

Individual Submission
Internet-Draft
Intended status: Informational
Expires: 16 October 2026

J. McConnell
Ask McConnell, LLC
14 April 2026

A Well-Known URI for Software Lifecycle Status
draft-mcconnell-software-status-wellknown-00

Abstract

This document defines a Well-Known URI [RFC8615] at which software vendors and open-source maintainers may publish machine-readable lifecycle status information for their products. A JSON resource retrieved from `/.well-known/software-status.json` allows consumers — including security tools, software composition analysis (SCA) platforms, vulnerability scanners, and system administrators — to programmatically determine whether a specific version of a software product is actively supported, in long-term support (LTS), under security-only maintenance, or at end-of-life (EOL).

This document also describes, in Appendix A (`#appendix-a`), a companion convention for open-source projects hosted on version-control platforms to publish equivalent information within the repository at `.github/software-status.json`.

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 16 October 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	3
1.1. Conventions and Definitions	4
2. The software-status.json Resource	4
2.1. Location and Retrieval	4
2.2. Caching	5
3. The software-status.json Schema	5
3.1. Root Object	5
3.2. Version Entries	6
3.3. Status Values	6
3.4. Package Identifiers	7
4. Processing Rules for Consuming Tools	8
4.1. Discovery	8
4.2. Version Matching	8
4.3. Conflict Resolution	8
5. Security Considerations	8
5.1. Authenticity	9
5.2. Denial of Service	9
5.3. Misleading Declarations	9
5.4. Privacy	9
6. IANA Considerations	9
6.1. Well-Known URI Registration	10
7. References	10
7.1. Normative References	10
7.2. Informative References	10
Appendix A. Examples	10
A.1. Minimal Resource	10
A.2. Multi-Version Resource with EOL History	11
Appendix B. Companion Convention: Repository-Hosted Lifecycle Status	13
B.1. Motivation	13
B.2. Location	13
B.3. Discovery	13
B.4. Schema	14
B.5. Precedence	14
B.6. Example	14
Appendix C. Related Work	15
Appendix D. Acknowledgements	15
Author's Address	15

1. Introduction

Software products have defined lifecycles. Vendors release versions, provide security patches and bug fixes for a period, and eventually discontinue support. When support ends, no further patches are issued — including patches for newly discovered security vulnerabilities. End-of-life (EOL) software represents a significant and persistent risk in organizational environments, as it may remain deployed indefinitely after its support window closes.

Determining whether a specific version of software is currently supported requires manual research: locating the vendor's lifecycle policy page, interpreting its contents, and correlating the installed version against published support dates. This process is not standardized, not machine-readable, and not reliable at scale. Many vendors do not publish structured lifecycle data at all. Security tools that attempt to automate EOL detection must resort to web scraping, curated third-party databases, or inference — all of which introduce latency, inaccuracy, and maintenance burden.

This document proposes a simple, low-overhead mechanism for software vendors and maintainers to publish authoritative lifecycle data in a machine-readable format at a predictable location, using the Well-Known URI mechanism defined in [RFC8615].

The design goals are:

- * ***Authoritative***: lifecycle data is published directly by the party that controls the software's support policy — the vendor or maintainer.
- * ***Machine-readable***: a single, well-defined JSON format that any consuming tool can parse without scraping or inference.
- * ***Low barrier***: adding a static JSON file to an existing web presence requires minimal effort and no new infrastructure.
- * ***Incrementally deployable***: absence of the resource (HTTP 404) is a well-defined signal. Tools can fall back to existing methods when the resource is not present.
- * ***Composable***: the same schema is used for both the Well-Known URI (Appendix A (#appendix-a) describes the version-control companion) so tools that understand one format understand both.

1.1. Conventions and Definitions

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.

The following terms are used in this document:

***Software product*:** A discrete, identifiable unit of software that has a defined vendor or maintainer and a versioned release history.

***Version series*:** A major or minor version designation under which multiple patch releases may be issued (e.g., "3.2" covering 3.2.0, 3.2.1, 3.2.2, etc.).

***End-of-life (EOL)*:** A state in which a vendor or maintainer has permanently ceased issuing any updates, including security patches, for a given version series.

***Long-Term Support (LTS)*:** A designated version series for which the vendor or maintainer commits to an extended support window, typically longer than the standard release lifecycle.

***Consuming tool*:** Any software system that retrieves and processes a software-status.json resource, including security scanners, SCA tools, package managers, and monitoring systems.

