Individual Submission
Internet-Draft
Intended status: Informational
Expires: 27 October 2026

J. McConnell
Ask McConnell, LLC

25 April 2026

A Well-Known URI for Software Lifecycle Status
draft-mcconnell-software-status-wellknown-02

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 revision (-02) extends the schema to support multi-product vendors — organizations that ship multiple distinct products or SKUs under a single domain. A new optional products array at the root of the resource allows a single software-status.json endpoint to serve lifecycle declarations for an entire product catalog, including firmware-based devices such as routers, switches, and IoT appliances. This extension is fully backward compatible: existing single-product resources require no modification.

This document also describes, in Appendix B (#appendix-b), 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/.

Copyright Notice

Table of Contents

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 problem is acutely pronounced for firmware-based network devices — routers, switches, wireless access points, firewalls, and IoT appliances. A single network equipment vendor may ship hundreds of distinct hardware SKUs across multiple product families, each running a different firmware version with different embedded component versions and different end-of-support dates. Supply chain security tools that scan such devices can identify installed software components but have no reliable, vendor-authoritative source from which to determine the support status of the firmware itself or the components it embeds.

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 -02 revision extends the schema to accommodate vendors with multiple products or SKUs under a single domain.

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 and the version-control companion described in Appendix B (#appendix-b), so tools that understand one format understand both.

* *Multi-product capable*: a single endpoint MAY describe an entire vendor product catalog, enabling network equipment manufacturers and IoT device vendors to serve lifecycle status for all of their SKUs from one location.

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.

SKU: Stock Keeping Unit. In this document, used to refer to a distinct hardware model or product variant shipped by a vendor (e.g., a specific router model number).

Firmware: Software that is embedded in a hardware device and controls its operation. Firmware versions are typically tied to specific hardware models or product families and are updated by the device vendor.

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.

Discontinued: A hardware product that is no longer manufactured or sold. A discontinued product MAY still receive firmware security updates for existing deployments; this is distinct from end-of-life.

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.

Vendors with large product catalogs (see Section 4.5 (#multi-product-resources)) whose software-status.json resource is updated frequently SHOULD set a shorter max-age (e.g., 86400, one day) and include an ETag to allow consuming tools to perform conditional GET requests.

3. The software-status.json Schema

The software-status.json resource is a JSON object ([RFC8259]) with the following structure. The resource operates in one of two modes, determined by the presence of the products field:

* *Single-product mode* (default): the root object describes one product. This is the mode defined in -00 and -01 and is unchanged.

* *Multi-product mode*: the root object describes a vendor catalog. A products array contains one entry per product or SKU. See Section 4.5 (#multi-product-resources).

The two modes are fully backward compatible. A consuming tool that does not implement multi-product support SHOULD ignore the products field if present. A consuming tool that implements multi-product support MUST check for the products field before processing root-level name and versions fields.

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.

vendor (string, REQUIRED): The name of the organization or individual responsible for the software's support lifecycle. In multi-product mode, this is the vendor organization name. In single-product mode, this is the product vendor.

In *single-product mode* (when products is absent), the root object MUST also contain:

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

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 in either mode:

specification (string): A URI identifying the specification that defines this schema. When present, consuming tools MAY use this field to verify schema compatibility or route processing to a version-appropriate parser. For resources conforming to this document, the value SHOULD be the IETF Datatracker URI for this Internet-Draft or its successor RFC (e.g., "https://datatracker.ietf.org/doc/draft-mcconnell-software-status-wellknown/"). This field is self-documenting: it allows a human or tool encountering an unfamiliar software-status.json resource to locate the governing specification without prior knowledge.

homepage (string): A URI for the vendor's primary website. In single-product mode, MAY refer to the product page specifically.

source (string): A URI for the software's canonical source repository. Typically absent in multi-product mode unless the vendor publishes a monorepo.

package_identifiers (object): In single-product mode: a map of package ecosystem names to the identifier used in that ecosystem. See Section 4.4 (#package-identifiers). SHOULD NOT be present in multi-product mode; use per-product package_identifiers in each product entry instead.

