Stream: Internet Engineering Task Force (IETF)
RFC: 9393
Category: Standards Track
Published: June 2023
ISSN: 2070-1721
Authors: H. Birkholz, *Fraunhofer SIT*; J. Fitzgerald-McKay, *National Security Agency*; C. Schmidt, *The MITRE Corporation*; D. Waltermire, *NIST*

RFC 9393
Concise Software Identification Tags

Abstract

ISO/IEC 19770-2:2015 Software Identification (SWID) tags provide an extensible XML-based structure to identify and describe individual software components, patches, and installation bundles. SWID tag representations can be too large for devices with network and storage constraints. This document defines a concise representation of SWID tags: Concise SWID (CoSWID) tags. CoSWID supports a set of semantics and features that are similar to those for SWID tags, as well as new semantics that allow CoSWIDs to describe additional types of information, all in a more memory-efficient format.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9393.

Copyright Notice

Table of Contents

Authors' Addresses

1. Introduction

SWID tags, as defined in ISO-19770-2:2015 [SWID], provide a standardized XML-based record format that identifies and describes a specific release of software, a patch, or an installation bundle, which are referred to as software components in this document. Different software components, and even different releases of a particular software component, each have a different SWID tag record associated with them. SWID tags are meant to be flexible and able to express a broad set of metadata about a software component.

SWID tags are used to support a number of processes, including but not limited to:

- Software Inventory Management, representing a part of a Software Asset Management process [SAM], which requires an accurate list of discernible deployed software components.
- Vulnerability Assessment, which requires a semantic link between standardized vulnerability descriptions and software components installed on IT assets [X.1520].
- Remote Attestation, which requires a link between Reference Integrity Manifests (RIMs) and Attester-produced event logs that complement attestation evidence [RFC9334].

While there are very few required fields in SWID tags, there are many optional fields that support different uses. A SWID tag consisting of only required fields might be a few hundred bytes in size; however, a tag containing many of the optional fields can be many orders of magnitude larger. Thus, real-world instances of SWID tags can be fairly large, and the communication of SWID tags in usage scenarios, such as those described earlier, can cause a large amount of data to be transported. This can be larger than acceptable for constrained devices and networks. Concise SWID (CoSWID) tags significantly reduce the amount of data transported as compared to a typical SWID tag through the use of the Concise Binary Object Representation (CBOR) [RFC8949].

Size comparisons between XML SWID and CoSWID mainly depend on domain-specific applications and the complexity of attributes used in instances. While the values stored in CoSWID are often unchanged and therefore not reduced in size compared to an XML SWID, the scaffolding that the CoSWID encoding represents is significantly smaller by taking up 10 percent or less in size. This effect is visible in representation sizes, which in early experiments benefited from a 50 percent to 85 percent reduction in generic usage scenarios. Additional size reduction is enabled with respect to the memory footprint of XML parsing/validation.

In a CoSWID, the human-readable labels of SWID data items are replaced with more concise integer labels (indices). This approach allows SWID and CoSWID to share a common implicit information model, with CoSWID providing an alternate data model [RFC3444]. While SWID and CoSWID are intended to share the same implicit information model, this specification does not define this information model or a mapping between the two data formats. While an attempt to

align SWID and CoSWID tags has been made here, future revisions of ISO/IEC 19770-2:2015 or this specification might cause this implicit information model to diverge, since these specifications are maintained by different standards groups.

The use of CBOR to express SWID information in CoSWID tags allows both CoSWID and SWID tags to be part of an enterprise security solution for a wider range of endpoints and environments.

1.1. The SWID and CoSWID Tag Lifecycle

In addition to defining the format of a SWID tag record, ISO/IEC 19770-2:2015 defines requirements concerning the SWID tag lifecycle. Specifically, when a software component is installed on an endpoint, that software component's SWID tag is also installed. Likewise, when the software component is uninstalled or replaced, the SWID tag is deleted or replaced, as appropriate. As a result, ISO/IEC 19770-2:2015 describes a system wherein there is a correspondence between the set of installed software components on an endpoint and the presence of the corresponding SWID tags for these components on that endpoint. CoSWIDs share the same lifecycle requirements as a SWID tag.

The SWID specification and supporting guidance provided in NIST Internal Report (NISTIR) 8060 ("Guidelines for the Creation of Interoperable Software Identification (SWID) Tags") [SWID-GUIDANCE] define four types of SWID tags: primary, patch, corpus, and supplemental. The following text is paraphrased from these sources.

Primary Tag: A SWID or CoSWID tag that identifies and describes an installed software component on an endpoint. A primary tag is intended to be installed on an endpoint along with the corresponding software component.

Patch Tag: A SWID or CoSWID tag that identifies and describes an installed patch that has made incremental changes to a software component installed on an endpoint. A patch tag is intended to be installed on an endpoint along with the corresponding software component patch.

Corpus Tag: A SWID or CoSWID tag that identifies and describes an installable software component in its pre-installation state. A corpus tag can be used to represent metadata about an installation package or installer for a software component, a software update, or a patch.

Supplemental Tag: A SWID or CoSWID tag that allows additional information to be associated with a referenced SWID tag. This allows tools and users to record their own metadata about a software component without modifying CoSWID primary or patch tags created by a software provider.

The type of a tag is determined by specific data elements, which are discussed in Section 3. Section 3 also provides normative language for CoSWID semantics that implement this lifecycle. The following information helps to explain how these semantics apply to the use of a CoSWID tag.

Corpus, primary, and patch tags have similar functions in that they describe the existence and/or presence of different types of software components (e.g., software installers, software installations, software patches) and, potentially, different states of these software components. Supplemental tags have the same structure as other tags but are used to provide information not contained in the referenced corpus, primary, and patch tags. All four tag types come into play at various points in the software lifecycle and support software management processes that depend on the ability to accurately determine where each software component is in its lifecycle.

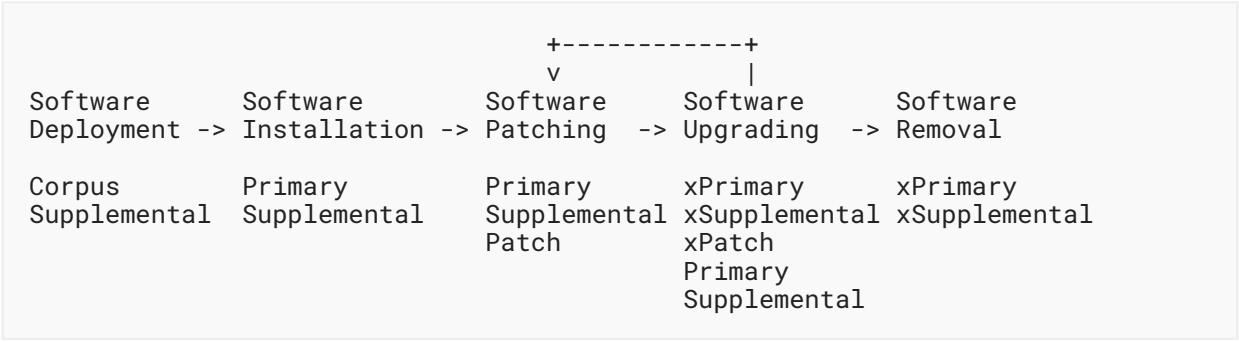

Figure 1: Use of Tag Types in the Software Lifecycle

Figure 1 illustrates the steps in the software lifecycle and the relationships among those lifecycle events supported by the four types of SWID and CoSWID tags. A detailed description of the four tag types is provided in Section 2.3. The figure identifies the types of tags that are used in each lifecycle event.

There are many ways in which software tags might be managed for the host the software is installed on. For example, software tags could be made available on the host or to an external software manager when storage is limited on the host.

In these cases, the host or external software manager is responsible for management of the tags, including deployment and removal of the tags as indicated by the above lifecycle. Tags are deployed, and previously deployed tags are typically removed (indicated by an "x" prefix) at each lifecycle stage as follows:

Software Deployment: Before the software component is installed (i.e., pre-installation), and while the product is being deployed, a corpus tag provides information about the installation files and distribution media (e.g., CD/DVD, distribution package).

Corpus tags are not actually deployed on the target system but are intended to support deployment procedures and their dependencies at install time, such as to verify the installation media.

Software Installation: A primary tag will be installed with the software component (or subsequently created) to uniquely identify and describe the software component. Supplemental tags are created to augment primary tags with additional site-specific or

extended information. While not illustrated in the figure, patch tags can also be installed during software installation to provide information about software fixes deployed along with the base software installation.

Software Patching: When a patch is applied to the software component, a new patch tag is provided, supplying details about the patch and its dependencies. While not illustrated in the figure, a corpus tag can also provide information about the patch installer and patching dependencies that need to be installed before the patch.

Software Upgrading: As a software component is upgraded to a new version, new primary and supplemental tags replace existing tags, enabling timely and accurate tracking of updates to software inventory. While not illustrated in the figure, a corpus tag can also provide information about the upgrade installer and dependencies that need to be installed before the upgrade.

> Note: In the context of software tagging, software patching and updating differ in an important way. When installing a patch, a set of file modifications are made to pre-installed software; these modifications do not alter the version number or the descriptive metadata of an installed software component. An update can also make a set of file modifications; in that case, the version number or the descriptive metadata of an installed software component is changed.

Software Removal: Upon removal of the software component, relevant SWID tags are removed. This removal event can trigger timely updates to software inventory reflecting the removal of the product and any associated patch or supplemental tags.

As illustrated in the figure, supplemental tags can be associated with any corpus, primary, or patch tag to provide additional metadata about an installer, installed software, or installed patch, respectively.

Understanding the use of CoSWIDs in the software lifecycle provides a basis for understanding the information provided in a CoSWID and the associated semantics of this information. Each different SWID and CoSWID tag type provides different sets of information. For example, a "corpus tag" is used to describe a software component's installation image on an installation medium, while a "patch tag" is meant to describe a patch that modifies some other software component.

1.2. Concise SWID Format

This document defines the CoSWID tag format, which is based on CBOR. CBOR-based CoSWID tags offer a more concise representation of SWID information as compared to the XML-based SWID tag representation in ISO-19770-2:2015. The structure of a CoSWID is described via the Concise Data Definition Language (CDDL) [RFC8610]. The resulting CoSWID data definition is aligned with the information able to be expressed with the XML Schema definition of

ISO-19770-2:2015 [SWID]. This alignment allows both SWID and CoSWID tags to represent a common set of software component information and allows CoSWID tags to support the same uses as a SWID tag.

The vocabulary (i.e., the CDDL names of the types and members used in the CoSWID CDDL specification) is mapped to more concise labels represented as small integer values (indices). The names used in the CDDL specification and the mapping to the CBOR representation using integer indices are based on the vocabulary of the XML attribute and element names defined in ISO/IEC 19770-2:2015.

1.3. Requirements Notation

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.

2. Concise SWID Data Definition

The following describes the general rules and processes for encoding data using CDDL representation. Prior familiarity with CBOR and CDDL concepts will be helpful in understanding this CoSWID specification.

This section describes the conventions by which a CoSWID is represented in the CDDL structure. The CamelCase notation [CamelCase] used in the XML Schema definition is changed to a hyphen-separated notation [KebabCase] (e.g., "ResourceCollection" is named "resource-collection") in the CoSWID CDDL specification. This deviation from the original notation used in the XML representation reduces ambiguity when referencing certain attributes in corresponding textual descriptions. An attribute referred to by its name in CamelCase notation explicitly relates to XML SWID tags; an attribute referred to by its name in KebabCase notation explicitly relates to CBOR CoSWID tags. This approach simplifies the composition of further work that will reference both XML SWID and CBOR CoSWID documents.

In most cases, mapping attribute names between SWID and CoSWID can be done automatically by converting between CamelCase and KebabCase attribute names. However, some CoSWID CDDL attribute names show greater variation relative to their corresponding SWID XML Schema attributes. This is done when the change improves clarity in the CoSWID specification. For example, the "name" and "version" SWID fields correspond to the "software-name" and "software-version" CoSWID fields, respectively. As such, it is not always possible to mechanically translate between corresponding attribute names in the two formats. In such cases, a manual mapping will need to be used. XPath expressions [W3C.REC-xpath20-20101214] need to use SWID names; see Section 5.2.

