisory.json, at which API providers MAY publish a machine-readable JSON file listing change advisories that affect their API. The file enables automated scanners and API consumers to discover pricing changes, deprecations, security advisories, and other relevant events without relying on manual monitoring of blogs or mailing lists.

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 15 November 2026.

Copyright Notice

Copyright (c) 2026 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.

Table of Contents

1. Introduction

API providers routinely make changes that affect their consumers: endpoints are deprecated, pricing models shift, authentication mechanisms are rotated, security vulnerabilities are disclosed. Today, this information is conveyed through provider blogs, developer mailing lists, and dashboard notifications, all of which require consumers to monitor multiple out-of-band channels and cannot be discovered programmatically.

This document defines a well-known URI ([RFC8615]) at which a provider MAY publish a JSON file containing a structured list of such advisories. Scanners and API clients can poll this URI periodically to detect changes relevant to their integration, without any prior coordination with the provider.

1.1. Requirements Language

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.

1.2. Scope and Limitations

This document is an independent submission and does not represent IETF consensus. It is published as a pre-draft proposal to gather community feedback on the problem space and the proposed approach.

2. The Well-Known URI

The well-known URI defined by this document is:

/.well-known/api-advisory.json

It MUST be served using HTTPS. The advisory file MUST be served from the exact Fully Qualified Domain Name (FQDN) on which the API being described is hosted. If the API is reachable at api.example.net, the advisory file MUST be available at:

https://api.example.net/.well-known/api-advisory.json

The file MUST NOT be considered authoritative for any other FQDN, including parent domains or subdomains. If the advisory file is served at api.example.net, it does not extend to v1.api.example.net, to example.net, or to any other hostname. Each FQDN requiring advisory coverage MUST host its own advisory file.

File location	Applies to	Does NOT apply to
api.example.net/.well-known/api-advisory.json	api.example.net	v1.api.example.net, example.net
v1.api.example.net/.well-known/api-advisory.json	v1.api.example.net	api.example.net
example.net/.well-known/api-advisory.json	example.net	api.example.net

Table 1: FQDN scope examples

3. JSON File Format

The advisory file is a JSON object. Producers MUST serve it with the Content-Type application/json.

3.1. Top-Level Fields

Field	Required	Type	Description
protocol_version	MUST	string	Version of this specification (currently "1.0")
namespace	MUST	string	The FQDN this file is authoritative for
last_updated	MUST	string ([RFC3339])	Datetime of the last change to this file
api_name	MUST	string	Human-readable name of the API
advisories	MUST	array	List of advisory objects, ordered

				most recent first
pagination	MAY	object	Pagination metadata (see Section 6)	

Table 2: Top-level fields

The namespace field MUST exactly match the FQDN from which the file is served, as it would appear in an HTTP Host header ([RFC9110]). Consumers MUST reject or ignore a file where namespace does not match the request host. The namespace field allows consumers aggregating multiple files to unambiguously associate each advisory with its origin, independently of the URL used to fetch the file.

The protocol_version field allows consumers to detect files produced by future revisions of this specification. Consumers MUST check this field before processing any other field. If the value of protocol_version is not a version the consumer implements, the consumer MUST NOT process the file and SHOULD surface an error or warning to the operator. A consumer that silently ignores an unrecognized version risks misinterpreting fields whose semantics may have changed. The only version defined by this document is "1.0".

3.2. Advisory Object Fields

Each element of the advisories array is an advisory object with the following fields.

