

SACM Working Group  
Internet-Draft

Intended status: Informational National Institute of Standards and Techno  
Expires: November 4, 2017

D. Waltermire  
S. Banghart  
May 3, 2017

Definition of the ROLIE Software Descriptor Extension  
draft-banghart-sacm-rolie-softwaredescriptor-01

Abstract

This document extends the Resource-Oriented Lightweight Information Exchange (ROLIE) core to add the information type category and related requirements needed to support Software Record and Software Inventory use cases. The 'software-descriptor' information type is defined as a ROLIE extension. Additional supporting requirements are also defined that describe the use of specific formats and link relations pertaining to the new information type.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 4, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. New information-types . . . . .	3
3.1. The "software-descriptor" information type . . . . .	4
4. Usage of CSIRT Information Types in the Atom Publishing Protocol . . . . .	5
5. Usage of the software-descriptor Information Type in the atom:feed element . . . . .	5
6. Usage of the software-descriptor Information Type in an atom:entry . . . . .	5
6.1. Use of the atom:link element . . . . .	5
6.2. Use of the rolie:format element . . . . .	6
6.2.1. The ISO SWID 2016 format . . . . .	6
6.2.2. The Concise SWID format . . . . .	7
6.3. Use of the rolie:property element . . . . .	7
6.3.1. urn:ietf:params:rolie:property:swd:id . . . . .	7
6.3.2. urn:ietf:params:rolie:property:swd:swname . . . . .	7
6.4. IANA Considerations . . . . .	7
6.4.1. incident information-type . . . . .	7
6.4.2. swd:id property . . . . .	8
6.4.3. swd:swname property . . . . .	8
6.5. Security Considerations . . . . .	8
7. Normative References . . . . .	9
Appendix A. Schema . . . . .	9
Appendix B. Examples of Use . . . . .	9
Authors' Addresses . . . . .	10

## 1. Introduction

This document defines an extension to the Resource-Oriented Lightweight Information Exchange (ROLIE) protocol to support the publication of software descriptor information. Software descriptor information is information that characterizes:

an installable software package, or

information about static software components that may be installed by a software package or patch.

Software descriptor information includes identifying, versioning, software creation and publication, and file artifact information. Software descriptor information provides data about what might be

installed, but doesn't describe where or how a specific software installation is installed, configured, or executed.

Some possible use cases for Software descriptor information include:

Software providers can publish software descriptor information so that software researchers and users of software can understand the collection of software produced by a that software provider.

Organizations can aggregate and syndicate collections of software descriptor information provided by multiple software providers to support software-related analysis processes (e.g., vulnerability analysis) and value added information (e.g., software configuration checklist repositories) using identification and characterization information derived from software descriptor information.

End user organizations can consume sources of software descriptor information, and other related software vulnerability and configuration information to provide the data needed to automate software asset, patch, and configuration management practices.

Organizations can use software descriptors to support verification of other entities, thru mechanisms such as RIM or other integrity measurements.

This document supports these use cases by describing the content requirements for Collections of software descriptor information that are to be published to or retrieved from a ROLIE repository. This document also discusses requirements around the use of link relationships and describing the data model formats used in a ROLIE Entry describing a software descriptor information resource.

## 2. Terminology

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Definitions for some of the common computer security-related terminology used in this document can be found in Section 2 of [RFC5070].

## 3. New information-types

This document defines the following information type:

### 3.1. The "software-descriptor" information type

The "software-descriptor" information type represents any information that describes a piece of software. This document uses the definition of software provided by [RFC4949]. Note that as per this definition, this information type pertains to static software, that is, code on the disc. The software-descriptor information type is intended to provide a category for information that does one or more of the following:

identifies and characterizes software This software identification and characterization information can be provided by a large variety of data, but always describes software in a pre-installed state.

provides software installer metadata This represents information about software used to install other software. This metadata identifies, and characterizes a software installation package or media.

describes stateless installation metadata Information that describes the software post-deployment, such as files that may be deployed during an installation. It is expected that this metadata is produced generally for a given installation, and may not exactly match the actual installed files on a given endpoint.

Provided below is a non-exhaustive list of information that may be considered to be of a software-descriptor information type.

- o Naming information: IDs and names that aid in the identification of a piece of software
- o Version and patching information: Version numbers, patch identifiers, or other information that
- o Vendor and source information: Includes where the software was developed or distributed from, as well as where the software installation media may be located.
- o Payload and file information: information that describes or enumerates the files and folders that make up the piece of software, and information about those files.
- o Descriptive information and data: Any information that otherwise characterizes a piece of software, such as libraries, runtime environments, target OSES, intended purpose or audience, etc.

Note again that this list is not exhaustive, any information that in is the abstract realm of an incident should be classified under this information-type.

This information type does not include descriptions of running software, or state and configuration information that is associated with a software installation.

#### 4. Usage of CSIRT Information Types in the Atom Publishing Protocol

This document does not specify any additional requirements for use of the Atom Publishing Protocol.

#### 5. Usage of the software-descriptor Information Type in the atom:feed element

This document does not specify any additional requirements for use of the atom:feed element.

#### 6. Usage of the software-descriptor Information Type in an atom:entry

This document specifies the following requirements for use of the software-descriptor information type with regards to Atom Entries.

##### 6.1. Use of the atom:link element

This section defines the requirements around the use of atom:links in Entries. Each relationship should be named,described, and given a requirement level.

Name	Description	Conformance
ancestor	Links to a software descriptor resource that defines an ancestor of the software being described by this Entry.	MAY
patches	Links to a software descriptor resource that defines the software being patched by this software	MAY
requires	Links to a software descriptor resource that defines a piece of software required for this software to function properly.	MAY
installs	Links to a software descriptor resource that defines the software being installed by this software.	MAY
installationrecord	Provides a link to a resource that describes an installation of this software.	MAY

Table 1: Link Relations for Resource-Oriented Lightweight Indicator Exchange

## 6.2. Use of the rolie:format element

This document does not contain any additional requirements for the rolie:format element, the formats that follow are provided as examples of formats that describe the software descriptor information type.

### 6.2.1. The ISO SWID 2016 format

The ISO SWID Tag 2016 format is a software descriptor and software record data format. It provides several tags: primary, which provides descriptive and naming information about software, patch, which describes non-standalone software meant to patch existing software, and corpus, which describes the software installation media that installs a given piece of software.

For a more complete overview as well as normative requirements, refer to TODO(ref?):ISO/IEC 19770-2

### 6.2.2. The Concise SWID format

The Concise SWID format is an alternative representation of the ISO SWID Tag 2016 format using a CBOR encoding defined by a CDDL specification. It provides the same features and attributes as are specified in ISO 19770-2, plus:

- o a straight forward method to sign and encrypt SWID Tags using COSE, and
- o additional attributes that provide an improved structure to include file hashes intended to be used as Reference Integrity Measurements (RIM).

### 6.3. Use of the rolie:property element

This document registers new valid rolie:property names as follows:

#### 6.3.1. urn:ietf:params:rolie:property:swd:id

This property provides an exposure point for an identification field from the associated software descriptor. The value of this property SHOULD be uniquely identifying information generated from the software descriptor linked to by the entry's atom:content element. swd:id property values SHOULD have a one-to-one mapping to individual pieces of SWD content.

#### 6.3.2. urn:ietf:params:rolie:property:swd:sname

This property provides an exposure point for the plain text name of the software being described. Due to the great variance in naming schemes, this property should be considered informative.

### 6.4. IANA Considerations

#### 6.4.1. incident information-type

IANA has added an entry to the "ROLIE Security Resource Information Type Sub-Registry" registry located at <https://www.iana.org/assignments/rolie/category/information-type> .

The entry is as follows:

name: software-descriptor

index: TBD

reference: This document, Section 3.1

#### 6.4.2. swd:id property

IANA has added an entry to the "ROLIE URN Parameters" registry located in <https://www.iana.org/assignments/rolie/>.

The entry is as follows:

name: property:swd:id

Extension IRI: urn:ietf:params:rolie:property:swd:id

Reference: This document, Section 6.3.1

Subregistry: None

#### 6.4.3. swd:swname property

IANA has added an entry to the "ROLIE URN Parameters" registry located in <https://www.iana.org/assignments/rolie/>.

The entry is as follows:

name: property:swd:swname

Extension IRI: urn:ietf:params:rolie:property:swd:swname

Reference: This document, Section 6.3.2

Subregistry: None

#### 6.5. Security Considerations

Use of this extension implies dealing with the security implications of both ROLIE and of software descriptors in general. As with any SWD information, care should be taken to verify the trustworthiness and veracity of the descriptor information to the fullest extent possible.

Ideally, software descriptors should have been signed by the software manufacturer, or signed by whichever agent processed the source code. SWD documents from these sources are more likely to be accurate than those generated by scraping installed software.

These "authoritative" sources of SWD content should consider additional security for their ROLIE repository beyond the typical recommendations, as the central importance of the repository is likely to make it a target.



Version information is often represented differently across manufacturers and even across product releases. If using SWD version information for low fault tolerance comparisons and searches, care should be taken that the correct version scheme is being utilized.

## 7. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5070] Danyliw, R., Meijer, J., and Y. Demchenko, "The Incident Object Description Exchange Format", RFC 5070, DOI 10.17487/RFC5070, December 2007, <<http://www.rfc-editor.org/info/rfc5070>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<http://www.rfc-editor.org/info/rfc4949>>.