The 57 human-readable text labels of the CDDL-based CoSWID vocabulary are mapped to integer indices via a block of rules at the bottom of the definition. This allows a more concise integer-based form to be stored or transported, as compared to the less efficient text-based form of the original vocabulary.

Through the use of CDDL-based integer labels, CoSWID allows for future expansion in subsequent revisions of this specification and through extensions (see Section 2.2). New constructs can be associated with a new integer index. A deprecated construct can be replaced by a new construct with a new integer index. An implementation can use these integer indices to identify the construct to parse. The "CoSWID Items" registry, defined in Section 6.1, is used to ensure that new constructs are assigned a unique index value. This approach avoids the need to have an explicit CoSWID version.

In a number of places, the value encoding admits both integer values and text strings. The integer values are defined in a registry specific to the kind of value; the text values are not intended for interchange and are exclusively meant for private use as defined in Section 6.2.2. Encoders **SHOULD NOT** use string values based on the names registered in the registry, as these values are less concise than their index value equivalent; a decoder **MUST**, however, be prepared to accept text strings that are not specified in this document (and ignore the construct if a string is unknown). In the rest of this document, we call this an "integer label with text escape".

The root of the CDDL specification provided by this document is the rule `coswid` (as defined in Section 8):

```
start = coswid
```

In CBOR, an array is encoded using bytes that identify the array, and the array's length or stop point (see [RFC8949]). To make items that support one or more values, the following CDDL notation is used.

```
_name_ = (_label_ => _data_ / [ 2* _data_ ])
```

The CDDL rule above allows either a single data item or an array of two or more data values to be provided. When a singleton data value is provided, the CBOR markers for the array, array length, and stop point are not needed, saving bytes. When two or more data values are provided, these values are encoded as an array. This modeling pattern is used frequently in the CoSWID CDDL specification to allow for more efficient encoding of singleton values.

Usage of this construct can be simplified using

```
one-or-more<T> = T / [ 2* T ]
```

simplifying the above example to

```
_name_ = (_label_ => one-or-more<_data_>)
```

The following subsections describe the different parts of the CoSWID model.

2.1. Character Encoding

The CDDL "text" type is represented in CBOR as a major type 3, which represents a string of Unicode characters that are encoded as UTF-8 [RFC3629] (see Section 3.1 of [RFC8949]). Thus, both SWID and CoSWID use UTF-8 for the encoding of characters in text strings.

To ensure that UTF-8 character strings are able to be encoded/decoded and exchanged interoperably, text strings in CoSWID **MUST** be encoded in a way that is consistent with the Net-Unicode definition provided in [RFC5198].

All names registered with IANA according to the requirements in Section 6.2 also **MUST** be valid according to the XML Schema NMTOKEN data type (see [W3C.REC-xmlschema-2-20041028], Section 3.3.4) to ensure compatibility with the SWID specification where these names are used.

2.2. Concise SWID Extensions

The CoSWID specification contains two features that are not included in the SWID specification on which it is based. These features are:

- The explicit definition of types for some attributes in the ISO-19770-2:2015 XML representation that are typically represented by the any-attribute item in the SWID model. These are covered in Section 2.5.
- The inclusion of extension points in the CoSWID specification using CDDL sockets (see Section 3.9 of [RFC8610]). The use of CDDL sockets allows for well-formed extensions to be defined in supplementary CDDL descriptions that support additional uses of CoSWID tags that go beyond the original scope of ISO-19770-2:2015 tags.

The following CDDL sockets (extension points) are defined in this document; they allow the addition of new information structures to their respective CDDL groups.

Map Name	CDDL Socket	Defined in
concise-swid-tag	$$coswid-extension	Section 2.3
entity-entry	$$entity-extension	Section 2.6
link-entry	$$link-extension	Section 2.7
software-meta-entry	$$software-meta-extension	Section 2.8
resource-collection	$$resource-collection-extension	Section 2.9.2
file-entry	$$file-extension	Section 2.9.2
directory-entry	$$directory-extension	Section 2.9.2
process-entry	$$process-extension	Section 2.9.2

Map Name	CDDL Socket	Defined in
resource-entry	$$resource-extension	Section 2.9.2
payload-entry	$$payload-extension	Section 2.9.3
evidence-entry	$$evidence-extension	Section 2.9.4

Table 1: CoSWID CDDL Group Extension Points

The "CoSWID Items" registry, defined in Section 6.1, provides a registration mechanism allowing new items, and their associated index values, to be added to the CoSWID model through the use of the CDDL sockets described in the table above. This registration mechanism provides for well-known index values for data items in CoSWID extensions, allowing these index values to be recognized by implementations supporting a given extension.

The following additional CDDL sockets are defined in this document to allow for adding new values to corresponding type choices (i.e., to represent enumerations) via custom CDDL specifications.

Enumeration Name	CDDL Socket	Defined in
version-scheme	$version-scheme	Section 4.1
role	$role	Section 4.2
ownership	$ownership	Section 4.3
rel	$rel	Section 4.4
use	$use	Section 4.5

Table 2: CoSWID CDDL Enumeration Extension Points

A number of IANA registries for CoSWID values are also defined in Section 6.2; these registries allow new values to be registered with IANA for the enumerations above. This registration mechanism supports the definition of new well-known index values and names for new enumeration values used by CoSWID, which can also be used by other software tagging specifications. This registration mechanism allows new standardized enumerated values to be shared between multiple tagging specifications (and associated implementations) over time.

2.3. The concise-swid-tag Map

The CDDL specification for the root concise-swid-tag map is as follows. This rule and its constraints **MUST** be followed when creating or validating a CoSWID tag:

```
concise-swid-tag = {
  tag-id => text / bstr .size 16,
  tag-version => integer,
  ? corpus => bool,
  ? patch => bool,
  ? supplemental => bool,
  software-name => text,
  ? software-version => text,
  ? version-scheme => $version-scheme,
  ? media => text,
  ? software-meta => one-or-more<software-meta-entry>,
  entity => one-or-more<entity-entry>,
  ? link => one-or-more<link-entry>,
  ? payload-or-evidence,
  * $$coswid-extension,
  global-attributes,
}

payload-or-evidence //= ( payload => payload-entry )
payload-or-evidence //= ( evidence => evidence-entry )

tag-id = 0
software-name = 1
entity = 2
evidence = 3
link = 4
software-meta = 5
payload = 6
corpus = 8
patch = 9
media = 10
supplemental = 11
tag-version = 12
software-version = 13
version-scheme = 14

$version-scheme /= multipartnumeric
$version-scheme /= multipartnumeric-suffix
$version-scheme /= alphanumeric
$version-scheme /= decimal
$version-scheme /= semver
$version-scheme /= int / text
multipartnumeric = 1
multipartnumeric-suffix = 2
alphanumeric = 3
decimal = 4
semver = 16384
```

The following list describes each member of the concise-swid-tag root map.

global-attributes: A list of items, including an optional language definition to support the processing of text-string values and an unbounded set of any-attribute items. Described in Section 2.5.

tag-id (index 0): A 16-byte binary string, or a textual identifier, uniquely referencing a software component. The tag identifier **MUST** be globally unique. Failure to ensure global uniqueness can create ambiguity in tag use, since the tag-id serves as the global key for matching and lookups. If represented as a 16-byte binary string, the identifier **MUST** be a valid Universally Unique Identifier (UUID) as defined by [RFC4122]. There are no strict guidelines on how the identifier is structured, but examples include a 16-byte Globally Unique Identifier (GUID) (e.g., class 4 UUID) [RFC4122], or a DNS domain name followed by a "/" and a text string, where the domain name serves to ensure uniqueness across organizations. A textual tag-id value **MUST NOT** contain a sequence of two underscores ("__"). This is because a sequence of two underscores is used to separate the TAG_CREATOR_REGID value and UNIQUE_ID value in a Software Identifier and a sequence of two underscores in a tag-id value could create ambiguity when parsing this identifier. See Section 6.7.

software-name (index 1): A textual item that provides the software component's name. This name is likely the same name that would appear in a package management tool. This item maps to '/SoftwareIdentity/@name' in [SWID].

entity (index 2): Provides information about one or more organizations responsible for producing the CoSWID tag, and producing or releasing the software component referenced by this CoSWID tag. Described in Section 2.6.

evidence (index 3): Can be used to record the results of a software discovery process used to identify untagged software on an endpoint or to represent indicators for why software is believed to be installed on the endpoint. In either case, a CoSWID tag can be created by the tool performing an analysis of the software components installed on the endpoint. This item is mutually exclusive to payload, as evidence is always generated on the target device ad hoc. Described in Section 2.9.4.

link (index 4): Provides a means to establish relationship arcs between the tag and another item. A given link can be used to establish the relationship between tags or to reference another resource that is related to the CoSWID tag, e.g., vulnerability database association, Resource-Oriented Lightweight Information Exchange (ROLIE) Feed [RFC8322], Manufacturer Usage Description (MUD) resource [RFC8520], software download location, etc.). This is modeled after the HTML "link" element. Described in Section 2.7.

software-meta (index 5): An open-ended map of key/value data pairs. A number of predefined keys can be used within this item providing for common usage and semantics across the industry. The use of this map allows any additional attribute to be included in the tag. It is expected that industry groups will use a common set of attribute names to allow for interoperability within their communities. Described in Section 2.8. This item maps to '/SoftwareIdentity/Meta' in [SWID].

payload (index 6): Represents a collection of software artifacts (described by child items) that compose the target software. For example, these artifacts could be the files included with an installer for a corpus tag or installed on an endpoint when the software component is installed for a primary or patch tag. The artifacts listed in a payload may be a superset of the software artifacts that are actually installed. Based on user selections at install time, an

installation might not include every artifact that could be created or executed on the endpoint when the software component is installed or run. This item is mutually exclusive to evidence, as payload can only be provided by an external entity. Described in Section 2.9.3.

corpus (index 8): A boolean value that indicates if the tag identifies and describes an installable software component in its pre-installation state. Installable software includes an installation package or installer for a software component, a software update, or a patch. If the CoSWID tag represents installable software, the corpus item **MUST** be set to "true". If not provided, the default value **MUST** be considered "false".

patch (index 9): A boolean value that indicates if the tag identifies and describes an installed patch that has made incremental changes to a software component installed on an endpoint. If a CoSWID tag is for a patch, the patch item **MUST** be set to "true". If not provided, the default value **MUST** be considered "false". A patch item's value **MUST NOT** be set to "true" if the installation of the associated software package changes the version of a software component.

media (index 10): A text value that provides a hint to the tag consumer to understand what target platform this tag applies to. This item **MUST** be formatted as a query as defined by the W3C "Media Queries Level 3" Recommendation (see [W3C.REC-mediaqueries-3-20220405]). Support for media queries is included here for interoperability with [SWID], which does not provide any further requirements for media query use. Thus, this specification does not clarify how a media query is to be used for a CoSWID.

supplemental (index 11): A boolean value that indicates if the tag is providing additional information to be associated with another referenced SWID or CoSWID tag. This allows tools and users to record their own metadata about a software component without modifying SWID primary or patch tags created by a software provider. If a CoSWID tag is a supplemental tag, the supplemental item **MUST** be set to "true". If not provided, the default value **MUST** be considered "false".

tag-version (index 12): An integer value that indicates the specific release revision of the tag. Typically, the initial value of this field is set to 0 and the value is increased for subsequent tags produced for the same software component release. This value allows a CoSWID tag producer to correct an incorrect tag previously released without indicating a change to the underlying software component the tag represents. For example, the tag-version could be changed to add new metadata, to correct a broken link, to add a missing payload entry, etc. When producing a revised tag, the new tag-version value **MUST** be greater than the old tag-version value.

software-version (index 13): A textual value representing the specific release or development version of the software component. This item maps to '/SoftwareIdentity/@version' in [SWID].

version-scheme (index 14): An integer or textual value representing the versioning scheme used for the software-version item, as an integer label with text escape. For the "Version Scheme" values, see Section 4.1. If an integer value is used, it **MUST** be an index value in the range -256 to 65535. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 65535 correspond to registered entries in the IANA "Software ID Version Scheme Values" registry (see Section 6.2.4).

$$coswid-extension: A CDDL socket that is used to add new information structures to the concise-swid-tag root map. See Section 2.2.