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., "2026-04-25"). Consuming tools MAY use this field to decide whether to re-fetch the resource regardless of caching policy.

products (array): When present, indicates multi-product mode. An

array of product entry objects as described in Section 4.5 (#multi-product-resources). When this field is present, consuming tools MUST use product entries for lifecycle lookups and SHOULD NOT use root-level name and versions for product matching.

3.2. Version Entries

Each object in the versions array (whether at the root level in single-product mode, or within a product entry in multi-product mode) 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. For firmware, this SHOULD be the firmware version string as displayed in the device’s management interface.

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.

3.5. Multi-Product Resources

3.5.1. Overview

Many vendors ship multiple distinct products or hardware SKUs under a single domain. A network equipment manufacturer may offer dozens of router models, switches, wireless access points, and security appliances — each with its own firmware version history and support lifecycle. An operating system vendor may ship a desktop edition, server edition, and embedded edition, each with independent support windows.

The products array at the root of the resource allows a single software-status.json endpoint to serve lifecycle declarations for an entire product catalog. This eliminates the need for per-product endpoints and allows consuming tools to retrieve all lifecycle data for a vendor in a single HTTP request.

3.5.2. Product Entries

Each object in the products array represents one product or SKU and MUST contain the following fields:

product_id (string, REQUIRED): A stable, machine-readable identifier for this product, unique within this vendor's products array. SHOULD use only ASCII alphanumeric characters, hyphens, and underscores (e.g., "UDM-PRO", "nginx", "RT-AX88U"). MUST NOT change between revisions of the software-status.json resource once published; consuming tools MAY use this value as a stable key for caching and database storage.

name (string, REQUIRED): The human-readable name of this product or SKU.

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.

Each product entry MAY contain the following fields:

sku (string): The vendor's official SKU or model number for this product (e.g., "UDM-PRO", "EAP670"). MAY differ from product_id if the product_id uses a normalized form. Consuming tools MAY use this field for fuzzy matching against hardware model strings discovered during device inventory.

product_type (string): The type of product. SHOULD be one of the

following values: "software" (a standalone software application), "firmware" (software embedded in a hardware device), "os" (a general-purpose operating system or OS distribution), "library" (a software library or SDK), "container" (a container image or OCI artifact), "hardware-firmware" (a hardware SKU whose primary versioned artifact is its firmware). Additional values MAY be used; consuming tools that encounter an unrecognized value SHOULD treat it as "software".

- platform (string): The hardware platform or CPU architecture for which this product is built (e.g., "arm64", "x86_64", "mips"). For products that support multiple architectures, this field MAY be omitted or set to a comma-separated list.
- discontinued (boolean): true if the hardware product has been discontinued by the vendor (i.e., is no longer manufactured or sold). Defaults to false if absent. A discontinued product MAY still be in an "active" or "security-only" support state for its firmware — discontinuation of hardware manufacturing does not imply end-of-life for security patches. Consuming tools SHOULD surface the discontinued flag alongside the firmware support status to give operators a complete picture.
- homepage (string): A URI for this product’s primary webpage.
- release_cycle_url (string): A URI for the support lifecycle policy specific to this product. When present, takes precedence over the root-level release_cycle_url for this product.
- package_identifiers (object): A map of package ecosystem names to identifiers, as defined in Section 4.4 (#package-identifiers). For firmware products, the github key MAY reference the vendor’s open-source firmware repository if applicable.
- notes (string): A human-readable note about this product. Consuming tools MAY surface this to users but MUST NOT rely on its content for programmatic decisions.

3.5.3. Product Discovery

When a consuming tool needs to determine the lifecycle status of a specific product from a multi-product resource, it SHOULD use the following matching strategy, in order of preference:

1. Exact match on product_id (case-insensitive)
2. Exact match on sku (case-insensitive)
3. Exact match on name (case-insensitive)
4. Partial/fuzzy match on name or sku (implementation-defined)

When no match is found, consuming tools MUST treat the product’s lifecycle status as unknown and MUST NOT infer EOL status from the absence of a matching entry.