## Appendix A. Schema

This document does not require any schema extensions.

## Appendix B. Examples of Use

Use of this extension in a ROLIE repository will not typically change that repo's operation. As such, the general examples provided by the ROLIE core document would serve as examples. Provided below is a sample SWD ROLIE entry:

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
  xmlns:rolie="urn:ietf:params:xml:ns:rolie-1.0">
  <id>dd786dba-88e6-440b-9158-b8fae67ef67c</id>
  <title>Sample Software Descriptor</title>
  <published>2015-08-04T18:13:51.0Z</published>
  <updated>2015-08-05T18:13:51.0Z</updated>
  <summary>A descriptor for a piece of software published by this
  organization. </summary>
  <link rel="self" href="http://www.example.org/provider/SWD/123456"/>
  <category
    scheme="urn:ietf:params:rolie:category:information-type"
    term="software-descriptor"/>
  <rolie:format ns="urn:example:COSWID"/>
  <content type="application/xml"
    src="http://www.example.org/provider/SWD/123456/data"/>
</entry>
```

#### Authors' Addresses

David Waltermire  
National Institute of Standards and Technology  
100 Bureau Drive  
Gaithersburg, Maryland 20877  
USA

Email: david.waltermire@nist.gov

Stephen Banghart  
National Institute of Standards and Technology  
100 Bureau Drive  
Gaithersburg, Maryland 20877  
USA

Email: stephen.banghart@nist.gov

SACM Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: 8 September 2022

H. Birkholz  
Fraunhofer SIT  
J. Fitzgerald-McKay  
National Security Agency  
C. Schmidt  
The MITRE Corporation  
D. Waltermire  
NIST  
7 March 2022

Concise Software Identification Tags  
draft-ietf-sacm-coswid-21

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 similar set of semantics and features as 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 Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1.	Introduction . . . . .	3
1.1.	The SWID and CoSWID Tag Lifecycle . . . . .	4
1.2.	Concise SWID Format . . . . .	8
1.3.	Requirements Notation . . . . .	8
2.	Concise SWID Data Definition . . . . .	8
2.1.	Character Encoding . . . . .	10
2.2.	Concise SWID Extensions . . . . .	10
2.3.	The concise-swid-tag Map . . . . .	13
2.4.	concise-swid-tag Co-Constraints . . . . .	18
2.5.	The global-attributes Group . . . . .	18
2.6.	The entity-entry Map . . . . .	19
2.7.	The link-entry Map . . . . .	21
2.8.	The software-meta-entry Map . . . . .	25
2.9.	The Resource Collection Definition . . . . .	28
2.9.1.	The hash-entry Array . . . . .	28
2.9.2.	The resource-collection Group . . . . .	28
2.9.3.	The payload-entry Map . . . . .	32
2.9.4.	The evidence-entry Map . . . . .	32
2.10.	Full CDDL Specification . . . . .	33
3.	Determining the Type of CoSWID . . . . .	39
4.	CoSWID Indexed Label Values . . . . .	40
4.1.	Version Scheme . . . . .	40
4.2.	Entity Role Values . . . . .	42
4.3.	Link Ownership Values . . . . .	44
4.4.	Link Rel Values . . . . .	44
4.5.	Link Use Values . . . . .	46
5.	URI Schemes . . . . .	47
5.1.	"swid" URI Scheme . . . . .	47
5.2.	"swidpath" URI Scheme . . . . .	48
6.	IANA Considerations . . . . .	49
6.1.	CoSWID Items Registry . . . . .	49
6.2.	Software Tag Values Registries . . . . .	52
6.2.1.	Registration Procedures . . . . .	52
6.2.2.	Private Use of Index and Name Values . . . . .	52
6.2.3.	Expert Review Criteria . . . . .	53
6.2.4.	Software Tag Version Scheme Values Registry . . . . .	53
6.2.5.	Software Tag Entity Role Values Registry . . . . .	55

6.2.6. Software Tag Link Ownership Values Registry . . . . .	56
6.2.7. Software Tag Link Relationship Values Registry . . . . .	57
6.2.8. Software Tag Link Use Values Registry . . . . .	60
6.3. swid+cbor Media Type Registration . . . . .	61
6.4. CoAP Content-Format Registration . . . . .	62
6.5. CBOR Tag Registration . . . . .	62
6.6. URI Scheme Registrations . . . . .	62
6.6.1. URI-scheme swid . . . . .	63
6.6.2. URI-scheme swidpath . . . . .	63
6.7. CoSWID Model for use in SWIMA Registration . . . . .	64
7. Signed CoSWID Tags . . . . .	64
8. CBOR-Tagged CoSWID Tags . . . . .	67
9. Security Considerations . . . . .	67
10. Privacy Consideration . . . . .	71
11. Change Log . . . . .	72
12. References . . . . .	77
12.1. Normative References . . . . .	77
12.2. Informative References . . . . .	80
Acknowledgments . . . . .	81
Contributors . . . . .	81
Authors' Addresses . . . . .	82

## 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, a part of a Software Asset Management [SAM] process, 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 measurements (RIM) and Attester-produced event logs that complement attestation Evidence [I-D.ietf-rats-architecture].

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 SWID Tags [SWID-GUIDANCE] defines four types of SWID tags: primary, patch, corpus, and supplemental. The following text is paraphrased from these sources.

1. 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.
2. 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.
3. 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.
4. 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, which also provides normative language for CoSWID semantics that implement this lifecycle. The following information helps to explain how these semantics apply to 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.

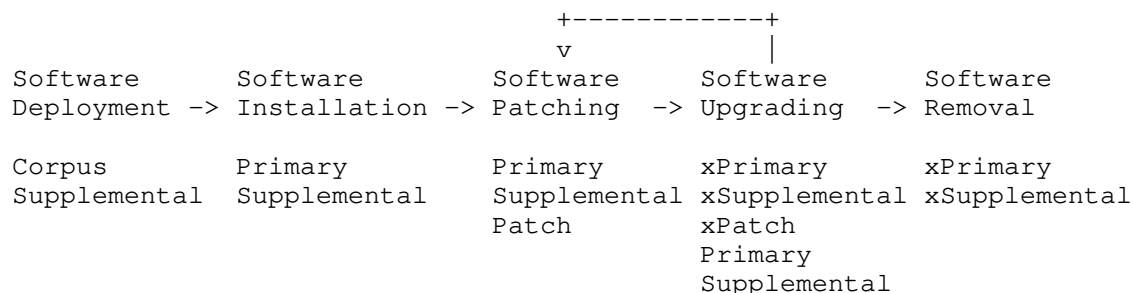


Figure 1: Use of Tag Types in the Software Lifecycle

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 tags 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 that 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. A new patch tag is provided, when a patch is applied to the software component, 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 which 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, but the version number or the descriptive metadata of an installed software component are 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 of the different SWID and CoSWID tag types provide different sets of information. For example, a "corpus tag" is used to describe a software component's installation image on an installation media, 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 to 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, are 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 is 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 [CamelCase] notation 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 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 corresponds 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 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 indexes 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 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 that string is unknown). In the rest of the 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 1 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 2 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 consistent with the Net-Unicode definition defined in [RFC5198].

All names registered with IANA according to 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" in the SWID model. These are covered in Section 2.4, Paragraph 2.

- \* The inclusion of extension points in the CoSWID specification using CDDL sockets (see [RFC8610] Section 3.9). The use of CDDL sockets allow 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. This extension mechanism can also be used to update the CoSWID format as revisions to ISO-19770-2 are published.

The following CDDL sockets (extension points) are defined in this document, which 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
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 CoSWID value registries are also defined in Section 6.2 that 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 and 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 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 2.4, Paragraph 2.
- \* **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 as defined by [RFC4122]. There are no strict guidelines on how the identifier is structured, but examples include a 16-byte 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 **MUST NOT** contain a sequence of two underscores ("\_\_", see Section 6.7).
- \* **tag-version (index 12)**: An integer value that indicate 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.
- \* **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.
- \* `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".
- \* `software-name` (index 1): This textual item 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].
- \* `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 (Section 2, for the "Version Scheme" registry 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 Tag Version Scheme Values" registry (see Section 6.2.4).
- \* `media` (index 10): This text value is 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 Recommendation (see [W3C.REC-css3-mediaqueries-20120619]). Support for media queries are 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.

- \* `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. 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].
- \* `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.
- \* `link` (index 4): Provides a means to establish relationship arcs between the tag and another items. 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, ROLIE feed [RFC8322], MUD resource [RFC8520], software download location, etc). This is modeled after the HTML "link" element. Described in Section 2.7.
- \* `payload` (index 6): This item 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.
- \* `evidence` (index 3): This item 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.
- \* `$$coswid-extension`: This CDDL socket 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. If any of these constraints is not met, a signed tag cannot be used anymore as a signed statement.

- \* The patch and supplemental items MUST NOT both be set to "true".
- \* If the patch item is set to "true", the tag SHOULD 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 the supplemental item is set to "true", the tag SHOULD contain at least one link item with both the rel item value of "supplemental" and an href item specifying an association with the software that is supplemented. Without at least one link item the target of supplement tag 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. This ensures that primary and corpus tags have an identifiable software version.

#### 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 follows:

```
global-attributes = (  
    ? lang => text,  
    * any-attribute,  
)  
  
any-attribute = (  
    label => one-or-more<text> / one-or-more<int>  
)  
  
label = text / int
```

The following describes each child item of this group.

- \* lang (index 15): A textual language tag that conforms with 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: This sub-group 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 describes each child item of this group.

- \* `global-attributes`: The `global-attributes` group described in Section 2.4, Paragraph 2.
- \* `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): The registration id 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 RFC 3986 URI; 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 Tag 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): The value of the thumbprint item provides a hash (i.e. the thumbprint) of the signing entity's public key certificate. This 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`: This CDDL socket 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 /= -356..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 describes each member of this map.

- \* `global-attributes`: The `global-attributes` group described in Section 2.4, Paragraph 2.
- \* `artifact` (index 37): To be used with `rel="installation-media"`, 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 relative to the base URI of the CoSWID tag, i.e., the URI under which the CoSWID tag was provided. For example, `"/folder/supplemental.coswid"`.
  - a physical resource location with any acceptable URI scheme (e.g., `file://` `http://` `https://` `ftp://`)
  - a URI with `"swid:"` as the scheme refers to another SWID or CoSWID by the referenced tag's tag-id. This URI needs to be resolved in the context of the endpoint by software that can lookup other SWID or CoSWID tags. For example, `"swid:2df9de35-0aff-4a86-ace6-f7dddlade4c"` references the tag with the tag-id value `"2df9de35-0aff-4a86-ace6-f7dddlade4c"`.



- a URI 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.
- \* media (index 10): A hint to the consumer of the link to what target platform the link is applicable to. This item represents a query as defined by the W3C Media Queries Recommendation (see [W3C.REC-css3-mediaqueries-20120619]). As highlighted in media defined in Section 2.3, support for media queries are 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.
- \* ownership (index 39): An integer or textual value (integer label with text escape, see Section 2, for the "Software Tag Link Ownership Values" registry Section 4.3) 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 Tag Link Ownership Values" registry.

- \* `rel` (index 40): An integer or textual value that (integer label with text escape, see Section 2, for the "Software Tag Link Relationship Values" registry Section 4.3) 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 Tag 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: <https://www.iana.org/assignments/link-relations/link-relations.xhtml> as defined by [RFC8288]. When a string value defined in the IANA "Software Tag Link Relationship Values" registry matches a Relation Name defined in the IANA "Link Relation Types" registry, the index value in the IANA "Software Tag 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 Tag Link Relationship Values" registry.
- \* `media-type` (index 41): A link can point to arbitrary resources on the endpoint, local network, or Internet using the `href` item. Use of this item supplies the resource consumer with a hint of what type of resource to expect. (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: <http://www.iana.org/assignments/media-types/media-types.xhtml>. 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, for the "Software Tag Link Relationship Values" registry Section 4.3) 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 "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 Tag Link Use Values" registry.

- \* `$$link-extension`: This CDDL socket 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 describes each child item of this group.

- \* `global-attributes`: The global-attributes group described in Section 2.4, Paragraph 2.

- \* `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, 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 similar value that are 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 (i.e., 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 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 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 server, 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. 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 UNSPSC classification code for the software component as defined by the United Nations Standard Products and Services Code (UNSPSC, [UNSPSC]).
- \* `unspsc-version` (index 57): The version of UNSPSC used to define the `unspsc-code` value.
- \* `$$meta-extension`: This CDDL socket 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] using the hash member (index 7) and the corresponding hash-entry type. 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, e.g., 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

A list of items both used in 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,  
}  
  
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 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 described in Section 2.4, Paragraph 2.
- \* directory (index 16): A directory item allows child directory and file items to be defined within a directory hierarchy for the software component.
- \* file (index 17): A file item allows details about a file to be provided for the software component.
- \* process (index 18): A process item 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): A resource item 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 under specified. As a result, this item might be further defined through extension 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].
- \* hash (index 7): A hash of the file as described in Section 2.9.1.
- \* 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 relative to the location of the parent directory item (preferred), or relative to the location of the CoSWID tag (as indicated in the location value in the evidence entry map) 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): This group 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.
- \* `$$resource-collection-extension`: This CDDL socket can be used to extend the resource-collection group model. This can be used to add new specialized types of resources. See Section 2.2.
- \* `$$file-extension`: This CDDL socket can be used to extend the file-entry group model. See Section 2.2.
- \* `$$directory-extension`: This CDDL socket can be used to extend the directory-entry group model. See Section 2.2.
- \* `$$process-extension`: This CDDL socket can be used to extend the process-entry group model. See Section 2.2.
- \* `$$resource-extension`: This CDDL socket 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 describes each child item of this group.

- \* `global-attributes`: The global-attributes group described in Section 2.4, Paragraph 2.
- \* `resource-collection`: The resource-collection group described in Section 2.9.2.
- \* `$$payload-extension`: This CDDL socket 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 describes each child item of this group.

- \* **global-attributes**: The global-attributes group described in Section 2.4, Paragraph 2.
- \* **resource-collection**: The resource-collection group described in Section 2.9.2.
- \* **date (index 35)**: The date and time the information was collected pertaining to the evidence item.
- \* **device-id (index 36)**: The endpoint's string identifier from which the evidence was collected.
- \* **location (index 23)**: The absolute filepath of the location of the CoSWID tag generated as evidence. (Location values in filesystem-items in the payload can be expressed relative to this location.)
- \* **\$\$evidence-extension**: This CDDL socket 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>  
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 /= licenser
$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..64436 / 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 indexes
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 indexes
multipartnumeric = 1
multipartnumeric-suffix = 2
alphanumeric = 3
decimal = 4
semver = 16384
```



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

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

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

; "use" integer indexes
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.

1. Primary Tag: A CoSWID tag MUST be considered a primary tag if the corpus, patch, and supplemental items are "false".
2. Supplemental Tag: A CoSWID tag MUST be considered a supplemental tag if the supplemental item is set to "true".

3. Corpus Tag: A CoSWID tag MUST be considered a corpus tag if the corpus item is "true".
4. Patch Tag: A CoSWID tag MUST be considered a patch tag if the patch item is "true".

Note: Multiple of the corpus, patch, and supplemental items can 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 a number of kinds of indexed label values that are maintained in a registry each (Section 6). 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. Version Scheme Name strings match the version schemes defined in the ISO/IEC 19770-2:2015 [SWID] specification. Index value indicates the value to use as the version-scheme item's value. The Version Scheme Name provides human-readable text for the value. The Definition 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 a single floating point number (e.g., 1.25 is less than 1.3).

The values above are registered in the IANA "Software Tag 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, the "decimal" version scheme SHOULD be assumed.
- \* "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, the "semver" version scheme SHOULD be assumed.
- \* "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, the other version scheme SHOULD be assumed instead of "alphanumeric".

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 [SWID] specification. The "Index" value indicates the value to use as the role item's value. The "Role Name" provides human-readable text for the value. The "Definition" 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 "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 Tag 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 [SWID] specification. The "Index" value indicates the value to use as the link-entry group ownership item's value. The "Ownership Type" provides human-readable text for the value. The "Definition" 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 sharing the software.

Table 5: Link Ownership Values

The values above are registered in the IANA "Software Tag 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 [SWID] specification. The "Index" value indicates the value to use as the link-entry group ownership item's value. The "Relationship Type" provides human-readable text for the value. The "Definition" 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.
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 another software 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 Tag 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 [SWID] specification. The "Index" value indicates the value to use as the link-entry group use item's value. The "Use Type" provides human-readable text for the value. The "Definition" 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 Tag Link Use Values" registry defined in Section 6.2.8. Additional values will likely be registered over time.

## 5. URI Schemes

This specification defines the following URI schemes for use in CoSWID and to provide interoperability with schemes used in [SWID].

Note: These URI schemes are used in [SWID] without an IANA registration. The present specification ensures that these URI schemes are properly defined going forward.

// RFC Ed.: throughout this section, please replace RFC-AAAA with the  
// RFC number of this specification and remove this note.

### 5.1. "swid" URI Scheme

There is a need for a scheme name that can be used in URIs that point to a specific software tag by that tag's tag-id, such as the use of the link entry as described in Section 2.7. Since this scheme is used both in a standards track document and an ISO standard, this scheme needs to be used without fear of conflicts with current or future actual schemes. In Section 6.6.1, the scheme "swid" is registered as a 'permanent' scheme for that purpose.

URIs 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 URIs with this scheme.

For URIs 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 [RFC3986] Section 2.1.

The following expression is a valid example:

```
swid:2df9de35-0aff-4a86-ace6-f7dddddade4c
```

## 5.2. "swidpath" URI Scheme

There is a need for a scheme name that can be used in URIs to identify a collection of specific software tags with data elements that match an XPath expression, such as the use of the link entry as described in Section 2.7. The scheme named "swidpath" is used for this purpose in [SWID], but not registered. To enable usage without fear of conflicts with current or future actual schemes, the present document registers it as a 'permanent' scheme for that purpose (see Section 6.6.2).

URIs 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 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 URI" 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 a URI 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 [RFC3986] Section 2.1.
- \* 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, 6 new registries are established with this request, with initial entries provided for each registry. New values for 5 other registries are also requested.

### 6.1. CoSWID Items Registry

This registry uses integer values as index values in CBOR maps.