2.4. concise-swid-tag Co-constraints

The following co-constraints apply to the information provided in the concise-swid-tag group.

- The patch and supplemental items **MUST NOT** both be set to "true".
- If the patch item is set to "true", the tag **MUST** contain at least one link item (see Section 2.7) with both the rel item value of "patches" and an href item specifying an association with the software that was patched. Without at least one link item, the target of the patch cannot be identified and the patch tag cannot be applied without external context.
- If all of the corpus, patch, and supplemental items are "false" or if the corpus item is set to "true", then a software-version item **MUST** be included with a value set to the version of the software component.

2.5. The global-attributes Group

The global-attributes group provides a list of items, including an optional language definition to support the processing of text-string values, and an unbounded set of any-attribute items allowing for additional items to be provided as a general point of extension in the model.

The CDDL for the global-attributes group follows:

```
global-attributes = (
  ? lang => text,
  * any-attribute,
)

any-attribute = (
  label => one-or-more<text> / one-or-more<int>
)

label = text / int
```

The following list describes each child item of this group.

lang (index 15): A textual language tag that conforms with the IANA "Language Subtag Registry" [RFC5646]. The context of the specified language applies to all sibling and descendant textual values, unless a descendant object has defined a different language tag. Thus, a new context is established when a descendant object redefines a new language tag. All textual values within a given context **MUST** be considered expressed in the specified language.

any-attribute: A sub-group that provides a means to include arbitrary information via label/index ("key") value pairs. Labels can be either a single integer or text string. Values can be a single integer, a text string, or an array of integers or text strings.

2.6. The entity-entry Map

The CDDL for the entity-entry map follows:

```
entity-entry = {
  entity-name => text,
  ? reg-id => any-uri,
  role => one-or-more<$role>,
  ? thumbprint => hash-entry,
  * $$entity-extension,
  global-attributes,
}

entity-name = 31
reg-id = 32
role = 33
thumbprint = 34

$role /= tag-creator
$role /= software-creator
$role /= aggregator
$role /= distributor
$role /= licensor
$role /= maintainer
$role /= int / text
tag-creator=1
software-creator=2
aggregator=3
distributor=4
licensor=5
maintainer=6
```

The following list describes each child item of this group.

global-attributes: The global-attributes group as described in Section 2.5.

entity-name (index 31): The textual name of the organizational entity claiming the roles specified by the role item for the CoSWID tag. This item maps to '/SoftwareIdentity/Entity/@name' in [SWID].

reg-id (index 32): Registration ID. This value is intended to uniquely identify a naming authority in a given scope (e.g., global, organization, vendor, customer, administrative domain, etc.) for the referenced entity. The value of a registration ID **MUST** be a URI as defined in [RFC3986]; it is not intended to be dereferenced. The scope will usually be the scope of an organization.

role (index 33): An integer or textual value (integer label with text escape; see Section 2) representing the relationship(s) between the entity and this tag or the referenced software component. If an integer value is used, it **MUST** be an index value in the range -256 to 255.

Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 255 correspond to registered entries in the IANA "Software ID Entity Role Values" registry (see Section 6.2.5).

The following additional requirements exist for the use of the role item:

- An entity item **MUST** be provided with the role of "tag-creator" for every CoSWID tag. This indicates the organization that created the CoSWID tag.
- An entity item **SHOULD** be provided with the role of "software-creator" for every CoSWID tag, if this information is known to the tag creator. This indicates the organization that created the referenced software component.

thumbprint (index 34): Value that provides a hash (i.e., the thumbprint) of the signing entity's public key certificate. This item provides an indicator of which entity signed the CoSWID tag, which will typically be the tag creator. See Section 2.9.1 for more details on the use of the hash-entry data structure.

$$entity-extension: A CDDL socket that can be used to extend the entity-entry group model. See Section 2.2.

2.7. The link-entry Map

The CDDL for the link-entry map follows:

```
link-entry = {
  ? artifact => text,
  href => any-uri,
  ? media => text,
  ? ownership => $ownership,
  rel => $rel,
  ? media-type => text,
  ? use => $use,
  * $$link-extension,
  global-attributes,
}

media = 10
artifact = 37
href = 38
ownership = 39
rel = 40
media-type = 41
use = 42

$ownership /= shared
$ownership /= private
$ownership /= abandon
$ownership /= int / text
abandon=1
private=2
shared=3
```

```
$rel /= ancestor
$rel /= component
$rel /= feature
$rel /= installationmedia
$rel /= packageinstaller
$rel /= parent
$rel /= patches
$rel /= requires
$rel /= see-also
$rel /= supersedes
$rel /= supplemental
$rel /= -256..65536 / text
ancestor=1
component=2
feature=3
installationmedia=4
packageinstaller=5
parent=6
patches=7
requires=8
see-also=9
supersedes=10
supplemental=11

$use /= optional
$use /= required
$use /= recommended
$use /= int / text
optional=1
required=2
recommended=3
```

The following list describes each member of this map.

global-attributes: The global-attributes group as described in Section 2.5.

media (index 10): A value that provides a hint to the consumer of the link so that the consumer understands what target platform the link is applicable to. This item represents a query as defined by the W3C "Media Queries Level 3" Recommendation (see [W3C.REC-mediaqueries-3-20220405]). As highlighted in the definition of the media item provided in Section 2.3, support for media queries is included here for interoperability with [SWID], which does not provide any further requirements for media query use. Thus, this specification does not clarify how a media query is to be used for a CoSWID.

artifact (index 37): To be used with rel="installationmedia". This item's value provides the absolute filesystem path to the installer executable or script that can be run to launch the referenced installation. Links with the same artifact name **MUST** be considered mirrors of each other, allowing the installation media to be acquired from any of the described sources.

href (index 38): A URI-reference [RFC3986] for the referenced resource. The href item's value can be, but is not limited to, the following (which is a slightly modified excerpt from [SWID]):

- If no URI scheme is provided, then the URI-reference is a relative reference to the base URI of the CoSWID tag, i.e., the URI under which the CoSWID tag was provided -- for example, "./folder/supplemental.coswid".
- This item can be a physical resource location with any acceptable URI scheme (e.g., <file://>, <http://>, <https://>, <ftp://>).
- A URI-like expression with "swid:" as the scheme refers to another SWID or CoSWID by the referenced tag's tag-id. This expression needs to be resolved in the context of the endpoint by software that can look up other SWID or CoSWID tags. For example, "swid:2df9de35-0aff-4a86-ace6-f7dddd1ade4c" references the tag with the tag-id value "2df9de35-0aff-4a86-ace6-f7dddd1ade4c". See Section 5.1 for guidance on the "swid" expressions.
- This item can be a URI-like expression with "swidpath:" as the scheme, which refers to another software tag via an XPath query [W3C.REC-xpath20-20101214] that matches items in that tag (Section 5.2). This scheme is provided for compatibility with [SWID]. This specification does not define how to resolve an XPath query in the context of CBOR. See Section 5.2 for guidance on the "swidpath" expressions.

ownership (index 39): An integer or textual value (integer label with text escape; see Section 2). See Section 4.3 for the list of values available for this item. This item is used when the href item references another software component to indicate the degree of ownership between the software component referenced by the CoSWID tag and the software component referenced by the link. If an integer value is used, it **MUST** be an index value in the range -256 to 255. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 255 correspond to registered entries in the "Software ID Link Ownership Values" registry.

rel (index 40): An integer or textual value (integer label with text escape; see Section 2). See Section 4.4 for the list of values available for this item. This item identifies the relationship between this CoSWID and the target resource identified by the href item. If an integer value is used, it **MUST** be an index value in the range -256 to 65535. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 65535 correspond to registered entries in the IANA "Software ID Link Relationship Values" registry (see Section 6.2.7). If a string value is used, it **MUST** be either a private use name as defined in Section 6.2.2 or a "Relation Name" from the IANA "Link Relation Types" registry (see <https://www.iana.org/assignments/link-relations/>) as defined by [RFC8288]. When a string value defined in the IANA "Software ID Link Relationship Values" registry matches a Relation Name defined in the IANA "Link Relation Types" registry, the index value in the IANA "Software ID Link Relationship Values" registry **MUST** be used instead, as this relationship has a specialized meaning in the context of a CoSWID tag. String values correspond to registered entries in the "Software ID Link Relationship Values" registry.

media-type (index 41): Supplies the resource consumer with a hint regarding what type of resource to expect. A link can point to arbitrary resources on the endpoint, local network, or Internet using the href item. (This is a *hint*: there is no obligation for the server hosting the target of the URI to use the indicated media type when the URI is dereferenced.) Media types are identified by referencing a "Name" from the IANA "Media Types" registry (see <https://www.iana.org/assignments/media-types/>). This item maps to '/SoftwareIdentity/Link/@type' in [SWID].

use (index 42): An integer or textual value (integer label with text escape; see Section 2). See Section 4.5 for the list of values available for this item. This item is used to determine if the referenced software component has to be installed before installing the software component identified by the CoSWID tag. If an integer value is used, it **MUST** be an index value in the range -256 to 255. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 255 correspond to registered entries in the IANA "Software ID Link Use Values" registry (see Section 6.2.8). If a string value is used, it **MUST** be a private use name as defined in Section 6.2.2. String values correspond to registered entries in the "Software ID Link Use Values" registry.

$$link-extension: A CDDL socket that can be used to extend the link-entry map model. See Section 2.2.

2.8. The software-meta-entry Map

The CDDL for the software-meta-entry map follows:

```
software-meta-entry = {
  ? activation-status => text,
  ? channel-type => text,
  ? colloquial-version => text,
  ? description => text,
  ? edition => text,
  ? entitlement-data-required => bool,
  ? entitlement-key => text,
  ? generator => text / bstr .size 16,
  ? persistent-id => text,
  ? product => text,
  ? product-family => text,
  ? revision => text,
  ? summary => text,
  ? unspsc-code => text,
  ? unspsc-version => text,
  * $$software-meta-extension,
  global-attributes,
}

activation-status = 43
channel-type = 44
colloquial-version = 45
description = 46
edition = 47
entitlement-data-required = 48
entitlement-key = 49
generator = 50
persistent-id = 51
product = 52
product-family = 53
revision = 54
summary = 55
unspsc-code = 56
unspsc-version = 57
```

The following list describes each child item of this group.

global-attributes: The global-attributes group as described in Section 2.5.

activation-status (index 43): A textual value that identifies how the software component has been activated, which might relate to specific terms and conditions for its use (e.g., trial, serialized, licensed, unlicensed, etc.) and relate to an entitlement. This attribute is typically used in supplemental tags, as it contains information that might be selected during a specific install.

channel-type (index 44): A textual value that identifies which sales, licensing, or marketing channel the software component has been targeted for (e.g., volume, retail, original equipment manufacturer (OEM), academic, etc.). This attribute is typically used in supplemental tags, as it contains information that might be selected during a specific install.

colloquial-version (index 45): A textual value for the software component's informal or colloquial version. Examples may include a year value, a major version number, or a similar value used to identify a group of specific software component releases that are part of the same release/support cycle. This version can be the same through multiple releases of a software component, while the software-version specified in the concise-swid-tag group is much more specific and will change for each software component release. This version is intended to be used for string comparison (byte by byte) only and is not intended to be used to determine if a specific value is earlier or later in a sequence.

description (index 46): A textual value that provides a detailed description of the software component. This value **MAY** be multiple paragraphs separated by CR LF characters as described by [RFC5198].

edition (index 47): A textual value indicating that the software component represents a functional variation of the code base used to support multiple software components. For example, this item can be used to differentiate enterprise, standard, or professional variants of a software component.