Consuming tools SHOULD expose matched product_id values in their output to allow operators to verify that the correct product entry was selected, particularly when fuzzy matching was used.

3.5.4. Large Catalogs

Vendors with very large product catalogs (hundreds or thousands of SKUs) may find it impractical to serve the complete catalog in a single JSON resource. For such vendors, this document does not prescribe a pagination mechanism. However, the following approaches are RECOMMENDED:

* Serve the complete catalog as a single resource and rely on HTTP compression (gzip, brotli) to manage response size. A catalog of 500 SKUs with three version entries each is typically well under 500 KB uncompressed and under 50 KB compressed.

* Use last_updated at both the root level and per-product entry level to allow consuming tools to detect staleness without re-fetching the complete resource.

* Include an ETag HTTP response header to enable conditional GET requests.

Future revisions of this document MAY define a pagination or linking mechanism for extremely large catalogs. Consuming tools SHOULD be prepared to handle resources of arbitrary size.

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
6. Check for the presence of a products field to determine whether to use single-product or multi-product processing (see Section 4.5 (#multi-product-resources))

4.2. Version Matching

When matching an installed version against the versions array (whether at the root level or within a product entry), 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.

For multi-product resources, a consuming tool that needs lifecycle status for multiple products from the same vendor SHOULD retrieve the resource once and process all required products from the cached response, rather than issuing repeated requests.

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, or omitting a product from the products array to avoid acknowledging its EOL status. 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.

The absence of a product from a vendor’s products array MUST NOT be interpreted as confirmation that the product is end-of-life or unsupported. It indicates only that the vendor has not published a declaration for that product via this mechanism.

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.

For multi-product resources, a single retrieval reveals interest in the vendor’s products generally, but does not disclose which specific SKU or version is deployed. This is a privacy improvement over per-product endpoints, which would reveal the specific product being queried.

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. The resource MAY describe a single product or a catalog of multiple products or hardware SKUs.

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>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <https://www.rfc-editor.org/info/rfc8520>.

Appendix A. Examples

A.1. Minimal Single-Product Resource

```
   {
     "schema_version": "1.0",
     "specification": "https://datatracker.ietf.org/doc/draft-mcconnell-software-status-w
ellknown/",
     "name": "ExampleDB",
     "vendor": "Example Software Inc.",
     "versions": [
       {
         "version": "5.0",
         "status": "active",
         "support_ends": null
       }
     ]
   }
```

A.2. Multi-Version Single-Product 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": "2026-04-25"
}
```

A.3. Multi-Product Resource: Network Equipment Vendor

The following example illustrates a network equipment vendor publishing lifecycle status for multiple hardware SKUs from a single endpoint. This pattern is appropriate for vendors such as router, switch, or wireless access point manufacturers that ship dozens to hundreds of distinct hardware models, each with its own firmware version history.

Note the use of product_type: "hardware-firmware" and the discontinued flag, which allows consuming tools to distinguish between hardware that is no longer sold and firmware that is no longer patched — two related but distinct conditions that affect operator decisions differently.


```
{
  "schema_version": "1.0",
  "specification": "https://datatracker.ietf.org/doc/draft-mcconnell-software-status-w
ellknown/",
  "vendor": "Example Networks Inc.",
  "homepage": "https://network-vendor.example.com",
  "release_cycle_url": "https://network-vendor.example.com/end-of-life",
  "last_updated": "2026-04-25",
  "products": [
    {
      "product_id": "ENI-GW-PRO",
      "name": "Example Gateway Pro",
      "sku": "ENI-GW-PRO",
      "product_type": "hardware-firmware",
      "platform": "arm64",
      "discontinued": false,
      "homepage": "https://network-vendor.example.com/products/gw-pro",
      "versions": [
        {
          "version": "5.0",
          "release_date": "2025-09-01",
          "status": "active",
          "support_ends": null,
          "notes": "Current production firmware"
        },
        {
          "version": "4.x",
          "release_date": "2023-03-15",
          "status": "security-only",
          "support_ends": "2026-09-01"
        },
        {
          "version": "3.x",
          "release_date": "2021-01-10",
```