Field	Required	Type	Description
id	MUST	string	Advisory identifier (see Section 4)
advisory_datetime	MUST	string ([RFC3339])	Datetime when this advisory was published
effective_datetime	MUST	string ([RFC3339])	Datetime when the change takes effect
status	MUST	string (enum)	Lifecycle status (see Section 9)

superseded_by	MUST when status is superseded	string	ID of the replacing advisory
category	MUST	string (enum)	Type of change (see Section 11)
priority	MUST	string (enum)	Severity level (see Section 10)
title	See Section 8	string	English-language title
title_i18n	See Section 8	object	Translated titles
description	See Section 8	string	English-language description
description_i18n	See Section 8	object	Translated descriptions
action_required	MUST	boolean	Whether the consumer must act
suggested_action	See Section 8	string	English-language recommended action
suggested_action_i18n	See Section 8	object	Translated recommended actions
scope	MUST	object	Affected portion of the API (see Section 12)
link	MAY	string (URI)	Link to full documentation or blog post

Table 3: Advisory object fields

4. Advisory Identifier

Advisory identifiers follow the pattern ADV-YYYY-N, where YYYY is a year and N is a positive integer sequence number.

4.1. Uniqueness Scope

Advisory IDs MUST be unique within a single advisory file. Uniqueness is NOT required across different files hosted on different FQDNs. Teams managing separate FQDNs are not expected to coordinate their ID sequences. A globally unambiguous advisory is identified by the tuple (namespace, normalized_id), not by normalized_id alone.

4.2. Normalization

Because the format is intentionally lenient to accommodate different producer conventions, consumers MUST normalize IDs before any comparison or deduplication. Normalization MUST follow these steps:

1. Split the raw string on the - character. The result MUST contain exactly three elements; if not, the ID is malformed and MUST be rejected.

2. Convert the first element to upper-case and verify it equals "ADV"; if not, the ID MUST be rejected as having an unknown prefix.

3. Parse the second and third elements as base-10 integers, discarding any leading zeros. If either element is not a valid integer, the ID MUST be rejected.

The canonical form for storage, comparison, and deduplication is the tuple (prefix, year, seq). Two IDs are equal if and only if their normalized tuples are identical.

4.3. Accepted Variants

The following raw strings all normalize to the same identity:

Raw string	prefix	year	seq
ADV-2026-001	ADV	2026	1
adv-2026-001	ADV	2026	1
ADV-2026-1	ADV	2026	1
ADV-002026-001	ADV	2026	1
adv-002026-1	ADV	2026	1

Table 4: Equivalent advisory ID representations

4.4. Producer Recommendations

Producers SHOULD emit IDs in the form ADV-YYYY-NNN (upper-case prefix, 4-digit year, sequence number zero-padded to at least 3 digits) for human readability. This is a SHOULD, not a MUST; parsers MUST handle all valid variants.

4.5. Reference Implementation

The following JavaScript functions implement normalization, equality testing, and key generation for use in deduplication structures such as Map or Set. They are provided for illustrative purposes and do not form a normative part of this specification.

```
/**
 * Parses and normalizes an advisory ID string into its three
 * components. The returned tuple {prefix, year, seq} MUST be used
 * for all comparisons (never compare raw ID strings directly).
 *
 * @param {string} id  Raw advisory ID (e.g. "ADV-2026-001")
 * @returns {{ prefix: string, year: number, seq: number }}
 * @throws {Error}     If the ID is malformed or contains
 *                     non-numeric segments
 */
function parseAdvisoryId(id) {
  const blocks = id.split("-");
  if (blocks.length !== 3)
    throw new Error(‘Malformed advisory ID: "${id}"‘);

  const prefix = blocks[0].toUpperCase();
  if (prefix !== "ADV")
```

```
      throw new Error(`Unknown prefix "${prefix}" in ID: "${id}"`);

    const year = parseInt(blocks[1], 10);
    const seq  = parseInt(blocks[2], 10);

    if (isNaN(year) || isNaN(seq))
      throw new Error(`Non-numeric segment in ID: "${id}"`);

    return { prefix, year, seq };
  }

  /**
   * Returns true if two raw ID strings refer to the same advisory,
   * regardless of casing or zero-padding.
   */
  function advisoryIdsEqual(a, b) {
    const pa = parseAdvisoryId(a);
    const pb = parseAdvisoryId(b);
    return pa.prefix === pb.prefix
        && pa.year   === pb.year
        && pa.seq    === pb.seq;
  }

  /**
   * Returns a stable string key suitable for use in a Map or Set.
   * For cross-namespace deduplication, prefix this key with the
   * namespace: `${namespace}::${advisoryIdKey(id)}`.
   *
   * @param {string} id  Raw advisory ID
   * @returns {string}   Canonical key, e.g. "ADV-2026-1"
   */
  function advisoryIdKey(id) {
    const { prefix, year, seq } = parseAdvisoryId(id);
    return `${prefix}-${year}-${seq}`;
  }
```