2. The software-status.json Resource

2.1. Location and Retrieval

A software vendor or maintainer that wishes to publish lifecycle status information for software associated with a domain SHOULD publish a software-status.json resource at the Well-Known URI:

`https://example.com/.well-known/software-status.json`

The resource MUST be served over HTTPS. HTTP retrieval MAY be supported for compatibility but consuming tools SHOULD prefer HTTPS and SHOULD warn or fail if only HTTP is available.

The resource MUST be served with a Content-Type of application/json.

If no lifecycle information is available or the vendor has not adopted this convention, the server SHOULD return HTTP 404. Consuming tools MUST treat a 404 response as "no declaration available" and fall back to other methods. Consuming tools MUST NOT treat a 404 as evidence that the software is end-of-life.

2.2. Caching

The resource SHOULD include standard HTTP caching headers. A max-age of 604800 (seven days) is RECOMMENDED for resources that are infrequently updated. Consuming tools SHOULD respect Cache-Control and Expires headers.

In the absence of explicit caching headers, consuming tools SHOULD cache a successfully retrieved resource for no fewer than 24 hours and no more than 30 days.

3. The software-status.json Schema

The software-status.json resource is a JSON object ([RFC8259]) with the following structure.

3.1. Root Object

The root object MUST contain the following fields:

schema_version (string, REQUIRED): The version of this schema. This document defines version "1.0". Consuming tools that encounter an unrecognized schema version SHOULD process fields they recognize and ignore fields they do not.

name (string, REQUIRED): The human-readable name of the software product.

vendor (string, REQUIRED): The name of the organization or individual responsible for the software's support lifecycle.

versions (array, REQUIRED): An array of version entry objects as described in Section 4.2 (#version-entries). MUST contain at least one entry. SHOULD include all version series for which the vendor has a defined support position, including EOL versions.

The root object MAY contain the following fields:

homepage (string): A URI for the software product's primary website.

source (string): A URI for the software's canonical source repository.

package_identifiers (object): A map of package ecosystem names to the identifier used in that ecosystem. See Section 4.4 (#package-identifiers).

release_cycle_url (string): A URI for the vendor's official support

lifecycle or release policy page.

`last_updated` (string): An ISO 8601 date string indicating when this resource was last reviewed or updated (e.g., "2026-04-14").

Consuming tools MAY use this field to decide whether to re-fetch the resource regardless of caching policy.

3.2. Version Entries

Each object in the versions array represents one version series and MUST contain the following fields:

`version` (string, REQUIRED): The version series identifier. This MAY be a specific version number ("3.2"), a major version with wildcard ("2.x"), a named track ("LTS 22.04"), or any string that meaningfully identifies the series to the vendor's users.

`status` (string, REQUIRED): The current support status of this version series. MUST be one of the values defined in Section 4.3 (#status-values).

Each version entry MAY contain the following fields:

`release_date` (string): An ISO 8601 date on which this version series reached general availability.

`support_ends` (string or null): An ISO 8601 date on which support for this version series is scheduled to end. A value of null indicates no currently planned end date. SHOULD be present for all version series where a planned end date is known.

`eol_date` (string): An ISO 8601 date on which this version series reached end-of-life. This field records a past event; `support_ends` records a future one. SHOULD be present when status is "eol" and the date is known.

`lts` (boolean): true if this version series is designated as Long-Term Support. Defaults to false if absent.

`notes` (string): A human-readable note providing additional context about this version entry. Consuming tools MAY surface this to users but MUST NOT rely on its content for programmatic decisions.

3.3. Status Values

The status field MUST be one of the following values:

"active": The version series is fully supported. The vendor intends to issue bug fixes, security patches, and potentially new features.

"lts": The version series is designated Long-Term Support. The vendor commits to issuing security patches (and possibly bug fixes) for an extended period.

"security-only": The version series is no longer receiving general

bug fixes or new features, but security vulnerabilities are still being patched.

"eol": The version series has reached end-of-life. No further patches of any kind will be issued, including security patches.

"unmaintained": The software project has been abandoned by its original maintainer. No patches are expected, but the project has not been formally declared end-of-life.

Additional status values MAY be defined in future revisions of this document. Consuming tools that encounter an unrecognized status value SHOULD treat it as equivalent to "unknown" and note the unrecognized value in any user-facing output.

3.4. Package Identifiers

The optional `package_identifiers` object maps package ecosystem names to the string identifier used in that ecosystem. Known ecosystem names include, but are not limited to:

Key	Ecosystem	Example value
pypi	Python Package Index	"requests"
npm	npm registry	"express"
rubygems	RubyGems	"rails"
cargo	crates.io	"tokio"
nuget	NuGet	"Newtonsoft.Json"
maven	Maven Central	"org.apache:commons"
apt	Debian/Ubuntu APT	"openssl"
brew	Homebrew	"openssl"
docker	Docker Hub	"library/nginx"
github	GitHub repository	"owner/repo"

Table 1

A value of null for a given key indicates the software is not distributed through that ecosystem.