This document defines a new registry titled "CoSWID Items". 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	Specification
0	tag-id	RFC-AAAA
1	software-name	RFC-AAAA
2	entity	RFC-AAAA
3	evidence	RFC-AAAA
4	link	RFC-AAAA
5	software-meta	RFC-AAAA
6	payload	RFC-AAAA

7	hash	RFC-AAAA
8	corpus	RFC-AAAA
9	patch	RFC-AAAA
10	media	RFC-AAAA
11	supplemental	RFC-AAAA
12	tag-version	RFC-AAAA
13	software-version	RFC-AAAA
14	version-scheme	RFC-AAAA
15	lang	RFC-AAAA
16	directory	RFC-AAAA
17	file	RFC-AAAA
18	process	RFC-AAAA
19	resource	RFC-AAAA
20	size	RFC-AAAA
21	file-version	RFC-AAAA
22	key	RFC-AAAA
23	location	RFC-AAAA
24	fs-name	RFC-AAAA
25	root	RFC-AAAA
26	path-elements	RFC-AAAA
27	process-name	RFC-AAAA
28	pid	RFC-AAAA
29	type	RFC-AAAA
30	Unassigned	

31	entity-name	RFC-AAAA
32	reg-id	RFC-AAAA
33	role	RFC-AAAA
34	thumbprint	RFC-AAAA
35	date	RFC-AAAA
36	device-id	RFC-AAAA
37	artifact	RFC-AAAA
38	href	RFC-AAAA
39	ownership	RFC-AAAA
40	rel	RFC-AAAA
41	media-type	RFC-AAAA
42	use	RFC-AAAA
43	activation-status	RFC-AAAA
44	channel-type	RFC-AAAA
45	colloquial-version	RFC-AAAA
46	description	RFC-AAAA
47	edition	RFC-AAAA
48	entitlement-data-required	RFC-AAAA
49	entitlement-key	RFC-AAAA
50	generator	RFC-AAAA
51	persistent-id	RFC-AAAA
52	product	RFC-AAAA
53	product-family	RFC-AAAA
54	revision	RFC-AAAA

55	summary	RFC-AAAA	
+-----+	+-----+	+-----+	+-----+
56	unspsc-code	RFC-AAAA	
+-----+	+-----+	+-----+	+-----+
57	unspsc-version	RFC-AAAA	
+-----+	+-----+	+-----+	+-----+
58-4294967295	Unassigned		
+-----+	+-----+	+-----+	+-----+

Table 9: CoSWID Items Initial Registrations

## 6.2. Software Tag Values Registries

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 nor 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 "swid" (rather than "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 [BCP26] in Section 4.1. This allows values -1 to -24 to be expressed as a single uint\_8t in CBOR, and values -25 to -256 to be expressed using an additional uint\_8t 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 an 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". Use of 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 the use of the entry's index value or name, and 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 Tag Version Scheme Values Registry

This document establishes a new registry titled "Software Tag 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].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/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: CoSWID 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 Tag Version Scheme Values" registry are provided below, which are derived from the textual version scheme names defined in [SWID].

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

Table 11: CoSWID 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 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 Tag Entity Role Values Registry

This document establishes a new registry titled "Software Tag 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].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/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: CoSWID 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 Tag Entity Role Values" registry are provided below, which are derived from the textual entity role names defined in [SWID].

Index	Role Name	Specification
0	Reserved	
1	tagCreator	See Section 4.2
2	softwareCreator	See Section 4.2
3	aggregator	See Section 4.2
4	distributor	See Section 4.2
5	licensor	See Section 4.2
6	maintainer	See Section 4.2
7-255	Unassigned	

Table 13: CoSWID Entity Role Initial Registrations

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

#### 6.2.6. Software Tag Link Ownership Values Registry

This document establishes a new registry titled "Software Tag 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].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/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: CoSWID 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 Tag Link Ownership Values" registry are provided below, which are derived from the textual entity role names defined in [SWID].

Index	Ownership Type Name	Definition
0	Reserved	
1	abandon	See Section 4.3
2	private	See Section 4.3
3	shared	See Section 4.3
4-255	Unassigned	

Table 15: CoSWID Link Ownership Initial Registrations

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

#### 6.2.7. Software Tag Link Relationship Values Registry

This document establishes a new registry titled "Software Tag 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].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/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: CoSWID 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 Tag Link Relationship Values" registry are provided below, which are derived from the link relationship values defined in [SWID].

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

Table 17: CoSWID 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 Tag Link Use Values Registry

This document establishes a new registry titled "Software Tag 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].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/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: CoSWID 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 Tag Link Use Values" registry are provided below, which are derived from the link relationship values defined in [SWID].

Index	Link Use Type Name	Specification
0	Reserved	
1	optional	See Section 4.5
2	required	See Section 4.5
3	recommended	See Section 4.5
4-255	Unassigned	

Table 19: CoSWID 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 is requested to add the following to the IANA "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-AAAA for details.

Security considerations: See Section 9 of RFC-AAAA.

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-AAAA

Applications that use this media type: The type is used by software asset management systems, vulnerability assessment systems, and 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-AAAA, there is no fragment identification syntax defined for "application/cbor".)

Additional information:

Magic number(s): if tagged, first five bytes in hex: da 53 57 49 44 (see Section 8 in RFC-AAAA)

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

Media type	Encoding	ID	Reference
application/swid+cbor	-	TBD1	RFC-AAAA

Table 20: CoAP Content-Format IDs

#### 6.5. CBOR Tag Registration

IANA is requested to allocate a tag in the "CBOR Tags" registry [IANA.cbor-tags], preferably with the specific value requested:

Tag	Data Item	Semantics
1398229316	map	Concise Software Identifier (CoSWID) [RFC-AAAA]

Table 21: CoSWID CBOR Tag

#### 6.6. URI Scheme Registrations

The ISO 19770-2:2015 SWID specification describes 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 in to ensure that a permanent registration exists for these schemes 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 is requested to register the URI scheme "swid". This registration request complies with [RFC7595].

Scheme name:  
swid

Status:  
Permanent

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

Contact:  
IETF Chair <chair@ietf.org>

Change controller:  
IESG <iesg@ietf.org>

Reference:  
Section 5.1 in RFC-AAAA

#### 6.6.2. URI-scheme swidpath

IANA is requested to register the URI scheme "swidpath". This registration request complies with [RFC7595].

Scheme name:  
swidpath

Status:  
Permanent

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

Contact:  
IETF Chair <chair@ietf.org>

Change controller:  
IESG <iesg@ietf.org>

Reference:  
Section 5.2 in RFC-AAAA

### 6.7. CoSWID Model for use in SWIMA Registration

The Software Inventory Message and Attributes (SWIMA) for PA-TNC specification [RFC8412] defines a standardized method for collecting an endpoint device's software inventory. A CoSWID can provide evidence of software installation which 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] [IANA.pa-tnc-parameters] to support CoSWID use in SWIMA as follows:

Pen: 0

Integer: TBD2

Name: Concise Software Identifier (CoSWID)

Reference: RFC-AAAA

Deriving Software Identifiers:

A Software Identifier generated from a CoSWID tag is expressed as a concatenation of the form 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 reference integrity measurements for files. The ISO-19770-2:2015 XML schema uses XML DSIG to support cryptographic signatures.

Signing CoSWID tags follows the procedures defined in CBOR Object Signing and Encryption [I-D.ietf-cose-rfc8152bis-struct]. 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>
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>
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-sign-coswid-header = {
    * cose-label => cose-values,
}

COSE_Signature = [
    protected: bstr .cbor protected-signature-coswid-header,
    unprotected: unprotected-sign-coswid-header,
    signature : bstr
]
<CODE ENDS>

```

Additionally, the COSE Header counter signature MAY be used as an attribute in the unprotected header map of the COSE envelope of a CoSWID [I-D.ietf-cose-countersign]. The application of counter signing enables second parties to provide a signature on a signature allowing for a 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 the security considerations (in Section 9) 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. Consecutively, the CBOR tag for CoSWID tags defined in Table 21 SHOULD be used in conjunction with CBOR data items that are a 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. In case 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. In case an unsigned CoSWID is tagged, a CoSWID CBOR tag MUST be appended before the CBOR data item that is the CoSWID tag.

```
<CODE BEGINS>
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 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 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 is 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 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 the 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 which 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 an association between the signature and the tag's entity item associated corresponding to the software provider. No mechanism is defined in this draft 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 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 create doubt about 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 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) does not need to be protected against unintended disclosure on an endpoint. Converse, generators of CoSWID tags need to ensure that only public information is disclosed. Entitlement Keys are an example for information where 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 appropriately controlled 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 mis-identification 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 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 file system 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 vendor and non-vendor created tags. 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 of [RFC8949], [I-D.ietf-cose-rfc8152bis-struct], and [I-D.ietf-cose-countersign] apply.

## 10. Privacy Consideration

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 to identify vulnerable software for attack. Collections of endpoint software information also can have privacy implications for users. The set of application a user installs can give clues to 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 the privacy risk (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 gives visibility into a user's application collection. For this reason, repositories of collected CoSWID tags not only 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. Change Log

This section is to be removed before publishing as an RFC.

[THIS SECTION TO BE REMOVED BY THE RFC EDITOR.]