5. Ordering

Advisories MUST be listed in reverse chronological order by advisory_datetime, most recent first. This ordering allows consumers to stop processing the list as soon as they encounter an advisory with an advisory_datetime older than their last known checkpoint.

6. Pagination

Pagination is OPTIONAL. A producer with a large number of advisories MAY split the list across multiple pages by including a pagination object at the top level. If no pagination field is present, the advisories array contains the complete list.

6.1. Pagination Fields

Field	Required	Type	Description
total	MAY	integer	Total advisory count across all pages
page	MUST	integer	Current page number, 1-indexed
page_size	MUST	integer	Maximum advisories per page
next	MAY	string (URI)	URL of the next (older) page
prev	MAY	string (URI)	URL of the previous (more recent) page

Table 5: Pagination fields

The next field MUST be omitted on the last page. The prev field MUST be omitted on the first page. Because advisories are ordered most recent first, next points toward older content and prev toward more recent content.

The first page MUST be accessible at the well-known URI without any query parameters. Subsequent pages MAY use query parameters or any other stable URL convention, as long as each page returns a valid advisory file conforming to this specification.

7. Caching

Producers SHOULD set a Cache-Control header (see [RFC7234]) on the advisory file response. A recommended value is:

Cache-Control: public, max-age=3600

A max-age of less than 60 seconds is NOT RECOMMENDED, as it provides no meaningful benefit over uncached responses. Consumers SHOULD respect the Cache-Control header when polling, and SHOULD also compare the top-level last_updated field to the value observed on their last successful fetch to determine whether the file content has actually changed.

8. Localization

The fields title, description, and suggested_action carry human-readable content intended for operators and developers, not for automated parsers. Each of these fields has a plain-string variant and an i18n object variant.

For each field, at least one of the two variants MUST be present. Both variants MAY be present simultaneously in the same advisory object.

8.1. Plain-String Variant

The plain-string variant (title, description, suggested_action) carries an English-language string. Producers that do not require multilingual output SHOULD use this variant.

"title": "Webhooks endpoint moving to paid model"

8.2. i18n Object Variant

The i18n object variant (title_i18n, description_i18n, suggested_action_i18n) is a JSON object whose keys are language tags conforming to [RFC5646] and whose values are the corresponding translated strings. The key "en" MUST be present in every i18n object.

```
"title_i18n": {
  "en": "Webhooks endpoint moving to paid model",
  "fr": "Le endpoint webhooks passe a un modele payant",
  "de":
"Der Webhooks-Endpunkt wechselt zu einem kostenpflichtigen Modell"
}
```

8.3. Resolution Rules

When both variants are present for a field, consumers MUST resolve field values as follows:

1. For English: the plain-string variant takes precedence. The "en" key of the i18n object MUST be ignored when the plain-string variant is also present.

2. For any language other than English: the i18n object variant is the only source.

3. If no match is found for the consumer’s preferred language, the consumer MUST fall back to the English value resolved by rule 1.

The plain-string variant always takes precedence over the "en" key of the i18n variant.

8.4. Mixing Variants

The two variants MAY be mixed freely across fields within the same advisory object. For example, title MAY be a plain string while description_i18n provides multilingual translations.

9. Advisory Status

Each advisory MUST carry a status field indicating its current lifecycle state. Producers MUST NOT remove advisories from the file once published; instead they MUST update the status field to reflect the advisory’s current state. This preserves the historical record and allows consumers that previously processed an advisory to detect status changes on subsequent polls.

