Network Working Group
Internet-Draft
Intended status: Informational
Expires: 25 November 2026

M. F. Callec
Independent

24 May 2026

A Well-Known URI for API Change Advisories
draft-callec-api-advisory-01

Abstract

This document defines a well-known URI, /.well-known/api-advisory.json, at which API providers MAY publish a machine-readable JSON discovery file. That file identifies the API and provides the URL of an Atom Syndication Format feed carrying structured change advisories. The Atom feed 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. Advisory metadata is conveyed through a dedicated XML namespace defined in this document.

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 25 November 2026.

Copyright Notice

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 discovery file. That file, in turn, points to an Atom feed ([RFC4287]) that carries the advisory entries. Delegating feed semantics, pagination, ordering, and syndication to Atom avoids reinventing those mechanisms and allows implementers to reuse existing Atom libraries. The advisory-specific metadata is conveyed through a dedicated XML namespace defined in Section 5.

Scanners and API clients can poll the well-known URI to locate the feed, then poll the feed itself 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. Terminology

The following terms are used throughout this document.

API provider: An entity that operates an HTTP-based service accessible at one or more FQDNs and that exposes a defined programmatic interface to external consumers.

FQDN: Fully Qualified Domain Name, as used in the HTTP Host header ([RFC9110]).

Advisory: A single structured announcement published by an API provider to inform consumers of a change that affects their integration.

1.3. 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 discovery file MUST be served from the exact FQDN on which the API being described is hosted. If the API is reachable at api.example.net, the discovery 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 discovery 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 discovery 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. Discovery File Format

The discovery 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
feed_url	MUST	string (URI)	URL of the Atom feed carrying advisories

Table 2: Discovery file 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 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. The only version defined by this document is "1.0".

The feed_url field MUST be an absolute URI ([RFC3986]) using the https scheme. It points to an Atom feed document as defined in Section 4. The feed URL MAY be on a different FQDN than the discovery file, for example if the provider hosts its advisory feed on a dedicated infrastructure domain.

3.2. Example Discovery File

```
{
  "protocol_version": "1.0",
  "namespace": "api.example.com",
  "last_updated": "2026-05-13T20:45:00Z",
  "api_name": "Example Payments API",
  "feed_url":
    "https://api.example.com/.well-known/api-advisory-feed.atom"
}
```

4. Atom Feed

Advisories are published as entries in an Atom Syndication Format feed ([RFC4287]). The feed MUST be served using HTTPS. Producers MUST serve the feed with the Content-Type application/atom+xml.

Using Atom delegates feed semantics, ordering, and pagination to a well-understood and widely implemented standard, allowing API providers to reuse existing Atom server infrastructure and consumers to reuse existing Atom client libraries.

4.1. Feed-Level Elements

The Atom <feed> element MUST include the following child elements as defined by [RFC4287]:

* <id>: A permanent, universally unique IRI identifying the feed.

* <title>: A human-readable title for the feed.

* <updated>: The datetime of the most recent advisory entry, in [RFC3339] format.

* <link rel="self">: The URL of the feed itself.

Producers SHOULD also include <link rel="alternate"> pointing to a human-readable page about the API.

4.2. Entry-Level Elements

Each advisory is represented as an Atom <entry> element. Entries MUST be ordered from most recent to oldest by <updated> value. This ordering allows consumers to stop processing the feed as soon as they encounter an entry with an <updated> value older than their last known checkpoint.

Each entry MUST include:

* <id>: A permanent, universally unique IRI identifying the advisory (see Section 6).

* <title>: A human-readable title for the advisory.

* <updated>: The datetime when the advisory was last modified.

* <published>: The datetime when the advisory was first published.

* <summary> or <content>: A human-readable description of the advisory.

Each entry MUST also include one <api:advisory> element from the advisory namespace (Section 5) carrying structured metadata.

Producers MAY include <link rel="alternate"> on each entry pointing to a full documentation page or blog post.

Producers MAY include <link rel="enclosure"> on each entry for attachments such as migration guides.

4.3. Localization

Atom natively supports per-element xml:lang attributes ([RFC4287], Section 2). Producers MAY provide multilingual <title> and <summary> elements by repeating those elements with distinct xml:lang attributes, following standard Atom practice.