```
            "status": "eol",
            "eol_date": "2025-01-10"
          }
        ]
      },
      {
        "product_id": "ENI-GW-SE",
        "name": "Example Gateway SE",
        "sku": "ENI-GW-SE",
        "product_type": "hardware-firmware",
        "platform": "arm64",
        "discontinued": false,
        "versions": [
          {
            "version": "5.0",
            "release_date": "2025-09-01",
            "status": "active",
            "support_ends": null
          },
          {
            "version": "4.x",
            "release_date": "2023-06-01",
            "status": "security-only",
            "support_ends": "2026-09-01"
          }
        ]
      },
      {
        "product_id": "ENI-AP-AX",
        "name": "Example Access Point AX",
        "sku": "ENI-AP-AX",
        "product_type": "hardware-firmware",
        "platform": "mips",
        "discontinued": true,
        "notes": "Hardware discontinued March 2025; firmware security patches continue t
hrough 2027",
        "versions": [
          {
            "version": "6.x",
            "release_date": "2022-11-01",
            "status": "security-only",
            "support_ends": "2027-03-01"
          },
          {
            "version": "5.x",
            "release_date": "2020-06-01",
            "status": "eol",
            "eol_date": "2024-06-01"
          }
```

```
        ]
      },
      {
        "product_id": "ENI-SW-24",
        "name": "Example Switch 24-Port",
        "sku": "ENI-SW-24",
        "product_type": "hardware-firmware",
        "platform": "arm32",
        "discontinued": false,
        "versions": [
          {
            "version": "3.x",
            "release_date": "2024-02-01",
            "status": "active",
            "support_ends": null
          }
        ]
      }
    ]
  }
```

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. The products array MAY be used in repository-hosted resources for projects that maintain multiple independently versioned components within a single repository (monorepos).

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": "2026-04-25"
}
```

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.

Manufacturer Usage Description (MUD) ([RFC8520]): Defines a mechanism by which IoT devices can describe their intended network behavior to network operators, via a MUD URL served from the device or supplied out-of-band. The multi-product schema defined in this document is complementary to MUD: MUD describes what network behaviors a device should exhibit; software-status.json describes the lifecycle status of the software and firmware running on that device. A MUD URL (ietf-mud:mud-url) and a software-status.json endpoint may

be published by the same vendor domain. Future work may explore referencing a product's software-status.json entry from within a MUD file to allow network management systems to correlate device identity, network policy, and firmware lifecycle status in a single workflow.

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. For firmware devices, an SBOM that enumerates embedded open-source components combined with per-component software-status.json lookups provides the most complete picture of a device's security posture — a pattern directly applicable to network equipment with multiple embedded OSS components (e.g., Linux kernel, OpenSSL, curl, dnsmasq) whose lifecycle status may differ from the firmware version's overall 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.

SCITT (Supply Chain Integrity, Transparency and Trust): The IETF SCITT working group is developing a framework for supply chain transparency using signed statements and transparent registries. The software-status.json mechanism is complementary: it provides a lightweight, vendor-self-published lifecycle declaration that does not require a transparency registry infrastructure, while SCITT provides stronger integrity guarantees through cryptographic signatures and append-only logs. Future work may explore a SCITT-signed envelope for software-status.json content to provide non-repudiation and tamper evidence for vendor lifecycle declarations.

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 and CVE assessment at scale across a wide range of software inventories including firmware-based network devices.

The author thanks members of the SCITT mailing list for feedback on -01, particularly on the topics of multi-product vendor schema design, IoT and firmware applicability, and the relationship between this mechanism and existing IETF work including [RFC8520].

Author’s Address

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