9.1. Status Values

Value	Meaning
active	The advisory is current and applicable
withdrawn	The advisory was retracted and should be disregarded
superseded	The advisory has been replaced by a newer one

Table 6: Advisory status values

When status is superseded, the superseded_by field MUST be present and MUST contain the normalized ID of the replacing advisory. The replacing advisory MUST exist within the same namespace (i.e., the same advisory file or a subsequent page of the same paginated file). Consumers SHOULD fetch and process the replacement advisory before acting on the superseded one.

10. Priority Levels

The priority field indicates the urgency of the advisory.

Value	Meaning
critical	Immediate action required; breaking or security impact
high	Action required before the effective datetime
medium	Action recommended; graceful degradation is possible
low	Minor or optional change
info	No action needed; informational only

Table 7: Priority values

11. Categories

The category field indicates the type of change being announced.

Value	Meaning
pricing_change	Free tier ending, price change, or quota change
legal_update	Terms of service, privacy policy, or data residency
compliance_update	Change imposed by an external regulation (e.g. GDPR, PCI-DSS)
deprecation	A feature or endpoint marked for future

	removal
sunset	A feature or endpoint being permanently removed
end_of_life	An entire API version being permanently decommissioned
breaking_change	A non-backward-compatible API change
maintenance	Planned downtime or degraded availability
incident	An ongoing or recently resolved incident
migration_required	Consumers must migrate to a new system
security_advisory	Vulnerability or key rotation requiring consumer action
credential_rotation	API keys, certificates, or OAuth secrets are being rotated
performance_update	SLA or throughput limit change
new_feature	A new endpoint or capability (non-breaking)
ownership_transfer	The API has been acquired, transferred, or rebranded
endpoint_moved	A path or domain has changed
rate_limit_change	Request quotas or throttling has been updated
data_retention_update	Data storage duration or policy has changed
region_change	Geographic availability has changed (new region, closure, or data relocation)

Table 8: Category values

12. Scope

The scope object describes which portion of the API is affected by an advisory. It uses a progressive model: a consumer can stop reading scope as soon as the level field indicates an impact broad enough to be relevant.

12.1. Scope Fields

Field	Required	Type	Description
level	MUST	string (enum)	Granularity of the scope
versions	MUST when level is versions; MAY when level is routes	array of strings	Affected API versions
routes	MUST when level is routes	array of objects	Affected routes

Table 9: Scope object fields

12.2. Level Values

Value	Meaning
global	The entire API is affected; versions and routes are ignored
versions	Only the listed versions are affected; routes is ignored
routes	Only the listed routes are affected; versions is an optional additional filter

Table 10: Scope level values

12.3. Route Objects

When level is routes, the routes array MUST be present and MUST contain one or more route objects.

Field	Required	Type	Description
method	MUST	string	HTTP method, or * to match all methods
path	MUST	string	Route path pattern (see Section 12.4)

Table 11: Route object fields

12.4. Path Pattern Syntax

The path field contains a pattern that is matched against the path component of request URIs. This section defines the syntax and matching semantics of these patterns normatively.

12.4.1. Syntax

Path patterns are defined using the following ABNF grammar ([RFC5234]). The rule pchar is imported from Section 3.3 of [RFC3986] and covers unreserved characters, percent-encoded characters, sub-delimiters, colons, and at-signs.

```
path-pattern     = "/" *( segment "/" ) terminal

segment          = 1*pchar
                   ; A literal path segment; no wildcards allowed.
                   ; MUST NOT be empty.

terminal         = segment
                 / wildcard-single
                 / wildcard-multi

wildcard-single  = "*"
                   ; Matches exactly one non-empty segment.

wildcard-multi   = "**"
                   ; Matches one or more segments at any depth.

pchar            = unreserved / pct-encoded / sub-delims / ":" / "@"
                   ; Imported from Section 3.3 of RFC 3986.
```

A path pattern MUST begin with a / character. The empty string and patterns that do not begin with / are malformed and MUST be rejected by consumers.

Wildcards (* and **) are only permitted as an entire terminal segment, they MUST NOT appear embedded within a literal segment (e.g., /v2/web* is invalid). A pattern containing an embedded wildcard MUST be rejected by consumers.