entitlement-data-required (index 48): A boolean value that can be used to determine if accompanying proof of entitlement is needed when a software license reconciliation process is performed.

entitlement-key (index 49): A vendor-specific textual key that can be used to identify and establish a relationship to an entitlement. Examples of an entitlement-key might include a serial number, product key, or license key. For values that relate to a given software component install (e.g., license key), a supplemental tag will typically contain this information. In other cases, where a general-purpose key can be provided that applies to all possible installs of the software component on different endpoints, a primary tag will typically contain this information. Since CoSWID tags are not intended to contain confidential information, tag authors are advised not to record unprotected, private software license keys in this field.

generator (index 50): The name (or tag-id) of the software component that created the CoSWID tag. If the generating software component has a SWID or CoSWID tag, then the tag-id for the generating software component **SHOULD** be provided.

persistent-id (index 51): A globally unique identifier used to identify a set of software components that are related. Software components sharing the same persistent-id can be different versions. This item can be used to relate software components, released at different points in time or through different release channels, that may not be able to be related through the use of the link item.

product (index 52): A basic name for the software component that can be common across multiple tagged software components (e.g., Apache HTTP daemon (HTTPD)).

product-family (index 53): A textual value indicating the software components' overall product family. This should be used when multiple related software components form a larger capability that is installed on multiple different endpoints. For example, some software families may consist of a server, a client, and shared service components that are part of a

larger capability. Email systems, enterprise applications, backup services, web conferencing, and similar capabilities are examples of families. The use of this item is not intended to represent groups of software that are bundled or installed together. The persistent-id or link items **SHOULD** be used to relate bundled software components.

revision (index 54): A string value indicating an informal or colloquial release version of the software. This value can provide a different version value as compared to the software-version specified in the concise-swid-tag group. This is useful when one or more releases need to have an informal version label that differs from the specific exact version value specified by software-version. Examples can include SP1, RC1, Beta, etc.

summary (index 55): A short description of the software component. This **MUST** be a single sentence suitable for display in a user interface.

unspsc-code (index 56): An 8-digit United Nations Standard Products and Services Code (UNSPSC) classification code for the software component as defined by the UNSPSC [UNSPSC].

unspsc-version (index 57): The UNSPSC version used to define the unspsc-code value.

$$software-meta-extension: A CDDL socket that can be used to extend the software-meta-entry group model. See Section 2.2.

2.9. The Resource Collection Definition

2.9.1. The hash-entry Array

CoSWID adds explicit support for the representation of hash entries using algorithms that are registered in the IANA "Named Information Hash Algorithm Registry" [IANA.named-information]. This array is used by both the hash (index 7) and thumbprint (index 34) values. This is the equivalent of the namespace qualified "hash" attribute in [SWID].

```
hash-entry = [
  hash-alg-id: int,
  hash-value: bytes,
]
```

The number used as a value for hash-alg-id is an integer-based hash algorithm identifier whose value **MUST** refer to an ID in the IANA "Named Information Hash Algorithm Registry" [IANA.named-information] with a Status of "current" (at the time the generator software was built or later); other hash algorithms **MUST NOT** be used. If the hash-alg-id is not known, then the integer value "0" **MUST** be used. This allows for conversion from ISO SWID tags [SWID], which do not allow an algorithm to be identified for this field.

The hash-value **MUST** represent the raw hash value as a byte string (as opposed to, for example, base64 encoded) generated from the representation of the resource using the hash algorithm indicated by hash-alg-id.

2.9.2. The resource-collection Group

The resource-collection group provides a list of items used in both evidence (created by a software discovery process) and payload (installed in an endpoint) content of a CoSWID tag document to structure and differentiate the content of specific CoSWID tag types. Potential content includes directories, files, processes, or resources.

The CDDL for the resource-collection group follows:

```
path-elements-group = ( ? directory => one-or-more<directory-entry>,
                        ? file => one-or-more<file-entry>,
                      )

resource-collection = (
  path-elements-group,
  ? process => one-or-more<process-entry>,
  ? resource => one-or-more<resource-entry>,
  * $$resource-collection-extension,
)

filesystem-item = (
  ? key => bool,
  ? location => text,
  fs-name => text,
  ? root => text,
)

file-entry = {
  filesystem-item,
  ? size => uint,
  ? file-version => text,
  ? hash => hash-entry,
  * $$file-extension,
  global-attributes,
}

directory-entry = {
  filesystem-item,
  ? path-elements => { path-elements-group },
  * $$directory-extension,
  global-attributes,
}

process-entry = {
  process-name => text,
  ? pid => integer,
  * $$process-extension,
  global-attributes,
}

resource-entry = {
  type => text,
  * $$resource-extension,
  global-attributes,
}
```

```
hash = 7
directory = 16
file = 17
process = 18
resource = 19
size = 20
file-version = 21
key = 22
location = 23
fs-name = 24
root = 25
path-elements = 26
process-name = 27
pid = 28
type = 29
```

The following list describes each member of the groups and maps illustrated above.

filesystem-item: A list of common items used for representing the filesystem root, relative location, name, and significance of a file or directory item.

global-attributes: The global-attributes group as described in Section 2.5.

hash (index 7): Value that provides a hash of a file. This item provides an integrity measurement with respect to a specific file. See Section 2.9.1 for more details on the use of the hash-entry data structure.

directory (index 16): Item that allows child directory and file items to be defined within a directory hierarchy for the software component.

file (index 17): Item that allows details about a file to be provided for the software component.

process (index 18): Item that allows details to be provided about the runtime behavior of the software component, such as information that will appear in a process listing on an endpoint.

resource (index 19): Item that can be used to provide details about an artifact or capability expected to be found on an endpoint or evidence collected related to the software component. This can be used to represent concepts not addressed directly by the directory, file, or process items. Examples include registry keys, bound ports, etc. The equivalent construct in [SWID] is currently underspecified. As a result, this item might be further defined through extensions in the future.

size (index 20): The file's size in bytes.

file-version (index 21): The file's version as reported by querying information on the file from the operating system (if available). This item maps to '/SoftwareIdentity/(Payload|Evidence)/File/@version' in [SWID].

key (index 22): A boolean value indicating if a file or directory is significant or required for the software component to execute or function properly. These are files or directories that can be used to affirmatively determine if the software component is installed on an endpoint.

location (index 23): The filesystem path where a file is expected to be located when installed or copied. The location **MUST** be either an absolute path, a path relative to the path value included in the parent directory item (preferred), or a path relative to the location of the CoSWID tag if no parent is defined. The location **MUST NOT** include a file's name, which is provided by the fs-name item.

fs-name (index 24): The name of the directory or file without any path information. This aligns with a file "name" in [SWID]. This item maps to '/SoftwareIdentity/(Payload|Evidence)/(File|Directory)/@name' in [SWID].

root (index 25): A host-specific name for the root of the filesystem. The location item is considered relative to this location if specified. If not provided, the value provided by the location item is expected to be relative to its parent or the location of the CoSWID tag if no parent is provided.

path-elements (index 26): Group that allows a hierarchy of directory and file items to be defined in payload or evidence items. This is a construction within the CDDL definition of CoSWID to support shared syntax and does not appear in [SWID].

process-name (index 27): The software component's process name as it will appear in an endpoint's process list. This aligns with a process "name" in [SWID]. This item maps to '/SoftwareIdentity/(Payload|Evidence)/Process/@name' in [SWID].

pid (index 28): The process ID identified for a running instance of the software component in the endpoint's process list. This is used as part of the evidence item.

type (index 29): A human-readable string indicating the type of resource.

$$resource-collection-extension: A CDDL socket that can be used to extend the resource-collection group model. This can be used to add new specialized types of resources. See Section 2.2.

$$file-extension: A CDDL socket that can be used to extend the file-entry group model. See Section 2.2.

$$directory-extension: A CDDL socket that can be used to extend the directory-entry group model. See Section 2.2.

$$process-extension: A CDDL socket that can be used to extend the process-entry group model. See Section 2.2.

$$resource-extension: A CDDL socket that can be used to extend the resource-entry group model. See Section 2.2.

2.9.3. The payload-entry Map

The CDDL for the payload-entry map follows:

```
payload-entry = {
  resource-collection,
  * $$payload-extension,
  global-attributes,
}
```

The following list describes each child item of this group.

global-attributes: The global-attributes group as described in Section 2.5.

resource-collection: The resource-collection group as described in Section 2.9.2.

$$payload-extension: A CDDL socket that can be used to extend the payload-entry group model. See Section 2.2.

2.9.4. The evidence-entry Map

The CDDL for the evidence-entry map follows:

```
evidence-entry = {
  resource-collection,
  ? date => integer-time,
  ? device-id => text,
  ? location => text,
  * $$evidence-extension,
  global-attributes,
}

date = 35
device-id = 36
```

The following list describes each child item of this group.

global-attributes: The global-attributes group as described in Section 2.5.

resource-collection: The resource-collection group as described in Section 2.9.2.

location (index 23): The filesystem path of the location of the CoSWID tag generated as evidence. This path is always an absolute file path (unlike the value of a location item found within a filesystem-item as described in Section 2.9.2, which can be either a relative path or an absolute path).

date (index 35): The date and time the information was collected pertaining to the evidence item in epoch-based date/time format as specified in Section 3.4.2 of [RFC8949].

device-id (index 36): The endpoint's string identifier from which the evidence was collected.

$$evidence-extension: A CDDL socket that can be used to extend the evidence-entry group model. See Section 2.2.