Changes from version 12 to version 14:

- \* Moved key identifier to protected COSE header
- \* Fixed index reference for hash
- \* Removed indirection of CDDL type definition for filesystem-item
- \* Fixed quantity of resource and process
- \* Updated resource-collection
- \* Renamed socket name in software-meta to be consistent in naming
- \* Aligned excerpt examples in I-D text with full CDDL
- \* Fixed titles where title was referring to group instead of map
- \* Added missing date in SEMVER
- \* Fixed root cardinality for file and directory, etc.
- \* Transformed path-elements-entry from map to group for re-usability
- \* Scrubbed IANA Section
- \* Removed redundant supplemental rule
- \* Aligned discrepancy with ISO spec.

- \* Addressed comments on typos.
- \* Fixed kramdown nits and BCP reference.
- \* Addressed comments from WGLC reviewers.

Changes in version 12:

- \* Addressed a bunch of minor editorial issues based on WGLC feedback.
- \* Added text about the use of UTF-8 in CoSWID.
- \* Adjusted tag-id to allow for a UUID to be provided as a bstr.
- \* Cleaned up descriptions of index ranges throughout the document, removing discussion of 8 bit, 16 bit, etc.
- \* Adjusted discussion of private use ranges to use negative integer values and to be more clear throughout the document.
- \* Added discussion around resolving overlapping value spaces for version schemes.
- \* Added a set of expert review criteria for new IANA registries created by this document.
- \* Added new registrations for the "swid" and "swidpath" URI schemes, and for using CoSWID with SWIMA.

Changes from version 03 to version 11:

- \* Reduced representation complexity of the media-entry type and removed the Section describing the older data structure.
- \* Added more signature schemes from COSE
- \* Included a minimal required set of normative language
- \* Reordering of attribute name to integer label by priority according to semantics.
- \* Added an IANA registry for CoSWID items supporting future extension.
- \* Cleaned up IANA registrations, fixing some inconsistencies in the table labels.

- \* Added additional CDDL sockets for resource collection entries providing for additional extension points to address future SWID/CoSWID extensions.
- \* Updated Section on extension points to address new CDDL sockets and to reference the new IANA registry for items.
- \* Removed unused references and added new references to address placeholder comments.
- \* Added table with semantics for the link ownership item.
- \* Clarified language, made term use more consistent, fixed references, and replacing lowercase RFC2119 keywords.

Changes from version 02 to version 03:

- \* Updated core CDDL including the CDDL design pattern according to RFC 8428.

Changes from version 01 to version 02:

- \* Enforced a more strict separation between the core CoSWID definition and additional usage by moving content to corresponding appendices.
- \* Removed artifacts inherited from the reference schema provided by ISO (e.g., NMTOKEN(S))
- \* Simplified the core data definition by removing group and type choices where possible
- \* Minor reordering of map members
- \* Added a first extension point to address requested flexibility for extensions beyond the any-element

Changes from version 00 to version 01:

- \* Ambiguity between evidence and payload eliminated by introducing explicit members (while still
- \* allowing for "empty" SWID tags)
- \* Added a relatively restrictive COSE envelope using cose\_sign1 to define signed CoSWID (single signer only, at the moment)

- \* Added a definition how to encode hashes that can be stored in the any-member using existing IANA tables to reference hash-algorithms

Changes since adopted as a WG I-D -00:

- \* Removed redundant any-attributes originating from the ISO-19770-2:2015 XML schema definition
- \* Fixed broken multi-map members
- \* Introduced a more restrictive item (any-element-map) to represent custom maps, increased restriction on types for the any-attribute, accordingly
- \* Fixed X.1520 reference
- \* Minor type changes of some attributes (e.g., NMTOKENS)
- \* Added semantic differentiation of various name types (e.g. fs-name)

Changes from version 06 to version 07:

- \* Added type choices/enumerations based on textual definitions in 19770-2:2015
- \* Added value registry request
- \* Added media type registration request
- \* Added content format registration request
- \* Added CBOR tag registration request
- \* Removed RIM appendix to be addressed in complementary draft
- \* Removed CWT appendix
- \* Flagged firmware resource collection appendix for revision
- \* Made use of terminology more consistent
- \* Better defined use of extension points in the CDDL
- \* Added definitions for indexed values
- \* Added IANA registry for Link use indexed values

Changes from version 05 to version 06:

- \* Improved quantities
- \* Included proposals for implicit enumerations that were NMTOKENS
- \* Added extension points
- \* Improved exemplary firmware-resource extension

Changes from version 04 to version 05:

- \* Clarified language around SWID and CoSWID to make more consistent use of these terms.
- \* Added language describing CBOR optimizations for single vs. arrays in the model front matter.
- \* Fixed a number of grammatical, spelling, and wording issues.
- \* Documented extension points that use CDDL sockets.
- \* Converted IANA registration tables to markdown tables, reserving the 0 value for use when a value is not known.
- \* Updated a number of references to their current versions.

Changes from version 03 to version 04:

- \* Re-index label values in the CDDL.
- \* Added a Section describing the CoSWID model in detail.
- \* Created IANA registries for entity-role and version-scheme

Changes from version 02 to version 03:

- \* Updated CDDL to allow for a choice between a payload or evidence
- \* Re-index label values in the CDDL.
- \* Added item definitions
- \* Updated references for COSE, CBOR Web Token, and CDDL.

Changes from version 01 to version 02:

- \* Added extensions for Firmware and CoSWID use as Reference Integrity Measurements (CoSWID RIM)
- \* Changes meta handling in CDDL from use of an explicit use of items to a more flexible unconstrained collection of items.
- \* Added Sections discussing use of COSE Signatures and CBOR Web Tokens

Changes from version 00 to version 01:

- \* Added CWT usage for absolute SWID paths on a device
- \* Fixed cardinality of type-choices including arrays
- \* Included first iteration of firmware resource-collection

## 12. References

### 12.1. 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, DOI 10.17487/RFC6648, June 2012, <<https://www.rfc-editor.org/info/rfc6648>>.
- [BCP26] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [I-D.ietf-cose-countersign] Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress, Internet-Draft, draft-ietf-cose-countersign-05, 23 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-countersign-05.txt>>.
- [I-D.ietf-cose-rfc8152bis-struct] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

- [IANA.cbor-tags]  
IANA, "Concise Binary Object Representation (CBOR) Tags",  
<<http://www.iana.org/assignments/cbor-tags>>.
- [IANA.core-parameters]  
IANA, "Constrained RESTful Environments (CoRE)  
Parameters",  
<<http://www.iana.org/assignments/core-parameters>>.
- [IANA.media-types]  
IANA, "Media Types",  
<<http://www.iana.org/assignments/media-types>>.
- [IANA.named-information]  
IANA, "Named Information",  
<<http://www.iana.org/assignments/named-information>>.
- [IANA.pa-tnc-parameters]  
IANA, "Posture Attribute (PA) Protocol Compatible with  
Trusted Network Connect (TNC) Parameters",  
<<http://www.iana.org/assignments/pa-tnc-parameters>>.
- [IANA.uri-schemes]  
IANA, "Uniform Resource Identifier (URI) Schemes",  
<<http://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>>.
- [SAM] "Information technology - Software asset management - Part 5: Overview and vocabulary", ISO/IEC 19770-5:2015, 15 November 2013.
- [SEMVER] Preston-Werner, T., "Semantic Versioning 2.0.0", <<https://semver.org/spec/v2.0.0.html>>.

- [SWID] "Information technology - Software asset management - Part 2: Software identification tag", ISO/IEC 19770-2:2015, 1 October 2015.
- [UNSPSC] "United Nations Standard Products and Services Code", 26 October 2020, <<https://www.unspsc.org/>>.
- [W3C.REC-css3-mediaqueries-20120619]  
Rivoal, F., "Media Queries", World Wide Web Consortium Recommendation REC-css3-mediaqueries-20120619, 19 June 2012, <<https://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619>>.
- [W3C.REC-xmlschema-2-20041028]  
Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, 28 October 2004, <<https://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.
- [W3C.REC-xpath20-20101214]  
Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., and J. Simeon, "XML Path Language (XPath) 2.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xpath20-20101214, 14 December 2010, <<https://www.w3.org/TR/2010/REC-xpath20-20101214>>.

## 12.2. Informative References

- [CamelCase]  
"UpperCamelCase", 29 August 2014, <<http://wiki.c2.com/?CamelCase>>.
- [I-D.ietf-rats-architecture]  
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-15, 8 February 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-15.txt>>.
- [KebabCase]  
"KebabCase", 18 December 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>>.
- [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] "Recommendation ITU-T X.1520 (2014), Common vulnerabilities and exposures", 20 April 2011.