5. Advisory XML Namespace

This document defines the XML namespace:

https://iana.org/api-advisory/1.0

(This namespace URI is a placeholder pending IANA registration.) In the examples in this document, the prefix api: is used for this namespace.

5.1. The <api:advisory> Element

Each Atom <entry> MUST contain exactly one <api:advisory> element. This element carries the structured metadata specific to this specification.

5.1.1. Child Elements of <api:advisory>

Element	Required	Type	Description
<api:id>	MUST	string	Advisory identifier (see Section 6)
<api:advisory_datetime>	MUST	string ([RFC3339])	Datetime when first published
<api:effective_datetime>	MUST	string ([RFC3339])	Datetime when the change takes effect
<api:status>	MUST	string (enum)	Lifecycle status (see Section 9)
<api:superseded_by>	MUST when status is superseded	string	ID of the replacing advisory
<api:category>	MUST	string (enum)	Type of change (see Section 11)
<api:priority>	MUST	string (enum)	Severity level (see Section 10)
<api:action_required>	MUST	boolean (true/ false)	Whether the consumer must act
<api:suggested_action>	SHOULD	string	Recommended action (English)

<api:scope>	MUST	element	Affected portion of the API (see Section 12)

Table 3: Child elements of api:advisory

6. Advisory Identifier

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

The Atom <entry><id> element MUST be a URI. Producers MUST construct this URI by appending the advisory identifier to a stable base IRI specific to their feed, for example:

https://api.example.com/advisories/ADV-2026-003

The <api:id> element inside <api:advisory> carries the short-form identifier string (e.g. ADV-2026-003) for use in structured processing and cross-reference.

6.1. Uniqueness Scope

Advisory IDs MUST be unique within a single advisory feed. Uniqueness is NOT required across different feeds 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.

6.2. 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.

6.3. 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

6.4. 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.

6.5. 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}`;
}
```

7. Pagination

Atom natively supports feed pagination through the use of <link> elements with rel="next" and rel="prev" relation types. Producers with a large number of advisories SHOULD use these standard link relations to split the feed across multiple pages, rather than defining a custom pagination mechanism.

Because entries are ordered most recent first, rel="next" points toward older entries and rel="prev" points toward more recent entries.

The feed URL returned in the feed_url field of the discovery file (Section 3) MUST point to the first (most recent) page of the feed.

8. Caching

Producers SHOULD set a Cache-Control header (see [RFC7234]) on the advisory feed 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 feed-level <updated> element to the value observed on their last successful fetch to determine whether the feed content has actually changed.

9. Advisory Status

Each advisory MUST carry an <api:status> element indicating its current lifecycle state. Producers MUST NOT remove advisory entries from the feed once published; instead they MUST update the <api:status> element to reflect the advisory's current state and update the Atom <updated> element accordingly. 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 5: Advisory status values

When <api:status> is superseded, the <api:superseded_by> element MUST be present and MUST contain the normalized ID of the replacing advisory. The replacing advisory MUST exist within the same feed (or a subsequent page of the same paginated feed). Consumers SHOULD fetch and process the replacement advisory before acting on the superseded one.

10. Priority Levels

The <api:priority> element 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 6: Priority values

11. Categories

The <api:category> element 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 7: Category values

12. Scope

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

12.1. Multiple Distinct APIs at the Same FQDN

A single FQDN MAY host multiple distinct APIs (for example, /payments/** and /identity/**). The scope mechanism defined in this section handles this case explicitly. An advisory with <api:level>global</api:level> applies to all APIs hosted at the FQDN. An advisory with <api:level>routes</api:level> targets a specific API or set of endpoints by path prefix or explicit route list. Producers operating multiple distinct APIs at the same FQDN SHOULD use route-level scope to avoid ambiguity, and SHOULD use path prefixes that clearly identify the target API (e.g., /payments/** or /identity/**). Consumers MUST NOT assume that a global-scoped advisory applies only to the API they are currently integrating with; they SHOULD surface it as affecting the entire FQDN.

12.2. Scope Child Elements

Element	Required	Type	Description
<api:level>	MUST	string (enum)	Granularity of the scope
<api:versions>	MUST when level is versions; MAY when level is routes	element	Affected API versions

<api:routes>	MUST when level is routes	element	Affected routes

Table 8: Scope child elements

12.3. Level Values

Value	Meaning
global	The entire FQDN is affected; <api:versions> and <api:routes> are ignored
versions	Only the listed versions are affected; <api:routes> is ignored
routes	Only the listed routes are affected; <api:versions> is an optional additional filter

Table 9: Scope level values

12.4. Versions Element

When <api:level> is versions, the <api:versions> element MUST be present and MUST contain one or more <api:version> child elements. When <api:level> is routes, <api:versions> MAY be present and, if so, MUST contain one or more <api:version> child elements.

Each <api:version> element MUST contain a single version identifier string as its text content. The version identifier is an opaque string and is compared case-sensitively. Producers SHOULD use the same version identifiers as those exposed in the API itself (e.g., v1, v2).

12.5. Route Elements

When <api:level> is routes, the <api:routes> element MUST be present and MUST contain one or more <api:route> child elements. Each <api:route> element MUST contain:

Child element	Required	Description
<api:method>	MUST	HTTP method, or * to match all methods
<api:path>	MUST	Route path pattern (see Section 12.6)

Table 10: Route child elements

12.6. Path Pattern Syntax

The <api:path> element 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.

12.6.1. 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.

12.6.2. 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.

12.6.3. 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 11: Path pattern matching examples

12.7. Scope Semantics

The following normative rules apply:

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

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

* When <api:level> is "routes", <api:routes> is REQUIRED; <api: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 complete Atom feed (first page) containing three advisories in reverse chronological order. It demonstrates: the advisory namespace; the <api:status> element including a superseded advisory; localization via xml:lang; and feed-level pagination via rel="next".

13.1. Discovery File