2.10. Full CDDL Specification

In order to create a valid CoSWID document, the structure of the corresponding CBOR message **MUST** adhere to the following CDDL specification.

```
<CODE BEGINS> file "concise-swid-tag.cddl"

concise-swid-tag = {
  tag-id => text / bstr .size 16,
  tag-version => integer,
  ? corpus => bool,
  ? patch => bool,
  ? supplemental => bool,
  software-name => text,
  ? software-version => text,
  ? version-scheme => $version-scheme,
  ? media => text,
  ? software-meta => one-or-more<software-meta-entry>,
  entity => one-or-more<entity-entry>,
  ? link => one-or-more<link-entry>,
  ? payload-or-evidence,
  * $$coswid-extension,
  global-attributes,
}

payload-or-evidence //= ( payload => payload-entry )
payload-or-evidence //= ( evidence => evidence-entry )

any-uri = uri
label = text / int

$version-scheme /= multipartnumeric
$version-scheme /= multipartnumeric-suffix
$version-scheme /= alphanumeric
$version-scheme /= decimal
$version-scheme /= semver
$version-scheme /= int / text

any-attribute = (
  label => one-or-more<text> / one-or-more<int>
)

one-or-more<T> = T / [ 2* T ]

global-attributes = (
  ? lang => text,
  * any-attribute,
)

hash-entry = [
  hash-alg-id: int,
  hash-value: bytes,
]

entity-entry = {
```

```
  entity-name => text,
  ? reg-id => any-uri,
  role => one-or-more<$role>,
  ? thumbprint => hash-entry,
  * $$entity-extension,
  global-attributes,
}

$role /= tag-creator
$role /= software-creator
$role /= aggregator
$role /= distributor
$role /= licensor
$role /= maintainer
$role /= int / text

link-entry = {
  ? artifact => text,
  href => any-uri,
  ? media => text,
  ? ownership => $ownership,
  rel => $rel,
  ? media-type => text,
  ? use => $use,
  * $$link-extension,
  global-attributes,
}

$ownership /= shared
$ownership /= private
$ownership /= abandon
$ownership /= int / text

$rel /= ancestor
$rel /= component
$rel /= feature
$rel /= installationmedia
$rel /= packageinstaller
$rel /= parent
$rel /= patches
$rel /= requires
$rel /= see-also
$rel /= supersedes
$rel /= supplemental
$rel /= -256..65536 / text

$use /= optional
$use /= required
$use /= recommended
$use /= int / text

software-meta-entry = {
  ? activation-status => text,
  ? channel-type => text,
  ? colloquial-version => text,
  ? description => text,
  ? edition => text,
  ? entitlement-data-required => bool,
```

```
  ? entitlement-key => text,
  ? generator => text / bstr .size 16,
  ? persistent-id => text,
  ? product => text,
  ? product-family => text,
  ? revision => text,
  ? summary => text,
  ? unspsc-code => text,
  ? unspsc-version => text,
  * $$software-meta-extension,
  global-attributes,
}

path-elements-group = ( ? directory => one-or-more<directory-entry>,
                        ? file => one-or-more<file-entry>,
                      )

resource-collection = (
  path-elements-group,
  ? process => one-or-more<process-entry>,
  ? resource => one-or-more<resource-entry>,
  * $$resource-collection-extension,
)

file-entry = {
  filesystem-item,
  ? size => uint,
  ? file-version => text,
  ? hash => hash-entry,
  * $$file-extension,
  global-attributes,
}

directory-entry = {
  filesystem-item,
  ? path-elements => { path-elements-group },
  * $$directory-extension,
  global-attributes,
}

process-entry = {
  process-name => text,
  ? pid => integer,
  * $$process-extension,
  global-attributes,
}

resource-entry = {
  type => text,
  * $$resource-extension,
  global-attributes,
}

filesystem-item = (
  ? key => bool,
  ? location => text,
  fs-name => text,
  ? root => text,
```

```
)

payload-entry = {
  resource-collection,
  * $$payload-extension,
  global-attributes,
}

evidence-entry = {
  resource-collection,
  ? date => integer-time,
  ? device-id => text,
  ? location => text,
  * $$evidence-extension,
  global-attributes,
}

integer-time = #6.1(int)

; "global map member" integer indices
tag-id = 0
software-name = 1
entity = 2
evidence = 3
link = 4
software-meta = 5
payload = 6
hash = 7
corpus = 8
patch = 9
media = 10
supplemental = 11
tag-version = 12
software-version = 13
version-scheme = 14
lang = 15
directory = 16
file = 17
process = 18
resource = 19
size = 20
file-version = 21
key = 22
location = 23
fs-name = 24
root = 25
path-elements = 26
process-name = 27
pid = 28
type = 29
entity-name = 31
reg-id = 32
role = 33
thumbprint = 34
date = 35
device-id = 36
artifact = 37
href = 38
```

```
ownership = 39
rel = 40
media-type = 41
use = 42
activation-status = 43
channel-type = 44
colloquial-version = 45
description = 46
edition = 47
entitlement-data-required = 48
entitlement-key = 49
generator = 50
persistent-id = 51
product = 52
product-family = 53
revision = 54
summary = 55
unspsc-code = 56
unspsc-version = 57

; "version-scheme" integer indices
multipartnumeric = 1
multipartnumeric-suffix = 2
alphanumeric = 3
decimal = 4
semver = 16384

; "role" integer indices
tag-creator=1
software-creator=2
aggregator=3
distributor=4
licensor=5
maintainer=6

; "ownership" integer indices
abandon=1
private=2
shared=3

; "rel" integer indices
ancestor=1
component=2
feature=3
installationmedia=4
packageinstaller=5
parent=6
patches=7
requires=8
see-also=9
supersedes=10
; supplemental=11 ; already defined

; "use" integer indices
optional=1
required=2
recommended=3
```

```
<CODE ENDS>
```

3. Determining the Type of CoSWID

The operational model for SWID and CoSWID tags was introduced in Section 1.1, which described four different CoSWID tag types. The following additional rules apply to the use of CoSWID tags to ensure that created tags properly identify the tag type.

The first matching rule **MUST** determine the type of the CoSWID tag.

Primary Tag: A CoSWID tag **MUST** be considered a primary tag if the corpus, patch, and supplemental items are "false".

Supplemental Tag: A CoSWID tag **MUST** be considered a supplemental tag if the supplemental item is set to "true".

Corpus Tag: A CoSWID tag **MUST** be considered a corpus tag if the corpus item is "true".

Patch Tag: A CoSWID tag **MUST** be considered a patch tag if the patch item is "true".

> Note: It is possible for some or all of the corpus, patch, and supplemental items to simultaneously have values set as "true". The rules above provide a means to determine the tag's type in such a case. For example, a SWID or CoSWID tag for a patch installer might have both corpus and patch items set to "true". In such a case, the tag is a "corpus tag". The tag installed by this installer would have only the patch item set to "true", making the installed tag type a "patch tag".

4. CoSWID Indexed Label Values

This section defines multiple kinds of indexed label values that are maintained in several IANA registries. See Section 6 for details. These values are represented as positive integers. In each registry, the value 0 is marked as Reserved.

4.1. Version Scheme

The following table contains a set of values for use in the concise-swid-tag group's version-scheme item. The "Index" value indicates the value to use as the version-scheme item's value. Strings in the "Version Scheme Name" column provide human-readable text for the value and match the version schemes defined in the ISO/IEC 19770-2:2015 specification [SWID]. The "Definition" column describes the syntax of allowed values for each entry.

Index	Version Scheme Name	Definition
1	multipartnumeric	Numbers separated by dots, where the numbers are interpreted as decimal integers (e.g., 1.2.3, 1.2.3.4.5.6.7, 1.4.5, 1.21)
2	multipartnumeric+suffix	Numbers separated by dots, where the numbers are interpreted as decimal integers with an additional textual suffix (e.g., 1.2.3a)
3	alphanumeric	Strictly a string, no interpretation as number
4	decimal	A single decimal floating-point number
16384	semver	A semantic version as defined by [SWID]. Also see the [SEMVER] specification for more information

Table 3: Version Scheme Values

"multipartnumeric" and the numbers part of "multipartnumeric+suffix" are interpreted as a sequence of numbers and are sorted in lexicographical order by these numbers (i.e., not by the digits in the numbers) and then the textual suffix (for "multipartnumeric+suffix"). "alphanumeric" strings are sorted lexicographically as character strings. "decimal" version numbers are interpreted as single floating-point numbers (e.g., 1.25 is less than 1.3).

The values above are registered in the IANA "Software ID Version Scheme Values" registry, defined in Section 6.2.4. Additional entries will likely be registered over time in this registry.

A CoSWID producer that is aware of the version scheme that has been used to select the version value **SHOULD** include the optional version-scheme item to avoid semantic ambiguity. If the CoSWID producer does not have this information, it **SHOULD** omit the version-scheme item. The following heuristics can be used by a CoSWID consumer, based on the version schemes' partially overlapping value spaces:

- "decimal" and "multipartnumeric" partially overlap in their value space when a value matches a decimal number. When a corresponding software-version item's value falls within this overlapping value space, it is expected that the "decimal" version scheme is used.
- "multipartnumeric" and "semver" partially overlap in their value space when a "multipartnumeric" value matches the semantic versioning syntax. When a corresponding software-version item's value falls within this overlapping value space, it is expected that the "semver" version scheme is used.
- "alphanumeric" and other version schemes might overlap in their value space. When a corresponding software-version item's value falls within this overlapping value space, it is expected that the other version scheme is used and "alphanumeric" is not used.

Note that these heuristics are imperfect and can guess wrong, which is the reason the version-scheme item **SHOULD** be included by the producer.

4.2. Entity Role Values

The following table indicates the index value to use for the entity-entry group's role item (see Section 2.6). These values match the entity roles defined in the ISO/IEC 19770-2:2015 specification [SWID]. The "Index" value indicates the value to use as the role item's value. Items in the "Role Name" column provide human-readable text for the value. The "Definition" column describes the semantic meaning of each entry.

Index	Role Name	Definition
1	tagCreator	The person or organization that created the containing SWID or CoSWID tag.
2	softwareCreator	The person or organization entity that created the software component.
3	aggregator	From [SWID], "An organization or system that encapsulates software from their own and/or other organizations into a different distribution process (as in the case of virtualization), or as a completed system to accomplish a specific task (as in the case of a value added reseller)."
4	distributor	From [SWID], "An entity that furthers the marketing, selling and/or distribution of software from the original place of manufacture to the ultimate user without modifying the software, its packaging or its labelling."
5	licensor	From [SAM], as a "software licensor", a "person or organization who owns or holds the rights to issue a software license for a specific software [component]."
6	maintainer	The person or organization that is responsible for coordinating and making updates to the source code for the software component. This **SHOULD** be used when the "maintainer" is a different person or organization than the original "softwareCreator".

Table 4: Entity Role Values

The values above are registered in the IANA "Software ID Entity Role Values" registry, defined in Section 6.2.5. Additional values will likely be registered over time.

4.3. Link Ownership Values

The following table indicates the index value to use for the link-entry group's ownership item (see Section 2.7). These values match the link ownership values defined in the ISO/IEC 19770-2:2015 specification [SWID]. The "Index" value indicates the value to use as the link-entry group ownership item's value. Items in the "Ownership Type" column provide human-readable text for the value. The "Definition" column describes the semantic meaning of each entry.

Index	Ownership Type	Definition
1	abandon	If the software component referenced by the CoSWID tag is uninstalled, then the referenced software **SHOULD NOT** be uninstalled.
2	private	If the software component referenced by the CoSWID tag is uninstalled, then the referenced software **SHOULD** be uninstalled as well.
3	shared	If the software component referenced by the CoSWID tag is uninstalled, then the referenced software **SHOULD** be uninstalled if no other components are sharing the software.

Table 5: Link Ownership Values

The values above are registered in the IANA "Software ID Link Ownership Values" registry, defined in Section 6.2.6. Additional values will likely be registered over time.

4.4. Link Rel Values

The following table indicates the index value to use for the link-entry group's rel item (see Section 2.7). These values match the link rel values defined in the ISO/IEC 19770-2:2015 specification [SWID]. The "Index" value indicates the value to use as the link-entry group ownership item's value. Items in the "Relationship Type" column provide human-readable text for the value. The "Definition" column describes the semantic meaning of each entry.

Index	Relationship Type	Definition
1	ancestor	The link references a software tag for a previous release of this software. This can be useful to define an upgrade path.
2	component	The link references a software tag for a separate component of this software.

Index	Relationship Type	Definition
3	feature	The link references a configurable feature of this software that can be enabled or disabled without changing the installed files.
4	installationmedia	The link references the installation package that can be used to install this software.
5	packageinstaller	The link references the installation software needed to install this software.
6	parent	The link references a software tag that is the parent of the referencing tag. This relationship can be used when multiple software components are part of a software bundle, where the "parent" is the software tag for the bundle and each child is a "component". In such a case, each child component can provide a "parent" link relationship to the bundle's software tag, and the bundle can provide a "component" link relationship to each child software component.
7	patches	The link references a software tag that the referencing software patches. Typically only used for patch tags (see Section 1.1).
8	requires	The link references a prerequisite for installing this software. A patch tag (see Section 1.1) can use this to represent base software or another patch that needs to be installed first.
9	see-also	The link references other software that may be of interest that relates to this software.
10	supersedes	The link references other software (e.g., an older software version) that this software replaces. A patch tag (see Section 1.1) can use this to represent another patch that this patch incorporates or replaces.
11	supplemental	The link references a software tag that the referencing tag supplements. Used on supplemental tags (see Section 1.1).

Table 6: Link Relationship Values

The values above are registered in the IANA "Software ID Link Relationship Values" registry, defined in Section 6.2.7. Additional values will likely be registered over time.

4.5. Link Use Values

The following table indicates the index value to use for the link-entry group's use item (see Section 2.7). These values match the link use values defined in the ISO/IEC 19770-2:2015 specification [SWID]. The "Index" value indicates the value to use as the link-entry group use item's value. Items in the "Use Type" column provide human-readable text for the value. The "Definition" column describes the semantic meaning of each entry.

Index	Use Type	Definition
1	optional	From [SWID], "Not absolutely required; the [Link]'d software is installed only when specified."
2	required	From [SWID], "The [Link]'d software is absolutely required for an operation software installation."
3	recommended	From [SWID], "Not absolutely required; the [Link]'d software is installed unless specified otherwise."

Table 7: Link Use Values

The values above are registered in the IANA "Software ID Link Use Values" registry, defined in Section 6.2.8. Additional values will likely be registered over time.