#### 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](mailto: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](mailto:henk.birkholz@sit.fraunhofer.de)

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

Charles Schmidt  
The MITRE Corporation  
202 Burlington Road  
Bedford, Massachusetts 01730  
United States of America  
Email: [cmschmidt@mitre.org](mailto: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](mailto:david.waltermire@nist.gov)

SACM Working Group  
Internet-Draft  
Intended status: Informational  
Expires: June 17, 2019

H. Birkholz  
Fraunhofer SIT  
J. Lu  
Oracle Corporation  
J. Strassner  
Huawei Technologies  
N. Cam-Winget  
Cisco Systems  
A. Montville  
CIS  
December 14, 2018

Security Automation and Continuous Monitoring (SACM) Terminology  
draft-ietf-sacm-terminology-16

Abstract

This memo documents terminology used in the documents produced by SACM (Security Automation and Continuous Monitoring).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 17, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terms and Definitions . . . . .	2
3. IANA Considerations . . . . .	21
4. Security Considerations . . . . .	21
5. Acknowledgements . . . . .	22
6. Change Log . . . . .	22
7. Contributors . . . . .	26
8. References . . . . .	27
8.1. Normative References . . . . .	28
8.2. Informative References . . . . .	28
Appendix A. The Attic . . . . .	29
Authors' Addresses . . . . .	29

## 1. Introduction

Our goal with this document is to improve our agreement on the terminology used in documents produced by the IETF Working Group for Security Automation and Continuous Monitoring. Agreeing on terminology should help reach consensus on which problems we're trying to solve, and propose solutions and decide which ones to use.

## 2. Terms and Definitions

This section describes terms that have been defined by other RFC's and defines new ones. The predefined terms will reference the RFC and where appropriate will be annotated with the specific context by which the term is used in SACM. Note that explanatory or informational augmentation to definitions are segregated from the definitions themselves. The definition for the term immediately follows the term on the same line, whereas expository text is contained in subsequent paragraphs immediately following the definition.

**Assertion:** Defined by the ITU in [X.1252] as "a statement made by an entity without accompanying evidence of its validity".

In the context of SACM, an assertion is the output of a SACM Component in the form of a SACM Statement (including metadata about the data source and data origin, e.g. timestamps). While the validity of an assertion about Content and Content Metadata cannot be verified without, for example, Integrity Proofing of the

Data Source, an assertion (and therefore a SACM statement, respectively) of the validity of Statement Metadata can be enabled by including corresponding Integrity Evidence created by the Data Origin.

**Assessment:** Defined in [RFC5209] as "the process of collecting posture for a set of capabilities on the endpoint (e.g., host-based firewall) such that the appropriate validators may evaluate the posture against compliance policy."

**Attribute:** Is a data element, as defined in [RFC5209], that is atomic.

In the context of SACM, attributes are "atomic" information elements and an equivalent to attribute-value-pairs. Attributes can be components of Subjects, the basic composite definitions that are defined in the SACM Information Model.

**Capability:** A set of features that are available from a SACM Component.

See also "capability" in [I-D.ietf-i2nsf-terminology].

In the context of SACM, the extent of a SACM component's ability is enabled by the functions it is composed of. Capabilities are registered at a SACM broker (potentially also at a proxy or a repository component if it includes broker functions) by a SACM component via the SACM component registration task and can be discovered by or negotiated with other SACM components via the corresponding tasks. For example, the capability of a SACM provider may be to provide target endpoint records (declarative guidance about well-known or potential target endpoints), or only a subset of that data.

A capability's description is in itself imperative guidance on what functions are exposed to other SACM components in a SACM domain and how to use them in workflows.

The SACM Vulnerability Assessment Scenario [I-D.ietf-sacm-vuln-scenario] defines the terms Endpoint Management Capabilities, Vulnerability Management Capabilities, and Vulnerability Assessment Capabilities, which illustrate specific sets of SACM capabilities on an enterprise IT department's point of view and therefore compose sets of declarative guidance.

**Collection Result:** Is a composition of one or more content elements carrying information about a target endpoint, that is produced by a collector when conducting a collection task.

**Collection Task:** A targeted task that collects attributes and/or corresponding attribute values from target endpoint.

There are four types of frequency collection tasks can be conducted with:

ad-hoc, e.g. triggered by a unsolicited query

conditional, e.g. triggered in accordance with policies included in the compositions of workflows

scheduled, e.g. in regular intervals, such as every minute or weekly

continuously, e.g. a network behavior observation

There are three types of collection methods, each requiring an appropriate set of functions to be included in the SACM component conducting the collection task:

**Self-Reporting:** A SACM component located on the target endpoint itself conducts the collection task.

**Remote-Acquisition:** A SACM component located on an Endpoint different from the target endpoint conducts the collection task via interfaces available on the target endpoint, e.g. SNMP/NETCONF or WMI.

**Behavior-Observation:** A SACM component located on an Endpoint different from the target endpoint observes network traffic related to the target endpoint and conducts the collection task via interpretation of that network traffic.

**Collector:** A piece of software that acquires information about one or more target endpoints by conducting collection tasks.

A collector can be distributed across multiple endpoints, e.g. across a target endpoint and a SACM component. The separate parts of the collector can communicate with a specialized protocol, such as PA-TNC [RFC5792]. At least one part of a distributed collector has to take on the role of a provider of information by providing SACM interfaces to propagate capabilities and to provide SACM content in the form of collection results.



**Configuration:** A non-volatile subset of the endpoint attributes of a endpoint that is intended to be unaffected by a normal reboot-cycle.

Configuration is a type of imperative guidance that is stored in files (files dedicated to contain configuration and/ or files that are software components), directly on block devices, or on specific hardware components that can be accessed via corresponding software components. Modification of configuration can be conducted manually or automatically via management (plane) interfaces that support management protocols, such as SNMP or WMI. A change of configuration can occur during both run-time and down-time of an endpoint. It is common practice to schedule a change of configuration during or directly after the completion of a boot-cycle via corresponding software components located on the target endpoint itself.

**Examples:** The static association of an IP address and a MAC address in a DHCP server configuration, a directory-path that identifies a log-file directory, a registry entry.

**Configuration Drift:** The disposition of endpoint characteristics to change over time.

Configuration drift exists for both hardware components and software components. Typically, the frequency and scale of configuration drift of software components is significantly higher than the configuration drift of hardware components.

**Consumer:** A SACM Role that requires a SACM Component to include SACM Functions enabling it to receive information from other SACM Components.

**Content Element:** Content elements constitute the payload data (SACM content) transferred via statement Subjects emitted by providers of information. Every content element Subject includes a specific content Subject and a corresponding content metadata Subject.

**Content Metadata:** Data about content Subjects. Every content-element includes a content metadata Subject. The Subject can include any information element that can annotate the content transferred. Examples include time stamps or data provenance Subjects.

**Control Plane:** An architectural component that provides common control functions to all SACM components.

Typically used as a term in the context of routing, e.g. [RFC6192]. SACM components may include authentication, authorization, (capability) discovery or negotiation, registration and subscription. The control plane orchestrates the flow on the data plane according to imperative guidance (i.e. configuration) received via the management plane. SACM components with interfaces to the control plane have knowledge of the capabilities of other SACM components within a SACM domain.

**Controller:** A controller is a SACM Role that is assigned to a SACM component containing control plane functions managing and facilitating information sharing or execute on security functions.

There are three types of SACM controllers: Broker, Proxy, and Repository. Depending on its type, a controller can also contain functions that have interfaces on the data plane.

**Data Confidentiality:** Defined in [RFC4949] as "the property that data is not disclosed to system entities unless they have been authorized to know the data."

**Data In Motion:** Data that is being transported via a network; also referred to as "Data in Transit" or "Data in Flight".

Data in motion requires a data model to transfer the data using a specific encoding. Typically, data in motion is serialized (marshalling) into a transport encoding by a provider of information and deserialized (unmarshalling) by a consumer of information. The termination points of provider of information and consumer of information data is transferred between are interfaces. In regard to data in motion, the interpretation of the roles consumer of information and provider of information depends on the corresponding OSI layer (e.g. on layer2: between interfaces connected to a broadcast domain, on layer4: between interfaces that maintain a TCP connection). In the context of SACM, consumer of information and provider of information are SACM components.

**Data At Rest:** Data that is stored.

Data at rest requires a data model to encode the data to be stored. In the context of SACM, data at rest located on a SACM component can be provided to other SACM components via discoverable capabilities.

**Data Integrity:** Defined in [RFC4949] as "the property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner."

**Data Origin:** The SACM Component that initially acquired or produced data about an endpoint.

Data Origin enables a SACM component to identify the SACM component that initially acquired or produced data about a (target) endpoint (e.g. via collection from a data source) and made it available to a SACM domain via a SACM statement. Data Origin can be expressed by an endpoint label information element (e.g. to be used as metadata in statement).

**Data Plane:** Is an architectural component providing operational functions enabling information exchange that is not command and control or management related.

Typically used as a term in the context of routing (and used as a synonym for forwarding plane, e.g. [RFC6192]). In the context of SACM, the data plane is an architectural component providing operational functions to enable a SACM component to provide and consume SACM statements and therefore SACM content, which composes the actual SACM content. The data plane in a SACM domain is used to conduct distributed SACM tasks by transporting SACM content via specific transport encodings and corresponding operations defined by SACM data models.

**Data Provenance:** An historical record of the sources, origins and evolution, as it pertains to data, that is influenced by inputs, entities, functions and processes.

Additional Information - In the context of SACM, data provenance is expressed as metadata that identifies SACM statements and corresponding content elements a new statement is created from. In a downstream process, this references can cascade, creating a data provenance tree that enables SACM components to trace back the original data sources involved in the creation of SACM statements and take into account their characteristics and trustworthiness.

**Data Source:** Is an endpoint from which a particular set of attributes and/or attribute values have been collected.

Data Source enables a SACM component to identify - and potentially characterize - a (target) endpoint that is claimed to be the original source of endpoint attributes in a SACM statement. Data Source can be expressed as metadata by an endpoint label information element or a corresponding subject of identifying endpoint attributes.