```
{
  "protocol_version": "1.0",
  "namespace": "api.example.com",
  "last_updated": "2026-05-13T20:45:00Z",
  "api_name": "Example Payments API",
  "feed_url":
    "https://api.example.com/.well-known/api-advisory-feed.atom"
}
```

13.2. Atom Feed

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:api="https://iana.org/api-advisory/1.0">

  <id>https://api.example.com/advisories/feed</id>
  <title>Example Payments API Advisories</title>
  <updated>2026-05-13T14:00:00Z</updated>
  <link rel="self"
  href="https://api.example.com/.well-known/api-advisory-feed.atom"/>
```

```
    <link rel="alternate" href="https://docs.example.com/advisories"/>
    <link rel="next"
      href=
  "https://api.example.com/.well-known/api-advisory-feed.atom?page=2"/>

    <!-- ADV-2026-003: active, deprecation (revised deadline) -->
    <entry>
      <id>https://api.example.com/advisories/ADV-2026-003</id>
      <title xml:lang="en">
        Deprecation of query parameter authentication (revised)
      </title>
      <title xml:lang="fr">
        Depreciation de l'authentification par parametre (revisee)
      </title>
      <published>2026-05-13T14:00:00Z</published>
      <updated>2026-05-13T14:00:00Z</updated>
      <summary xml:lang="en">
        The migration deadline has been extended to January 1, 2027.
      </summary>
      <link rel="alternate"
        href="https://docs.example.com/auth-migration"/>
      <api:advisory>
        <api:id>ADV-2026-003</api:id>
        <api:advisory_datetime>
          2026-05-13T14:00:00Z
        </api:advisory_datetime>
        <api:effective_datetime>
          2027-01-01T00:00:00Z
        </api:effective_datetime>
        <api:status>active</api:status>
        <api:category>deprecation</api:category>
        <api:priority>high</api:priority>
        <api:action_required>true</api:action_required>
        <api:suggested_action>
          Replace the api_key query parameter with a Bearer token.
        </api:suggested_action>
        <api:scope>
          <api:level>global</api:level>
        </api:scope>
      </api:advisory>
    </entry>

    <!-- ADV-2026-002: superseded by ADV-2026-003 -->
    <entry>
      <id>https://api.example.com/advisories/ADV-2026-002</id>
      <title xml:lang="en">
        Deprecation of query parameter authentication
      </title>
```

```
      <title xml:lang="fr">
        Depreciation de l'authentification par parametre de requete
      </title>
      <published>2026-05-13T09:00:00Z</published>
      <updated>2026-05-13T14:00:00Z</updated>
      <summary xml:lang="en">
        Authentication via the api_key query parameter is deprecated.
      </summary>
      <link rel="alternate"
        href="https://docs.example.com/auth-migration"/>
      <api:advisory>
        <api:id>ADV-2026-002</api:id>
        <api:advisory_datetime>
          2026-05-13T09:00:00Z
        </api:advisory_datetime>
        <api:effective_datetime>
          2026-10-01T00:00:00Z
        </api:effective_datetime>
        <api:status>superseded</api:status>
        <api:superseded_by>ADV-2026-003</api:superseded_by>
        <api:category>deprecation</api:category>
        <api:priority>medium</api:priority>
        <api:action_required>true</api:action_required>
        <api:suggested_action>
          Migrate to the Authorization HTTP header.
        </api:suggested_action>
        <api:scope>
          <api:level>global</api:level>
        </api:scope>
      </api:advisory>
    </entry>

    <!-- ADV-2026-001: active, pricing change on webhooks routes -->
    <entry>
      <id>https://api.example.com/advisories/ADV-2026-001</id>
      <title xml:lang="en">
        Webhooks endpoint moving to paid model
      </title>
      <published>2026-05-10T10:00:00Z</published>
      <updated>2026-05-10T10:00:00Z</updated>
      <summary xml:lang="en">
        Webhook usage will be billed at $0.01 per call.
      </summary>
      <link rel="alternate"
        href="https://example.com/blog/pricing-2026"/>
      <api:advisory>
        <api:id>ADV-2026-001</api:id>
        <api:advisory_datetime>
```

```
          2026-05-10T10:00:00Z
        </api:advisory_datetime>
        <api:effective_datetime>
          2026-12-01T00:00:00Z
        </api:effective_datetime>
        <api:status>active</api:status>
        <api:category>pricing_change</api:category>
        <api:priority>high</api:priority>
        <api:action_required>true</api:action_required>
        <api:suggested_action>
          Review your webhook usage and update your billing plan.
        </api:suggested_action>
        <api:scope>
          <api:level>routes</api:level>
          <api:versions>
            <api:version>v2</api:version>
          </api:versions>
          <api:routes>
            <api:route>
              <api:method>POST</api:method>
              <api:path>/v2/webhooks</api:path>
            </api:route>
            <api:route>
              <api:method>*</api:method>
              <api:path>/v2/webhooks/**</api:path>
            </api:route>
          </api:routes>
        </api:scope>
      </api:advisory>
    </entry>

  </feed>
```