5. "swid" and "swidpath" Expressions

This specification defines the following scheme names for use in CoSWID and to provide interoperability with scheme names used in [SWID]. Because both the "swid" and "swidpath" scheme names are to be interpreted within a local (rather than a global) context, neither of these are technically URI scheme names as defined in [RFC3986]. For this reason, the "swid" and "swidpath" scheme names are registered with IANA as provisional, rather than permanent, scheme names. However, registering these scheme names as provisional ensures that the scheme names are reserved and that they are properly defined going forward.

The swid and swidpath expressions conform to all rules for URI syntax. All uses of these expressions encountered within a CoSWID are to be interpreted as described in this section.

5.1. "swid" Expressions

Expressions specifying the "swid" scheme are used to reference a software tag by its tag-id. A tag-id referenced in this way can be used to identify the tag resource in the context of where it is referenced from. For example, when a tag is installed on a given device, that tag can reference related tags on the same device using expressions with this scheme.

For expressions that use the "swid" scheme, the scheme-specific part **MUST** consist of a referenced software tag's tag-id. This tag-id **MUST** be URI encoded according to Section 2.1 of [RFC3986].

The following expression is a valid example:

```
swid:2df9de35-0aff-4a86-ace6-f7dddd1ade4c
```

5.2. "swidpath" Expressions

Expressions specifying the "swidpath" scheme are used to filter tags out of a base collection, so that matching tags are included in the identified tag collection. The XPath expression [W3C.REC-xpath20-20101214] references the data that must be found in a given software tag out of the base collection for that tag to be considered a matching tag. Tags to be evaluated (the base collection) include all tags in the context of where the "swidpath" expression is referenced from. For example, when a tag is installed on a given device, that tag can reference related tags on the same device using an expression with this scheme.

For URIs that use the "swidpath" scheme, the following requirements apply:

- The scheme-specific part **MUST** be an XPath expression as defined by [W3C.REC-xpath20-20101214]. The included XPath expression will be URI encoded according to Section 2.1 of [RFC3986].
- This XPath is evaluated over SWID tags, or CoSWID tags transformed into SWID tags, found on a system. A given tag **MUST** be considered a match if the XPath evaluation result value has an effective boolean value of "true" according to [W3C.REC-xpath20-20101214], Section 2.4.3.

6. IANA Considerations

This document has a number of IANA considerations, as described in the following subsections. In summary, six new registries are established by this document, with initial entries provided for each registry. New values for five other registries are also defined.

6.1. CoSWID Items Registry

This document defines a new registry titled "CoSWID Items". This registry uses integer values as index values in CBOR maps. Future registrations for this registry are to be made based on [BCP26] as follows:

Range	Registration Procedures
0-32767	Standards Action with Expert Review
32768-4294967295	Specification Required

Table 8: CoSWID Items Registration Procedures

All negative values are reserved for private use.

Initial registrations for the "CoSWID Items" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Reference
0	tag-id	RFC 9393
1	software-name	RFC 9393
2	entity	RFC 9393
3	evidence	RFC 9393
4	link	RFC 9393
5	software-meta	RFC 9393
6	payload	RFC 9393
7	hash	RFC 9393
8	corpus	RFC 9393
9	patch	RFC 9393
10	media	RFC 9393
11	supplemental	RFC 9393
12	tag-version	RFC 9393
13	software-version	RFC 9393
14	version-scheme	RFC 9393
15	lang	RFC 9393
16	directory	RFC 9393
17	file	RFC 9393
18	process	RFC 9393
19	resource	RFC 9393
20	size	RFC 9393
21	file-version	RFC 9393
22	key	RFC 9393
23	location	RFC 9393

Index	Item Name	Reference
24	fs-name	RFC 9393
25	root	RFC 9393
26	path-elements	RFC 9393
27	process-name	RFC 9393
28	pid	RFC 9393
29	type	RFC 9393
30	Unassigned	
31	entity-name	RFC 9393
32	reg-id	RFC 9393
33	role	RFC 9393
34	thumbprint	RFC 9393
35	date	RFC 9393
36	device-id	RFC 9393
37	artifact	RFC 9393
38	href	RFC 9393
39	ownership	RFC 9393
40	rel	RFC 9393
41	media-type	RFC 9393
42	use	RFC 9393
43	activation-status	RFC 9393
44	channel-type	RFC 9393
45	colloquial-version	RFC 9393
46	description	RFC 9393
47	edition	RFC 9393

Index	Item Name	Reference
48	entitlement-data-required	RFC 9393
49	entitlement-key	RFC 9393
50	generator	RFC 9393
51	persistent-id	RFC 9393
52	product	RFC 9393
53	product-family	RFC 9393
54	revision	RFC 9393
55	summary	RFC 9393
56	unspsc-code	RFC 9393
57	unspsc-version	RFC 9393
58-4294967295	Unassigned	

Table 9: CoSWID Items Initial Registrations

6.2. Registries for Software ID Values

The following IANA registries provide a mechanism for new values to be added over time to common enumerations used by SWID and CoSWID. While neither the CoSWID specification nor the SWID specification is subordinate to the other and will evolve as their respective standards group chooses, there is value in supporting alignment between the two standards. Shared use of common code points, as spelled out in these registries, will facilitate this alignment -- hence the intent for shared use of these registries and the decision to use "swidsoftware-id" (rather than "swid" or "coswid") in registry names.

6.2.1. Registration Procedures

The following registries allow for the registration of index values and names. New registrations will be permitted through either a Standards Action with Expert Review policy or a Specification Required policy [BCP26].

The following registries also reserve the integer-based index values in the range of -1 to -256 for private use as defined by Section 4.1 of [BCP26]. This allows values -1 to -24 to be expressed as a single uint8_t in CBOR and values -25 to -256 to be expressed using an additional uint8_t in CBOR.

6.2.2. Private Use of Index and Name Values

The integer-based index values in the private use range (-1 to -256) are intended for testing purposes and closed environments; values in other ranges **SHOULD NOT** be assigned for testing.

For names that correspond to private use index values, an Internationalized Domain Name prefix **MUST** be used to prevent name conflicts using the form

```
domainprefix/name
```

where both "domainprefix" and "name" **MUST** each be either a Non-Reserved LDH (NR-LDH) label or a U-label as defined by [RFC5890], and "name" also **MUST** be a unique name within the namespace defined by the "domainprefix". ("LDH" is an abbreviation for "letters, digits, hyphen".) Using a prefix in this way allows for a name to be used in the private use range. This is consistent with the guidance in [BCP178].

6.2.3. Expert Review Criteria

Designated experts **MUST** ensure that new registration requests meet the following additional criteria:

- The requesting specification **MUST** provide a clear semantic definition for the new entry. This definition **MUST** clearly differentiate the requested entry from other previously registered entries.
- The requesting specification **MUST** describe the intended use of the entry, including any co-constraints that exist between (1) the use of the entry's index value or name and (2) other values defined within the SWID/CoSWID model.
- Index values and names outside the private use space **MUST NOT** be used without registration. This is considered "squatting" and **MUST** be avoided. Designated experts **MUST** ensure that reviewed specifications register all appropriate index values and names.
- Standards Track documents **MAY** include entries registered in the range reserved for entries under the Specification Required policy. This can occur when a Standards Track document provides further guidance on the use of index values and names that are in common use but were not registered with IANA. This situation **SHOULD** be avoided.
- All registered names **MUST** be valid according to the XML Schema NMTOKEN data type (see [W3C.REC-xmlschema-2-20041028], Section 3.3.4). This ensures that registered names are compatible with the SWID format [SWID] where they are used.
- Registration of vanity names **SHOULD** be discouraged. The requesting specification **MUST** provide a description of how a requested name will allow for use by multiple stakeholders.

6.2.4. Software ID Version Scheme Values Registry

This document establishes a new registry titled "Software ID Version Scheme Values". This registry provides index values for use as version-scheme item values in this document and Version Scheme Names for use in [SWID].

This registry uses the registration procedures defined in Section 6.2.1, with the following associated ranges:

Range	Registration Procedures
0-16383	Standards Action with Expert Review
16384-65535	Specification Required

Table 10: Software ID Version Scheme Registration Procedures

Assignments **MUST** consist of an integer index value, the Version Scheme Name, and a reference to the defining specification.

Initial registrations for the "Software ID Version Scheme Values" registry are provided below and are derived from the textual Version Scheme Names defined in [SWID].

Index	Version Scheme Name	Reference
0	Reserved	
1	multipartnumeric	RFC 9393, Section 4.1
2	multipartnumeric+suffix	RFC 9393, Section 4.1
3	alphanumeric	RFC 9393, Section 4.1
4	decimal	RFC 9393, Section 4.1
5-16383	Unassigned	
16384	semver	RFC 9393, Section 4.1
16385-65535	Unassigned	

Table 11: Software ID Version Scheme Initial Registrations

Registrations **MUST** conform to the expert review criteria defined in Section 6.2.3.

Designated experts **MUST** also ensure that newly requested entries define a value space for the corresponding software-version item that is unique from other previously registered entries.

> Note: The initial registrations violate this requirement but are included for backwards compatibility with [SWID]. See also Section 4.1.

6.2.5. Software ID Entity Role Values Registry

This document establishes a new registry titled "Software ID Entity Role Values". This registry provides index values for use as entity-entry role item values in this document and entity role names for use in [SWID].

This registry uses the registration procedures defined in Section 6.2.1, with the following associated ranges:

Range	Registration Procedures
0-127	Standards Action with Expert Review
128-255	Specification Required

Table 12: Software ID Entity Role Registration Procedures

Assignments consist of an integer index value, a role name, and a reference to the defining specification.

Initial registrations for the "Software ID Entity Role Values" registry are provided below and are derived from the textual entity role names defined in [SWID].

Index	Role Name	Reference
0	Reserved	
1	tagCreator	RFC 9393, Section 4.2
2	softwareCreator	RFC 9393, Section 4.2
3	aggregator	RFC 9393, Section 4.2
4	distributor	RFC 9393, Section 4.2
5	licensor	RFC 9393, Section 4.2
6	maintainer	RFC 9393, Section 4.2
7-255	Unassigned	

Table 13: Software ID Entity Role Initial Registrations

Registrations **MUST** conform to the expert review criteria defined in Section 6.2.3.

6.2.6. Software ID Link Ownership Values Registry

This document establishes a new registry titled "Software ID Link Ownership Values". This registry provides index values for use as link-entry ownership item values in this document and link ownership names for use in [SWID].

This registry uses the registration procedures defined in Section 6.2.1, with the following associated ranges:

Range	Registration Procedures
0-127	Standards Action with Expert Review
128-255	Specification Required

Table 14: Software ID Link Ownership Registration Procedures

Assignments consist of an integer index value, an ownership type name, and a reference to the defining specification.

Initial registrations for the "Software ID Link Ownership Values" registry are provided below and are derived from the textual entity role names defined in [SWID].

Index	Ownership Type Name	Reference
0	Reserved	
1	abandon	RFC 9393, Section 4.3
2	private	RFC 9393, Section 4.3
3	shared	RFC 9393, Section 4.3
4-255	Unassigned	

Table 15: Software ID Link Ownership Initial Registrations

Registrations **MUST** conform to the expert review criteria defined in Section 6.2.3.

6.2.7. Software ID Link Relationship Values Registry

This document establishes a new registry titled "Software ID Link Relationship Values". This registry provides index values for use as link-entry rel item values in this document and link ownership names for use in [SWID].

This registry uses the registration procedures defined in Section 6.2.1, with the following associated ranges:

Range	Registration Procedures
0-32767	Standards Action with Expert Review
32768-65535	Specification Required

Table 16: Software ID Link Relationship Registration Procedures

Assignments consist of an integer index value, the relationship type name, and a reference to the defining specification.

Initial registrations for the "Software ID Link Relationship Values" registry are provided below and are derived from the link relationship values defined in [SWID].