**Endpoint:** Defined in [RFC5209] as "any computing device that can be connected to a network."

Additional Information - The [RFC5209] definition continues, "Such devices normally are associated with a particular link layer address before joining the network and potentially an IP address once on the network. This includes: laptops, desktops, servers, cell phones, or any device that may have an IP address."

To further clarify the [RFC5209] definition, an endpoint is any physical or virtual device that may have a network address. Note that, network infrastructure devices (e.g. switches, routers, firewalls), which fit the definition, are also considered to be endpoints within this document.

Physical endpoints are always composites that are composed of hardware components and software components. Virtual endpoints are composed entirely of software components and rely on software components that provide functions equivalent to hardware components.

The SACM architecture differentiates two essential categories of endpoints: Endpoints whose security posture is intended to be assessed (target endpoints) and endpoints that are specifically excluded from endpoint posture assessment (excluded endpoints).

Based on the definition of an asset, an endpoint is a type of asset.

**Endpoint Attribute:** Is a discreet endpoint characteristic that is computably observable.

Endpoint Attributes typically constitute Attributes that can be bundled into Subject (e.g. information about a specific network interface can be represented via a set of multiple AVP).

**Endpoint Characteristics:** The state, configuration and composition of the software components and (virtual) hardware components a target endpoint is composed of, including observable behavior, e.g. sys-calls, log-files, or PDU emission on a network.

In SACM work-flows, (Target) Endpoint Characteristics are represented via Information Elements.

**Endpoint Characterization Task:** The task of endpoint characterization that uses endpoint attributes that represent distinct endpoint characteristics.

**Endpoint Classification:** The categorization of of the endpoint into one or more taxonomic structures.

Endpoint classification requires declarative guidance in the form of an endpoint profile, discovery results and potentially collection results. Types, classes or the characteristics of an individual target endpoint are defined via endpoint profiles.

**Endpoint Classification Task:** The task of endpoint classification that uses an endpoint's characteristics to determine how to categorize the given endpoint into one or more taxonomic structures.

**Endpoint Label:** A unique label associated with a unique endpoint.

Endpoint specializations have corresponding endpoint label specializations. For example, an endpoint label used on a SACM Component is a SACM Component Label.

**Endpoint Management Capabilities:** Enterprise IT management capabilities that are tailored to manage endpoint identity, endpoint information, and associated metadata.

**Evaluation Task:** A task by which an endpoint's asserted attribute value is evaluated against a policy-compliant attribute value.

**Evaluation Result:** The resulting value from having evaluated a set of posture attributes.

**Expected Endpoint Attribute State:** The policy-compliant state of an endpoint attribute that is to be compared against.

Sets of expected endpoint attribute states are transported as declarative guidance in target endpoint profiles via the management plane. This, for example, can be a policy, but also a recorded past state. An expected state is represented by an Attribute or a Subject that represents a set of multiple attribute value pairs.

**Guidance:** Machine-processable input directing SACM processes or tasks.

Examples of such processes/tasks include automated device management, remediation, collection, evaluation. Guidance influences the behavior of a SACM Component and is considered content of the management plane. In the context of SACM, guidance is machine-readable and can be manually or automatically generated

or provided. Typically, the tasks that provide guidance to SACM components have a low-frequency and tend to be sporadic.

There are two types of guidance:

**Declarative Guidance:** Guidance that defines the configuration or state an endpoint is supposed to be in, without providing specific actions or methods to produce that desired state. Examples include Target Endpoint Profiles or network topology based requirements.

**Imperative Guidance:** Guidance that prescribes specific actions to be conducted or methods to be used in order to achieve an outcome. Examples include a targeted Collection Task or the IP-Address of a SACM Component that provides a registration function.

Prominent examples include: modification of the configuration of a SACM component or updating a target endpoint profile that resides on an evaluator. In essence, guidance is transported via the management plane.

**Endpoint Hardware Inventory:** The set of hardware components that compose a specific endpoint representing its hardware configuration.

**Hardware Component:** A distinguishable physical component used to compose an endpoint.

The composition of an endpoint can be changed over time by adding or removing hardware components. In essence, every physical endpoint is potentially a composite of multiple hardware components, typically resulting in a hierarchical composition of hardware components. The composition of hardware components is based on interconnects provided by specific hardware types (e.g. FRU in a chassis are connected via redundant busses). In general, a hardware component can be distinguished by its serial number. Occasionally, hardware components are referred to as power sucking aliens.

**Information Element:** A representation of information about physical and virtual "objects of interest".

Information elements are the building blocks that constitute the SACM information model. In the context of SACM, an information element that expresses a single value with a specific name is referred to as an Attribute (analogous to an attribute-value-pair). A set of attributes that is bundled into a more complex composite information element is referred to as a Subject. Every

information element in the SACM information model has a unique name. Endpoint attributes or time stamps, for example, are represented as information elements in the SACM information model.

**Information Model:** An abstract representation of data, their properties, relationships between data and the operations that can be performed on the data.

While there is some overlap with a data model, [RFC3444] distinguishes an information model as being protocol and implementation neutral whereas a data model would provide such details. The purpose of the SACM information model is to ensure interoperability between SACM data models (that are used as transport encoding) and to provide a standardized set of information elements for communication between SACM components.

**Interaction Model:** The definition of specific sequences regarding the exchange of messages (data in motion), including, for example, conditional branching, thresholds and timers.

An interaction model, for example, can be used to define operations, such as registration or discovery, on the control plane. A composition of data models for data in motion and a corresponding interaction model is a protocol.

**Internal Collector:** A collector that runs on a target endpoint to acquire information from that target endpoint.

**Management Plane:** An architectural component providing common functions to steer the behavior of SACM components, e.g. their behavior on the control plane.

Typically, a SACM component can fulfill its purpose without continuous input from the management plane. In contrast, without continuous availability of control plane functions a typical SACM component could not function properly. In general, interaction on the management plane is less frequent and less regular than on the control plane. Input via the management plane can be manual (e.g. via a CLI), or can be automated via management plane functions that are part of other SACM components.

**Network Address:** A layer-specific address that follows a layer-specific address scheme.

The following characteristics are a summary derived from the Common Information Model and ITU-T X.213. Each Network Interface of a specific layer can be associated with one or more addresses appropriate for that layer. There is no guarantee that a network

address is globally unique. A dedicated authority entity can provide a level of assurance that a network address is unique in its given scope. In essence, there is always a scope to a network address, in which it is intended to be unique.

Examples include: physical Ethernet port with a MAC address, layer 2 VLAN interface with a MAC address, layer 3 interface with multiple IPv6 addresses, layer 3 tunnel ingress or egress with an IPv4 address.

**Network Interface:** An Endpoint is connected to a network via one or more Network Interfaces. Network Interfaces can be physical (Hardware Component) or logical (virtual Hardware component, i.e. a dedicated Software Component). Network Interfaces of an Endpoint can operate on different layers, most prominently what is now commonly called layer 2 and 3. Within a layer, interfaces can be nested.

In SACM, the association of Endpoints and Network Addresses via Network Interfaces is vital to maintain interdependent autonomous processes that can be targeted at Target Endpoints, unambiguously.

Examples include: physical Ethernet port, layer 2 VLAN interface, a MC-LAG setup, layer 3 Point-to-Point tunnel ingress or egress.

**Metadata:** Data about data.

In the SACM information model, data is referred to as Content. Metadata about the content is referred to as Content-Metadata, respectively. Content and Content-Metadata are combined into Subjects called Content-Elements in the SACM information model. Some information elements defined by the SACM information model can be part of the Content or the Content-Metadata. Therefore, if an information element is considered data or data about data depends on which kind of Subject it is associated with. The SACM information model also defines metadata about the data origin via the Subject Statement-Metadata. Typical examples of metadata are time stamps, data origin or data source.

**Posture:** Defined in [RFC5209] as "configuration and/or status of hardware or software on an endpoint as it pertains to an organization's security policy."

This term is used within the scope of SACM to represent the configuration and state information that is collected from a target endpoint in the form of endpoint attributes (e.g. software/hardware inventory, configuration settings, dynamically assigned



addresses). This information may constitute one or more posture attributes.

**Posture Attributes:** Defined in [RFC5209] as "attributes describing the configuration or status (posture) of a feature of the endpoint. A Posture Attribute represents a single property of an observed state. For example, a Posture Attribute might describe the version of the operating system installed on the system."

Within this document this term represents a specific assertion about endpoint configuration or state (e.g. configuration setting, installed software, hardware) represented via endpoint attributes. The phrase "features of the endpoint" highlighted above refers to installed software or software components.

**Provider:** A provider is a SACM role assigned to a SACM component that provides role-specific functions to provide information to other SACM components.

**Repository:** A repository is a controller that contains functions to consume, store and provide information of a particular kind.

Such information is typically data transported on the data plane, but potentially also data and metadata from the control and management plane. A single repository may provide the functions of more than one specific repository type (i.e. configuration baseline repository, assessment results repository, etc.)

**SACM Broker Controller:** A SACM Broker Controller is a controller that contains control plane functions to provide and/or connect services on behalf of other SACM components via interfaces on the control plane.

A broker may provide, for example, authorization services and find, upon request, SACM components providing requested services.