12.4.2. Matching Semantics

For the purpose of pattern matching, a subject path is split into a sequence of segments by splitting on / and discarding empty strings produced by leading, trailing, or consecutive / characters. A pattern is split into its component segments in the same way.

The following rules apply:

* A *literal segment* matches a subject segment if and only if the two strings are identical after percent-decoding both (case-sensitive comparison).

* A ** wildcard* (wildcard-single) matches exactly one subject segment. It MUST NOT match an empty segment. It MUST NOT match across a / boundary, i.e., it matches at most one path component.

* A *** wildcard* (wildcard-multi) matches one or more consecutive subject segments. It MUST NOT match zero segments. It may match segments at any depth, i.e., it consumes one or more path components including their intervening / characters.

* A pattern matches a subject path if and only if the entire sequence of subject segments is consumed by the pattern segments, with no subject segments remaining after the last pattern segment is applied.

Consumers MUST apply these rules deterministically. Where a ** wildcard could match a variable number of segments, it MUST be treated greedily: it consumes the maximum number of segments that still allows the overall pattern to match. In practice, because ** is only permitted as a terminal segment in this grammar, no backtracking is required.

12.4.3. Matching Examples

The following table illustrates the matching rules. All examples assume case-sensitive comparison after percent-decoding.

Pattern	Subject path	Match?	Reason
/v2/webhooks	/v2/webhooks	Yes	Exact literal match
/v2/webhooks	/v2/webhooks/	Yes	Trailing slash discarded before matching
/v2/webhooks	/v2/webhooks/123	No	Extra segment not covered
/v2/webhooks/*	/v2/webhooks/abc	Yes	* matches single segment abc
/v2/webhooks/*	/v2/webhooks/abc/def	No	* does not match across /
/v2/webhooks/*	/v2/webhooks/	No	* does not match an empty segment
/v2/webhooks/**	/v2/webhooks/abc	Yes	** matches one segment
/v2/webhooks/**	/v2/webhooks/abc/def/ghi	Yes	** matches multiple segments
/v2/webhooks/**	/v2/webhooks	No	** requires at least one segment
/v1/**	/v1/users/123/orders	Yes	** matches remaining segments
/v1/**	/v2/users	No	Literal v1 does not match v2
/v2/web*	/v2/webhooks	Invalid	Embedded wildcard; pattern MUST be rejected

Table 12: Path pattern matching examples

12.5. Scope Semantics

The following normative rules apply:

* When level is "global", versions and routes MUST be omitted or, if present, MUST be ignored by consumers.

* When level is "versions", versions is REQUIRED; routes MUST be omitted or, if present, MUST be ignored by consumers.

* When level is "routes", routes is REQUIRED; versions is OPTIONAL and, if present, acts as an additional filter, the advisory applies only to the listed routes within the listed versions.