Index	Relationship Type Name	Reference
0	Reserved	
1	ancestor	RFC 9393, Section 4.4
2	component	RFC 9393, Section 4.4
3	feature	RFC 9393, Section 4.4
4	installationmedia	RFC 9393, Section 4.4
5	packageinstaller	RFC 9393, Section 4.4
6	parent	RFC 9393, Section 4.4
7	patches	RFC 9393, Section 4.4
8	requires	RFC 9393, Section 4.4
9	see-also	RFC 9393, Section 4.4
10	supersedes	RFC 9393, Section 4.4
11	supplemental	RFC 9393, Section 4.4
12-65535	Unassigned	

Table 17: Software ID Link Relationship Initial Registrations

Registrations **MUST** conform to the expert review criteria defined in Section 6.2.3.

Designated experts **MUST** also ensure that a newly requested entry documents the URI schemes allowed to be used in an href associated with the link relationship and the expected resolution behavior of these URI schemes. This will help to ensure that applications processing software tags are able to interoperate when resolving resources referenced by a link of a given type.

6.2.8. Software ID Link Use Values Registry

This document establishes a new registry titled "Software ID Link Use Values". This registry provides index values for use as link-entry use item values in this document and link use names for use in [SWID].

This registry uses the registration procedures defined in Section 6.2.1, with the following associated ranges:

Range	Registration Procedures
0-127	Standards Action with Expert Review
128-255	Specification Required

Table 18: Software ID Link Use Registration Procedures

Assignments consist of an integer index value, the link use type name, and a reference to the defining specification.

Initial registrations for the "Software ID Link Use Values" registry are provided below and are derived from the link relationship values defined in [SWID].

Index	Link Use Type Name	Reference
0	Reserved	
1	optional	RFC 9393, Section 4.5
2	required	RFC 9393, Section 4.5
3	recommended	RFC 9393, Section 4.5
4-255	Unassigned	

Table 19: Software ID Link Use Initial Registrations

Registrations **MUST** conform to the expert review criteria defined in Section 6.2.3.

6.3. swid+cbor Media Type Registration

IANA has added the following to the "Media Types" registry [IANA.media-types].

Type name: application

Subtype name: swid+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Binary (encoded as CBOR [RFC8949]). See RFC 9393 for details.

Security considerations: See Section 9 of RFC 9393.

Interoperability considerations: Applications **MAY** ignore any key value pairs that they do not understand. This allows backwards-compatible extensions to this specification.

Published specification: RFC 9393

Applications that use this media type: The type is used by software asset management systems and vulnerability assessment systems and is used in applications that use remote integrity verification.

Fragment Identifier Considerations: The syntax and semantics of fragment identifiers specified for "application/swid+cbor" are as specified for "application/cbor". (At publication of RFC 9393, there is no fragment identification syntax defined for "application/cbor".)

Additional information:
Magic number(s): If tagged, the first five bytes in hex: da 53 57 49 44 (see Section 8 of RFC 9393).
File extension(s): coswid
Macintosh file type code(s): none
Macintosh Universal Type Identifier code: org.ietf.coswid conforms to public.data.

Person & email address to contact for further information:
IESG <iesg@ietf.org>

Intended usage: COMMON

Restrictions on usage: none

Author: Henk Birkholz <henk.birkholz@sit.fraunhofer.de>

Change controller: IESG

6.4. CoAP Content-Format Registration

IANA has assigned a CoAP Content-Format ID for the CoSWID media type in the "CoAP Content-Formats" subregistry, from the "IETF Review or IESG Approval" space (256..999), within the "CoRE Parameters" registry [RFC7252] [IANA.core-parameters]:

Content Type	Content Coding	ID	Reference
application/swid+cbor	-	258	RFC 9393

Table 20: CoAP Content-Format IDs

6.5. CBOR Tag Registration

IANA has allocated a tag in the "CBOR Tags" registry [IANA.cbor-tags]:

Tag	Data Item	Semantics	Reference
1398229316	map	Concise Software Identifier (CoSWID)	RFC 9393

Table 21: CoSWID CBOR Tag

6.6. URI Scheme Registrations

The ISO 19770-2:2015 SWID specification [SWID] describes the use of the "swid" and "swidpath" URI schemes, which are currently in use in implementations. This document continues this use for CoSWID. The following subsections provide registrations for these schemes to ensure that a registration for these schemes exists that is suitable for use in the SWID and CoSWID specifications.

URI schemes are registered within the "Uniform Resource Identifier (URI) Schemes" registry maintained at [IANA.uri-schemes].

6.6.1. URI Scheme "swid"

IANA has registered the URI scheme "swid". This registration complies with [RFC7595].

Scheme name: swid

Status: Provisional

Applications/protocols that use this scheme name: Applications that require Software IDs (SWIDs) or Concise Software IDs (CoSWIDs); see Section 5.1 of RFC 9393.

Contact: IETF Chair <chair@ietf.org>

Change controller: IESG <iesg@ietf.org>

Reference: Section 5.1 of RFC 9393

> Note: This scheme has been documented by an IETF working group and is mentioned in an IETF Standard specification. However, as it describes a locally scoped, limited-purpose form of identification, it does not fully meet the requirements for permanent registration.
>
> As long as this specification (or any successors that describe this scheme) is a current IETF specification, this scheme should be considered to be "in use" and not considered for removal from the registry.

6.6.2. URI Scheme "swidpath"

IANA has registered the URI scheme "swidpath". This registration complies with [RFC7595].

Scheme name: swidpath

Status: Provisional

Applications/protocols that use this scheme name: Applications that require Software IDs (SWIDs) or Concise Software IDs (CoSWIDs); see Section 5.2 of RFC 9393.

Contact: IETF Chair <chair@ietf.org>

Change controller: IESG <iesg@ietf.org>

Reference: Section 5.2 of RFC 9393

> Note: This scheme has been documented by an IETF working group and is mentioned in an IETF Standard specification. However, as it describes a locally scoped, limited-purpose form of identification, it does not fully meet the requirements for permanent registration.
>
> As long as this specification (or any successors that describe this scheme) is a current IETF specification, this scheme should be considered to be "in use" and not considered for removal from the registry.

6.7. CoSWID Model for Use in SWIMA Registration

"Software Inventory Message and Attributes (SWIMA) for PA-TNC" [RFC8412] defines a standardized method for collecting an endpoint device's software inventory. A CoSWID can provide evidence of software installation that can then be used and exchanged with SWIMA. This registration adds a new entry to the IANA "Software Data Model Types" registry defined by [RFC8412] and [IANA.pa-tnc-parameters] to support CoSWID use in SWIMA as follows:

Pen: 0

Integer: 2

Name: Concise Software Identifier (CoSWID)

Reference: RFC 9393

Deriving Software Identifiers: A Software Identifier generated from a CoSWID tag is expressed as a concatenation of the form used in [RFC5234] as follows --

```
TAG_CREATOR_REGID "_" "_" UNIQUE_ID
```

where TAG_CREATOR_REGID is the reg-id item value of the tag's entity item having the role value of 1 (corresponding to "tag-creator"), and the UNIQUE_ID is the same tag's tag-id item. If the tag-id item's value is expressed as a 16-byte binary string, the UNIQUE_ID **MUST** be represented using the UUID string representation defined in [RFC4122], including the "urn:uuid:" prefix.

The TAG_CREATOR_REGID and the UNIQUE_ID are connected with a double underscore (_), without any other connecting character or whitespace.

7. Signed CoSWID Tags

SWID tags, as defined in the ISO-19770-2:2015 XML Schema, can include cryptographic signatures to protect the integrity of the SWID tag. In general, tags are signed by the tag creator (typically, although not exclusively, the vendor of the software component that the SWID tag identifies). Cryptographic signatures can make any modification of the tag detectable, which is especially important if the integrity of the tag is important, such as when the tag is providing RIMs for files. The ISO-19770-2:2015 XML Schema uses XML Digital Signatures (XMLDSIG) to support cryptographic signatures.

Signing CoSWID tags follows the procedures defined in CBOR Object Signing and Encryption (COSE) [RFC9052]. A CoSWID tag **MUST** be wrapped in a COSE Signature structure, either COSE_Sign1 or COSE_Sign. In the first case, a Single Signer Data Object (COSE_Sign1) contains a single signature and **MUST** be signed by the tag creator. The following CDDL specification defines a restrictive subset of COSE header parameters that **MUST** be used in the protected header in this case.

```
<CODE BEGINS> file "sign1.cddl"

COSE_Sign1-coswid<payload> = [
    protected: bstr .cbor protected-signed-coswid-header,
    unprotected: unprotected-signed-coswid-header,
    payload: bstr .cbor payload,
    signature: bstr,
]

cose-label = int / tstr
cose-values = any

protected-signed-coswid-header = {
    1 => int,                       ; algorithm identifier
    3 => "application/swid+cbor",
    * cose-label => cose-values,
}

unprotected-signed-coswid-header = {
    * cose-label => cose-values,
}

<CODE ENDS>
```

The COSE_Sign structure allows for more than one signature, one of which **MUST** be issued by the tag creator, to be applied to a CoSWID tag and **MAY** be used. The corresponding usage scenarios are domain specific and require well-specified application guidance.

```
<CODE BEGINS> file "sign.cddl"

COSE_Sign-coswid<payload> = [
    protected: bstr .cbor protected-signed-coswid-header1,
    unprotected: unprotected-signed-coswid-header,
    payload: bstr .cbor payload,
    signature: [ * COSE_Signature ],
]

protected-signed-coswid-header1 = {
    3 => "application/swid+cbor",
    * cose-label => cose-values,
}

protected-signature-coswid-header = {
    1 => int,                       ; algorithm identifier
    * cose-label => cose-values,
}

unprotected-signed-coswid-header = {
    * cose-label => cose-values,
}

COSE_Signature =  [
    protected: bstr .cbor protected-signature-coswid-header,
    unprotected: unprotected-signed-coswid-header,
    signature: bstr
]

<CODE ENDS>
```

Additionally, the COSE header countersignature **MAY** be used as an attribute in the unprotected header map of the COSE envelope of a CoSWID [RFC9338]. The application of countersigning enables second parties to provide a signature on a signature allowing for proof that a signature existed at a given time (i.e., a timestamp).

A CoSWID **MUST** be signed, using the above mechanism, to protect the integrity of the CoSWID tag. See Section 9 ("Security Considerations") for more information on why a signed CoSWID is valuable in most cases.

8. CBOR-Tagged CoSWID Tags

This specification allows for tagged and untagged CBOR data items that are CoSWID tags. Consequently, the CBOR tag defined by this document (Table 21) for CoSWID tags **SHOULD** be used in conjunction with CBOR data items that are CoSWID tags. Other CBOR tags **MUST NOT** be used with a CBOR data item that is a CoSWID tag. If tagged, both signed and unsigned CoSWID tags **MUST** use the CoSWID CBOR tag. If a signed CoSWID is tagged, a CoSWID CBOR tag **MUST** be appended before the COSE envelope, whether it is a COSE_Untagged_Message or a COSE_Tagged_Message. If an unsigned CoSWID is tagged, a CoSWID CBOR tag **MUST** be appended before the CBOR data item that is the CoSWID tag.

```
<CODE BEGINS> file "tags.cddl"

coswid = unsigned-coswid / signed-coswid
unsigned-coswid = concise-swid-tag / tagged-coswid<concise-swid-tag>
signed-coswid1 = signed-coswid-for<unsigned-coswid>
signed-coswid = signed-coswid1 / tagged-coswid<signed-coswid1>

tagged-coswid<T> = #6.1398229316(T)

signed-coswid-for<payload> = #6.18(COSE_Sign1-coswid<payload>)
    / #6.98(COSE_Sign-coswid<payload>)

<CODE ENDS>
```

This specification allows for a CBOR-tagged CoSWID tag to reside in a COSE envelope that is also tagged with a CoSWID CBOR tag. In cases where a tag creator is not a signer (e.g., hand-offs between entities in a trusted portion of a supply chain), retaining CBOR tags attached to unsigned CoSWID tags can be of great use. Nevertheless, redundant use of tags **SHOULD** be avoided when possible.