4. Processing Rules for Consuming Tools

4.1. Discovery

To retrieve lifecycle status for software associated with a known domain, a consuming tool SHOULD:

1. Construct the URI `https://{domain}/.well-known/software-status.json`
2. Issue an HTTP GET request with an `Accept: application/json` header
3. Treat a 2xx response containing valid JSON as a successful retrieval
4. Treat a 404 or 410 response as "no declaration" and fall back
5. Treat other error responses (5xx, timeouts) as transient failures; retry with exponential backoff before treating as unavailable

4.2. Version Matching

When matching an installed version against the versions array, consuming tools SHOULD use the following logic:

1. Attempt an exact match on the version field
2. If no exact match, attempt a prefix match: an installed version of 3.2.7 matches a version series of 3.2
3. If no prefix match, attempt a major-version match: 3.2.7 matches 3.x
4. If no match is found, treat the version's lifecycle status as unknown

When multiple entries match, consuming tools SHOULD prefer the most specific match (exact > prefix > major-version).

4.3. Conflict Resolution

When a consuming tool has lifecycle data for a product from multiple sources (e.g., a third-party database and a `software-status.json` resource), data retrieved from `software-status.json` SHOULD be treated as authoritative and given precedence, as it represents a declaration by the party responsible for the software's support policy.

5. Security Considerations

5.1. Authenticity

The `software-status.json` resource is served over HTTPS and inherits the authenticity guarantees of the TLS connection. A consuming tool that retrieves the resource over HTTPS from the vendor's canonical domain can be reasonably confident that the content reflects the vendor's intent.

Consuming tools **MUST NOT** retrieve this resource over plain HTTP and treat it as authoritative. Consuming tools **SHOULD** validate that the hostname of the retrieval URI matches the domain associated with the software in question.

5.2. Denial of Service

A consuming tool that retrieves `software-status.json` resources at scale **SHOULD** implement rate limiting, caching, and exponential backoff to avoid placing undue load on vendor infrastructure.

5.3. Misleading Declarations

This mechanism relies on vendors publishing accurate information. A vendor could theoretically publish a `software-status.json` that misrepresents the support status of a product — for example, declaring an EOL product as "active" to discourage users from migrating. Consuming tools **SHOULD** note the source of lifecycle determinations in user-facing output so that users can evaluate the claim in context.

Consuming tools that maintain independent lifecycle databases **MAY** flag discrepancies between their own records and a vendor-published `software-status.json` for human review.

5.4. Privacy

Retrieving `software-status.json` from a vendor's domain may reveal to that vendor that a consuming tool is checking lifecycle status, potentially disclosing information about what software is deployed in an organization. Consuming tools operating in sensitive environments **SHOULD** consider whether direct retrieval is appropriate or whether a privacy-preserving intermediary (such as a shared cache or mirroring service) should be used.

6. IANA Considerations

6.1. Well-Known URI Registration

IANA is requested to register the following Well-Known URI in the "Well-Known URIs" registry established by [RFC8615]:

URI suffix: software-status.json

Change controller: IETF

Specification document(s): This document

Related information: This URI returns a JSON document describing the software lifecycle status for products associated with the host. The format is defined in Section 4 (#the-software-statusjson-schema) of this document.

7. References

7.1. 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>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <<https://www.rfc-editor.org/info/rfc8615>>.

7.2. Informative References

- [RFC9116] Foudil, E. and Y. Shafranovich, "A File Format to Aid in Security Vulnerability Disclosure", RFC 9116, DOI 10.17487/RFC9116, April 2022, <<https://www.rfc-editor.org/info/rfc9116>>.

Appendix A. Examples

A.1. Minimal Resource

```
{
  "schema_version": "1.0",
  "name": "ExampleDB",
  "vendor": "Example Software Inc.",
  "versions": [
    {
      "version": "5.0",
      "status": "active",
      "support_ends": null
    }
  ]
}
```