The HTTP response for the feed 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 discovery object as defined in Section 3, pointing to an Atom feed as defined in Section 4.

14.2. XML Namespace Registration

This document requests the registration of the following XML namespace:

* URI: https://iana.org/api-advisory/1.0

* Registrant Contact: the author of this document

* XML: None; the namespace does not have an associated schema at this time.

15. Security Considerations

15.1. HTTPS Requirement

Consumers MUST only fetch discovery files and advisory feeds over HTTPS. A file or feed 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 discovery file exactly matches the FQDN from which the file was retrieved. Failure to do so may allow a malicious party to serve a discovery file for a different domain, causing consumers to subscribe to an advisory feed for the wrong API.

15.3. Feed URL Validation

Consumers MUST verify that the feed_url value uses the https scheme before fetching the feed. Consumers SHOULD also verify that the feed URL is not a redirect to a non-HTTPS location.

15.4. Availability

A discovery file or advisory feed that is temporarily or permanently unavailable MUST be treated as a transient error. Consumers MUST NOT interpret the absence of a discovery file or feed as confirmation that no advisories exist. Producers SHOULD ensure that both the discovery and feed endpoints are subject to the same availability guarantees as their API.

15.5. Content Integrity

This specification does not define a mandatory mechanism for signing advisory feeds. 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 feed response) using asymmetric keys. Consumers can then verify the signature before processing the feed content, without relying solely on HTTPS for authenticity.

15.6. Historical Record Preservation

Producers MUST NOT remove advisory entries from the feed after they have been published. Removing a superseded or withdrawn entry 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>.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <https://www.rfc-editor.org/rfc/rfc4287>.

[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>.

[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>.

Acknowledgements

The author would like to thank Dale R. Worley for his review and feedback on the mailing list, in particular for suggesting the use of Atom ([RFC4287]) as the advisory delivery mechanism.

Author’s Address

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