13. Example

The following example shows a paginated first page containing three advisories in reverse chronological order. It demonstrates: the namespace field; the status field including a superseded advisory; the localization resolution rules (ADV-2026-003 carries both a plain-string title and a title_i18n object without an "en" key, illustrating that the plain-string variant takes precedence for English while the i18n object supplies other languages); and the pagination object. The Cache-Control header shown after the JSON is a separate HTTP response header, not part of the JSON body.

```
{
  "protocol_version": "1.0",
  "namespace": "api.acme.com",
  "last_updated": "2026-05-13T20:45:00Z",
  "api_name": "Acme Payments API",
  "pagination": {
    "total": 42,
    "page": 1,
    "page_size": 3,
    "next":
      "https://api.acme.com/.well-known/api-advisory.json?page=2"
  },
  "advisories": [
    {
      "id": "ADV-2026-003",
      "advisory_datetime": "2026-05-13T14:00:00Z",
      "effective_datetime": "2027-01-01T00:00:00Z",
      "status": "active",
      "category": "deprecation",
      "priority": "high",
```

```
        "title":
          "Deprecation of query parameter authentication (revised)",
        "title_i18n": {
          "fr":
   "Depreciation de l’authentification par parametre (revisee)"
        },
        "description":
   "The migration deadline has been extended to January 1, 2027.",
        "action_required": true,
        "suggested_action":
          "Replace the api_key query parameter with a Bearer token.",
        "scope": { "level": "global" },
        "link": "https://docs.acme.com/auth-migration"
      },
      {
        "id": "ADV-2026-002",
        "advisory_datetime": "2026-05-13T09:00:00Z",
        "effective_datetime": "2026-10-01T00:00:00Z",
        "status": "superseded",
        "superseded_by": "ADV-2026-003",
        "category": "deprecation",
        "priority": "medium",
        "title_i18n": {
          "en": "Deprecation of query parameter authentication",
          "fr":
   "Depreciation de l’authentification par parametre de requete"
        },
        "description":
   "Authentication via the api_key query parameter is deprecated.",
        "action_required": true,
        "suggested_action":
          "Migrate to the Authorization HTTP header.",
        "scope": { "level": "global" },
        "link": "https://docs.acme.com/auth-migration"
      },
      {
        "id": "ADV-2026-001",
        "advisory_datetime": "2026-05-10T10:00:00Z",
        "effective_datetime": "2026-12-01T00:00:00Z",
        "status": "active",
        "category": "pricing_change",
        "priority": "high",
        "title": "Webhooks endpoint moving to paid model",
        "description":
          "Webhook usage will be billed at $0.01 per call.",
        "action_required": true,
        "suggested_action":
   "Review your webhook usage and update your billing plan.",
```

```
        "scope": {
          "level": "routes",
          "versions": ["v2"],
          "routes": [
            { "method": "POST", "path": "/v2/webhooks" },
            { "method": "*",    "path": "/v2/webhooks/**" }
          ]
        },
        "link": "https://acme.com/blog/pricing-2026"
      }
    ]
  }
```

The HTTP response SHOULD include:

Cache-Control: public, max-age=3600

14. IANA Considerations

14.1. Well-Known URI Registration

This document requests the registration of the following well-known URI in the "Well-Known URIs" registry established by [RFC8615]:

* URI suffix: api-advisory.json
* Change controller: IETF
* Specification document: this document
* Related information: The resource is a JSON object as defined in Section 3.

15. Security Considerations

15.1. HTTPS Requirement

Consumers MUST only fetch advisory files over HTTPS. A file served over plain HTTP is subject to manipulation in transit and MUST NOT be trusted.

15.2. Namespace Validation

Consumers MUST verify that the namespace field in the advisory file exactly matches the FQDN from which the file was retrieved. Failure to do so may allow a malicious party to serve an advisory file for a different domain, causing consumers to apply advisories to the wrong API.

15.3. Availability

An advisory file that is temporarily or permanently unavailable MUST be treated as a transient error. Consumers MUST NOT interpret the absence of an advisory file as confirmation that no advisories exist. Producers SHOULD ensure that the advisory endpoint is subject to the same availability guarantees as their API.

15.4. Content Integrity

This specification does not define a mandatory mechanism for signing advisory files. Consumers relying on advisory content for automated decisions (e.g., automated circuit-breaking) SHOULD consider whether additional integrity guarantees are appropriate for their threat model. One approach is HTTP Message Signatures ([RFC9421]), which allows producers to sign HTTP responses (including the advisory file response) using asymmetric keys. Consumers can then verify the signature before processing the file content, without relying solely on HTTPS for authenticity.

15.5. Historical Record Preservation

Producers MUST NOT remove advisories from the file after they have been published. Removing a superseded or withdrawn advisory may cause consumers that had previously observed it to re-process it as a new advisory on a subsequent poll, depending on their deduplication implementation.

16. References

16.1. 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/rfc/rfc2119>.

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

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

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

[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/rfc/rfc5646>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

16.2. Informative References

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

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/rfc/rfc9421>.

Author’s Address

Mateo Florian Callec
Independent
France
Email: mateo@callec.net