9. Security Considerations

The following security considerations for the use of CoSWID tags focus on:

- ensuring the integrity and authenticity of a CoSWID tag
- the application of CoSWID tags to address security challenges related to unmanaged or unpatched software
- reducing the potential for unintended disclosure of a device's software load

A tag is considered "authoritative" if the CoSWID tag was created by the software provider. An authoritative CoSWID tag contains information about a software component provided by the supplier of the software component, who is expected to be an expert in their own software. Thus, authoritative CoSWID tags can represent authoritative information about the software component. The degree to which this information can be trusted depends on the tag's chain of custody and the ability to verify a signature provided by the supplier if present in the CoSWID tag. The provisioning and validation of CoSWID tags are handled by local policy and are outside the scope of this document.

A signed CoSWID tag (see Section 7) whose signature has been validated can be relied upon to be unchanged since the time at which it was signed. By contrast, the data contained in unsigned tags can be altered by any user or process with write access to the tag. To support signature validation, there is a need to associate the right key with the software provider or party originating the signature in a secure way. This operation is application specific and needs to be addressed by the application or a user of the application; a specific approach for this topic is out of scope for this document.

When an authoritative tag is signed, the originator of the signature can be verified. A trustworthy association between the signature and the originator of the signature can be established via trust anchors. A certification path between a trust anchor and a certificate, including a public key enabling the validation of a tag signature, can realize the assessment of trustworthiness of an authoritative tag. Verifying that the software provider is the signer is a different matter. This requires verifying that the party that signed the tag is the same party given in the software-creator role of the tag's entity item. No mechanism is defined in this document to make this association; therefore, this association will need to be handled by local policy. As always, the validity of a signature does not imply the veracity of the signed statements: anyone can sign assertions such that the software is from a specific software-creator or that a specific persistent-id applies; policy needs to be applied to evaluate these statements and to determine their suitability for a specific use.

Loss of control of signing credentials used to sign CoSWID tags would cast doubt on the authenticity and integrity of any CoSWID tags signed using the compromised keys. In such cases, the legitimate tag signer (namely, the software provider for an authoritative CoSWID tag) can employ uncompromised signing credentials to create a new signature on the original tag. The tag's version number would not be incremented, since the tag itself was not modified. Consumers of CoSWID tags would need to validate the tag using the new credentials and would also need to make use of revocation information available for the compromised credentials to avoid validating tags signed with them. The process for doing this is beyond the scope of this specification.

The CoSWID format allows the use of hash values without an accompanying hash algorithm identifier. This exposes the tags to some risk of cross-algorithm attacks. We believe that this can become a practical problem only if some implementations allow the use of insecure hash algorithms. Since it may not become known immediately when an algorithm becomes insecure, this leads to a strong recommendation to only include support for hash algorithms that are generally considered secure, and not just marginally so.

CoSWID tags are intended to contain public information about software components and, as such, the contents of a CoSWID tag (as opposed to the set of tags that apply to the endpoint; see below) do not need to be protected against unintended disclosure on an endpoint. Conversely, generators of CoSWID tags need to ensure that only public information is disclosed. The entitlement-key item is an example of information for which particular care is required; tag authors are advised not to record unprotected, private software license keys in this field.

CoSWID tags are intended to be easily discoverable by authorized applications and users on an endpoint in order to make it easy to determine the tagged software load. Access to the collection of an endpoint's CoSWID tags needs to be limited to authorized applications and users using an appropriate access control mechanism.

Since the tag-id of a CoSWID tag can be used as a global index value, failure to ensure the tag-id's uniqueness can cause collisions or ambiguity in CoSWID tags that are retrieved or processed using this identifier. CoSWID is designed to not require a registry of identifiers. As a result, CoSWID requires the tag creator to employ a method of generating a unique tag identifier. Specific methods of generating a unique identifier are beyond the scope of this specification. A

collision in tag-ids may result in false positives/negatives in software integrity checks or misidentification of installed software, undermining CoSWID use cases such as vulnerability identification, software inventory, etc. If such a collision is detected, then the tag consumer may want to contact the maintainer of the CoSWID to have them issue a correction addressing the collision; however, this also discloses to the maintainer that the consumer has the other tag with the given tag-id in their database. More generally speaking, a tag consumer needs to be robust against such collisions lest the collision become a viable attack vector.

CoSWID tags are designed to be easily added and removed from an endpoint along with the installation or removal of software components. On endpoints where the addition or removal of software components is tightly controlled, the addition or removal of CoSWID tags can be similarly controlled. On more open systems, where many users can manage the software inventory, CoSWID tags can be easier to add or remove. On such systems, it can be possible to add or remove CoSWID tags in a way that does not reflect the actual presence or absence of corresponding software components. Similarly, not all software products automatically install CoSWID tags, so products can be present on an endpoint without providing a corresponding CoSWID tag. As such, any collection of CoSWID tags cannot automatically be assumed to represent either a complete or fully accurate representation of the software inventory of the endpoint. However, especially on endpoint devices that more strictly control the ability to add or remove applications, CoSWID tags are an easy way to provide a preliminary understanding of that endpoint's software inventory.

As CoSWID tags do not expire, inhibiting new CoSWID tags from reaching an intended consumer would render that consumer stuck with outdated information, potentially leaving associated vulnerabilities or weaknesses unmitigated. Therefore, a CoSWID tag consumer should actively check for updated tag-versions via more than one means.

This specification makes use of relative paths (e.g., filesystem paths) in several places. A signed CoSWID tag cannot make use of these to derive information that is considered to be covered under the signature. Typically, relative filesystem paths will be used to identify targets for an installation, not sources of tag information.

Any report of an endpoint's CoSWID tag collection provides information about the software inventory of that endpoint. If such a report is exposed to an attacker, this can tell them which software products and versions thereof are present on the endpoint. By examining this list, the attacker might learn of the presence of applications that are vulnerable to certain types of attacks. As noted earlier, CoSWID tags are designed to be easily discoverable by authorized applications and users on an endpoint, but this does not present a significant risk, since an attacker would already need to have access to the endpoint to view that information. However, when the endpoint transmits its software inventory to another party or that inventory is stored on a server for later analysis, this can potentially expose this information to attackers who do not yet have access to the endpoint. For this reason, it is important to protect the confidentiality of CoSWID tag information that has been collected from an endpoint in transit and at rest, not because those tags individually contain sensitive information but because the collection of CoSWID tags and their association with an endpoint reveals information about that endpoint's attack surface.

Finally, both the ISO-19770-2:2015 XML Schema SWID definition and the CoSWID CDDL specification allow for the construction of "infinite" tags with link item loops or tags that contain malicious content with the intent of creating non-deterministic states during validation or processing of those tags. While software providers are unlikely to do this, CoSWID tags can be created by any party and the CoSWID tags collected from an endpoint could contain a mixture of tags created by vendors and tags not created by vendors. For this reason, a CoSWID tag might contain potentially malicious content. Input sanitization, loop detection, and signature verification are ways that implementations can address this concern.

More generally speaking, the Security Considerations sections of [RFC8949], [RFC9052], and [RFC9338] apply.

10. Privacy Considerations

As noted in Section 9, collected information about an endpoint's software load, such as what might be represented by an endpoint's CoSWID tag collection, could be used by attackers to identify vulnerable software. Collections of endpoint software information also can have privacy implications for users. The set of applications a user installs can provide clues regarding personal matters such as political affiliation, banking and investments, gender, sexual orientation, medical concerns, etc. While the collection of CoSWID tags on an endpoint wouldn't increase privacy risks (since a party able to view those tags could also view the applications themselves), if those CoSWID tags are gathered and stored in a repository somewhere, visibility into the repository now also provides visibility into a user's application collection. For this reason, not only do repositories of collected CoSWID tags need to be protected against collection by malicious parties but even authorized parties will need to be vetted and made aware of privacy responsibilities associated with having access to this information. Likewise, users should be made aware that their software inventories are being collected from endpoints. Furthermore, when collected and stored by authorized parties or systems, the inventory data needs to be protected as both security and privacy-sensitive information.

11. References

11.1. Normative References

[BCP178] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, June 2012.

<https://www.rfc-editor.org/info/bcp178>

[BCP26] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, June 2017.

<https://www.rfc-editor.org/info/bcp26>

[IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[IANA.media-types] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[IANA.named-information] IANA, "Named Information", <https://www.iana.org/assignments/named-information>.

[IANA.pa-tnc-parameters] IANA, "Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC) Parameters", <https://www.iana.org/assignments/pa-tnc-parameters>.

[IANA.uri-schemes] IANA, "Uniform Resource Identifier (URI) Schemes", <https://www.iana.org/assignments/uri-schemes>.

[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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

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

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/info/rfc5198>.

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

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

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

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[RFC8412] Schmidt, C., Haynes, D., Coffin, C., Waltermire, D., and J. Fitzgerald-McKay, "Software Inventory Message and Attributes (SWIMA) for PA-TNC", RFC 8412, DOI 10.17487/RFC8412, July 2018, <https://www.rfc-editor.org/info/rfc8412>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/info/rfc9052>.

[RFC9338] Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", STD 96, RFC 9338, DOI 10.17487/RFC9338, December 2022, <https://www.rfc-editor.org/info/rfc9338>.

[SAM] "Information technology - IT asset management - Part 5: Overview and vocabulary", ISO/IEC 19770-5:2015, August 2015, <https://www.iso.org/standard/68291.html>.

[SWID] "Information technology - IT asset management - Part 2: Software identification tag", ISO/IEC 19770-2:2015, October 2015, <https://www.iso.org/standard/65666.html>.

[UNSPSC] "United Nations Standard Products and Services Code", 2022, <https://www.unspsc.org/>.

[W3C.REC-mediaqueries-3-20220405] Rivoal, F., Ed., "Media Queries Level 3", W3C Recommendation REC-mediaqueries-3-20220405, 5 April 2022, <https://www.w3.org/TR/mediaqueries-3/>.

[W3C.REC-xmlschema-2-20041028] Biron, P. V., Ed. and A. Malhotra, Ed., "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation REC-xmlschema-2-20041028, 28 October 2004, <https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.

[W3C.REC-xpath20-20101214] Berglund, A., Ed., Boag, S., Ed., Chamberlin, D., Ed., Fernández, M. F., Ed., Kay, M., Ed., Robie, J., Ed., and J. Siméon, Ed., "XML Path Language (XPath) 2.0 (Second Edition)", W3C Recommendation REC-xpath20-20101214, 14 December 2010, <https://www.w3.org/TR/2010/REC-xpath20-20101214/>.

11.2. Informative References

[CamelCase] "Camel Case (upper camel case)", 18 December 2014, <http://wiki.c2.com/?CamelCase>.

[KebabCase] "Kebab Case", 29 August 2014, <http://wiki.c2.com/?KebabCase>.

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003, <https://www.rfc-editor.org/info/rfc3444>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[RFC8322] Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, <https://www.rfc-editor.org/info/rfc8322>.

[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>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/info/rfc9334>.

[SEMVER] Preston-Werner, T., "Semantic Versioning 2.0.0", <https://semver.org/spec/v2.0.0.html>.

[SWID-GUIDANCE] Waltermire, D., Cheikes, B. A., Feldman, L., and G. Witte, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags", NISTIR 8060, April 2016, <https://doi.org/10.6028/NIST.IR.8060>.

[X.1520] ITU-T, "Common vulnerabilities and exposures", ITU-T Recommendation X.1520, January 2014, <https://www.itu.int/rec/T-REC-X.1520>.

Acknowledgments

This document draws heavily on the concepts defined in the ISO/IEC 19770-2:2015 specification. The authors of this document are grateful for the prior work of the 19770-2 contributors.

We are also grateful for the careful reviews provided by the IESG reviewers. Special thanks go to Benjamin Kaduk.

Contributors

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Carsten Bormann contributed to the CDDL specifications and the IANA considerations.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Jessica Fitzgerald-McKay
National Security Agency
9800 Savage Road
Ft. Meade, Maryland 20755
United States of America
Email: jmfitz2@cyber.nsa.gov

Charles Schmidt
The MITRE Corporation
202 Burlington Road
Bedford, Massachusetts 01730
United States of America
Email: cmschmidt@mitre.org

David Waltermire
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, Maryland 20877
United States of America
Email: david.waltermire@nist.gov