A.2. Multi-Version Resource with EOL History

```
{
  "schema_version": "1.0",
  "name": "ExampleDB",
  "vendor": "Example Software Inc.",
  "homepage": "https://example.com/db",
  "package_identifiers": {
    "apt": "exampledb-server",
    "docker": "exampleinc/exampledb"
  },
  "versions": [
    {
      "version": "5.0",
      "release_date": "2025-03-01",
      "status": "active",
      "support_ends": null,
      "lts": false
    },
    {
      "version": "4.2",
      "release_date": "2023-06-15",
      "status": "lts",
      "support_ends": "2027-06-15",
      "lts": true,
      "notes": "LTS release — security and critical bug fixes through June 2027"
    },
    {
      "version": "4.0",
      "release_date": "2022-01-10",
      "status": "security-only",
      "support_ends": "2026-01-10",
      "lts": false
    },
    {
      "version": "3.x",
      "release_date": "2019-04-01",
      "status": "eol",
      "eol_date": "2024-04-01",
      "support_ends": "2024-04-01",
      "lts": false
    }
  ],
  "release_cycle_url": "https://example.com/support-policy",
  "last_updated": "2026-04-14"
}
```

Appendix B. Companion Convention: Repository-Hosted Lifecycle Status

This appendix describes a companion convention for software projects hosted on version-control platforms (such as GitHub, GitLab, or Gitea). This convention is informative and does not define a standards-track mechanism; it is documented here because it uses the same schema as the Well-Known URI defined in this document and is intended to be consumed by the same tools.

B.1. Motivation

Many software projects are distributed as source code and do not have a corresponding web domain from which a Well-Known URI could be served. Open-source libraries, developer tools, and components published to package registries often have a repository URL as their primary identity, not a product domain.

For these projects, a file committed to a predictable location within the repository serves the same purpose as the Well-Known URI.

B.2. Location

The resource SHOULD be placed at:

```
{repository-root}/.github/software-status.json
```

The `.github/` directory is an established convention on major version-control platforms for repository-level metadata. This location is intentionally analogous to `.github/SECURITY.md`, `.github/FUNDING.yml`, and similar files.

B.3. Discovery

Consuming tools that resolve a software package to a GitHub-hosted repository SHOULD check for the presence of this file at the raw content URL:

```
https://raw.githubusercontent.com/{owner}/{repo}/HEAD/.github/software-status.json
```

A 404 response MUST be treated as "no declaration available". A 200 response with valid JSON MUST be processed according to the schema defined in Section 4 (`#the-software-statusjson-schema`).

B.4. Schema

The `.github/software-status.json` resource uses the same JSON schema as defined in Section 4 (`#the-software-statusjson-schema`), with one addition: the `source` field at the root object SHOULD be set to the canonical repository URI.

B.5. Precedence

When a project has both a Well-Known URI resource and a repository-hosted resource, consuming tools SHOULD prefer the Well-Known URI, as it is served from infrastructure explicitly controlled by the vendor and subject to the HTTPS authenticity guarantees described in Section 6 (`#security-considerations`).

B.6. Example

```
{
  "schema_version": "1.0",
  "name": "example-lib",
  "vendor": "Example Org",
  "source": "https://github.com/example/example-lib",
  "package_identifiers": {
    "pypi": "example-lib",
    "npm": null
  },
  "versions": [
    {
      "version": "2.0",
      "release_date": "2025-01-01",
      "status": "active",
      "support_ends": null
    },
    {
      "version": "1.x",
      "release_date": "2022-06-01",
      "status": "eol",
      "eol_date": "2025-06-01"
    }
  ],
  "last_updated": "2026-04-14"
}
```

Appendix C. Related Work

endoflife.date: A community-maintained database of software lifecycle information, available at <https://endoflife.date>. Provides an API for querying EOL dates for approximately 400 products. This proposal is complementary: `endoflife.date` is a centralized catalog; `software-status.json` is a decentralized, vendor-authoritative mechanism. Consuming tools should use both.

security.txt ([RFC9116]): Defines a Well-Known URI (`/.well-known/security.txt`) at which organizations publish security contact information. This proposal follows the same pattern for lifecycle data.

Software Bill of Materials (SBOM): Formats including CycloneDX and SPDX describe the components present in a software artifact. SBOMs identify `_what_` is present; `software-status.json` declares the lifecycle status of `_what_` is present. The two are complementary: an SBOM tool that resolves component origins can use `software-status.json` to annotate each component with its current support status.

NIST Secure Software Development Framework (SSDF): The SSDF (NIST SP 800-218) recommends that organizations track the EOL status of software components they depend on. This proposal provides a standardized data source for that activity.

Appendix D. Acknowledgements

The author thanks the maintainers of `endoflife.date` for their sustained effort in curating software lifecycle data, which demonstrated both the value of this information and the limitations of a purely centralized approach. The design of this proposal was informed by operational experience building S3C-Tool (<https://askmcconnell.com/s3c/> (<https://askmcconnell.com/s3c/>)), a software supply chain security tool that performs EOL assessment at scale.

Author's Address

Jim McConnell
Ask McConnell, LLC
Email: jim@askmcconnell.com
URI: <https://askmcconnell.com>