**SACM Component:** Is a component, as defined in [I-D.ietf-i2nsf-terminology], that is composed of SACM capabilities.

In the context of SACM, a set of SACM functions composes a SACM component. A SACM component conducts SACM tasks, acting on control plane, data plane and/or management plane via corresponding SACM interfaces. SACM defines a set of standard components (e.g. a collector, a broker, or a data store). A SACM component contains at least a basic set of control plane functions and can contain data plane and management plane functions. A SACM component residing on an endpoint assigns one or more SACM roles

to the corresponding endpoint due to the SACM functions it is composed of. A SACM component "resides on" an endpoint and an endpoint "contains" a SACM component, correspondingly. For example, a SACM component that is composed solely of functions that provide information would only take on the role of a provider.

**SACM Component Discovery:** The task of discovering the capabilities provided by SACM components within a SACM domain.

This is likely to be performed via an appropriate set of control plane functions.

**SACM Component Label:** A specific endpoint label that is used to identify a SACM component.

In content-metadata, this label is called data origin.

**SACM Content:** The payload provided by SACM components to the SACM domain on the data plane.

SACM content includes the SACM data models.

**SACM Domain:** Endpoints that include a SACM component compose a SACM domain.

(To be revised, additional definition content TBD, possible dependencies to SACM architecture)

**SACM Function:** A behavioral aspect of a SACM component that provides external SACM Interfaces or internal interfaces to other SACM Functionse.

For example, a SACM Function with SACM Interfaces on the Control Plane can provide a brokering function to other SACM Components. Via Data Plane interfaces, a SACM Function can act as a provider and/or as a consumer of information. SACM Functions can be propagated as the Capabilities of a SACM Component and can be discovered by or negotiated with other SACM Components.

**SACM Interface:** An interface, as defined in [I-D.ietf-i2nsf-terminology], that provides SACM-specific operations.

[I-D.ietf-i2nsf-terminology] defines interface as a "set of operations one object knows it can invoke on, and expose to, another object," and further defines interface by stating that an interface "decouples the implementation of the operation from its

specification. An interface is a subset of all operations that a given object implements. The same object may have multiple types of interfaces to serve different purposes."

In the context of SACM, SACM Functions provide SACM Interfaces on the management, control, or data plane. Operations a SACM Interface provides are based on corresponding data model defined by SACM. SACM Interfaces are used for communication between SACM components.

**SACM Proxy Controller:** A SACM Proxy Controller is a controller that provides data plane and control plane functions, information, or services on behalf of another component, which is not directly participating in the SACM architecture.

**SACM Role:** Is a role, as defined in [I-D.ietf-i2nsf-terminology], that requires the SACM Component assuming the role to bear a set of SACM functions or interfaces.

SACM Roles provide three important benefits. First, it enables different behavior to be supported by the same Component for different contexts. Second, it enables the behavior of a Component to be adjusted dynamically (i.e., at runtime, in response) to changes in context, by using one or more Roles to define the behavior desired for each context. Third, it decouples the Roles of a Component from the Applications that use that Component."

In the context of SACM, SACM roles are associated with SACM components and are defined by the set of functions and interfaces a SACM component includes. There are three SACM roles: provider, consumer, and controller. The roles associated with a SACM component are determined by the purpose of the SACM functions and corresponding SACM interfaces the SACM component is composed of.

**SACM Statement:** Is an assertion that is made by a SACM Component.

**Security Automation:** The process of which security alerts can be automated through the use of different components to monitor, analyze and assess endpoints and network traffic for the purposes of detecting misconfigurations, misbehaviors or threats.

Security Automation is intended to identify target endpoints that cannot be trusted (see "trusted" in [RFC4949]). This goal is achieved by creating and processing evidence (assessment statements) that a target endpoint is not a trusted system [RFC4949].

**Software Package:** A generic software package (e.g. a text editor).

**Software Component:** A software package installed on an endpoint.

The software component may include a unique serial number (e.g. a text editor associated with a unique license key).

**Software Instance:** A running instance of a software component.

For example, on a multi-user system, one logged-in user has one instance of a text editor running and another logged-in user has another instance of the same text editor running, or on a single-user system, a user could have multiple independent instances of the same text editor running.

**State:** A volatile set of endpoint attributes of a (target) endpoint that is affected by a reboot-cycle.

Local state is created by the interaction of components with other components via the control plane, via processing data plane payload, or via the functional properties of local hardware and software components. Dynamic configuration (e.g. IP address distributed dynamically via an address distribution and management services, such as DHCP) is considered state that is the result of the interaction with another component (e.g. provided by a DHCP server with a specific configuration).

**Examples:** The static association of an IP address and a MAC address in a DHCP server configuration, a directory-path that identifies a log-file directory, a registry entry.

**Statement:** A statement is the root/top-level subject defined in the SACM information model.

A statement is used to bundle Content Elements into one subject and includes metadata about the data origin.

**Subject:** A semantic composite information element pertaining to a system entity that is a target endpoint.

Like Attributes, subjects have a name and are composed of attributes and/or other subjects. Every IE that is part of a subject can have a quantity associated with it (e.g. zero-one, none-unbounded). The content IE of a subject can be an unordered or an ordered list.

In contrast to the definitions of subject provided by [RFC4949], a subject in the scope of SACM is neither "a system entity that

causes information to flow among objects or changes the system state" nor "a name of a system entity that is bound to the data items in a digital certificate".

In the context of SACM, a subject is a semantic composite of information elements about a system entity that is a target endpoint. Every acquirable subject-as defined in the scope of SACM-about a target endpoint represents and therefore identifies every subject-as defined by [RFC4949]-that is a component of that target endpoint. The semantic difference between both definitions can be subtle in practice and is in consequence important to highlight.

**Supplicant:** A component seeking to be authenticated via the control plane for the purpose of participating in a SACM domain.

**System Resource:** Defined in [RFC4949] as "data contained in an information system; or a service provided by a system; or a system capacity, such as processing power or communication bandwidth; or an item of system equipment (i.e., hardware, firmware, software, or documentation); or a facility that houses system operations and equipment."

**Target Endpoint:** Is an endpoint that is under assessment at some point in, or region of, time.

Every endpoint that is not specifically designated as an excluded endpoint is a target endpoint. A target endpoint is not part of a SACM domain unless it contains a SACM component (e.g. a SACM component that publishes collection results coming from an internal collector).

A target endpoint is similar to a device that is a Target of Evaluation (TOE) as defined in Common Criteria and as referenced by {{RFC4949}}.

**Target Endpoint Address:** An address that is layer specific and which follows layer specific address schemes.

Each interface of a specific layer can be associated with one or more addresses appropriate for that layer. There is no guarantee that an address is globally unique. In general, there is a scope to an address in which it is intended to be unique.

Examples include: physical Ethernet port with a MAC address, layer 2 VLAN interface with a MAC address, layer 3 interface with multiple IPv6 addresses, layer 3 tunnel ingress or egress with an IPv4 address.

**Target Endpoint Characterization:** The description of the distinctive nature of a target endpoint, that is based on its characteristics.

**Target Endpoint Characterization Record:** A set of endpoint attributes about a target endpoint that was encountered in a SACM domain, which are associated with that target endpoint as a result of a Target Endpoint Characterization Task.

A characterization record is intended to be a representation of an endpoint. It cannot be assured that a record distinctly represents a single target endpoint unless a set of one or more endpoint attributes that compose a unique set of identifying endpoint attributes are included in the record. Otherwise, the set of identifying attributes included in a record can match more than one target endpoints, which are - in consequence - indistinguishable to a SACM domain until more qualifying endpoint attributes can be acquired and added to the record. A characterization record is maintained over time in order to assert that acquired endpoint attributes are either about an endpoint that was encountered before or an endpoint that has not been encountered before in a SACM domain. A characterization record can include, for example, acquired configuration, state or observed behavior of a specific target endpoint. Multiple and even conflicting instances of this information can be included in a characterization record by using timestamps and/or data origins to differentiate them. The endpoint attributes included in a characterization record can be used to re-identify a distinct target endpoint over time. Classes or profiles can be associated with a characterization record via the Classification Task in order to guide collection, evaluation or remediation tasks.

**Target Endpoint Characterization Task:** An ongoing task of continuously adding acquired endpoint attributes to a corresponding record. The TE characterization task manages the representation of encountered target endpoints in the SACM domain in the form of characterization records. For example, the output of a target endpoint discovery task or a collection task can be processed by the characterization task and added to the record. The TE characterization Task also manages these representations of target endpoints encountered in the SACM domain by splitting or merging the corresponding records as new or more refined endpoint attributes become available.

**Target Endpoint Classification Task:** The task of associating a class from an extensible list of classes with an endpoint characterization record. TE classes function as imperative and declarative guidance for collection, evaluation, remediation and security posture assessment in general.

**Target Endpoint Discovery Task:** The ongoing task of detecting previously unknown interaction of a potential target endpoint in the SACM domain. TE Discovery is not directly targeted at a specific target endpoint and therefore an un-targeted task. SACM Components conducting the discovery task as a part of their function are typically distributed and located, for example, on infrastructure components or collect from those remotely via appropriate interfaces. Examples of infrastructure components that are of interest to the discovery task include routers, switches, VM hosting or VM managing components, AAA servers, or servers handling dynamic address distribution.

**Target Endpoint Identifier:** The target endpoint discovery task and the collection tasks can result in a set of identifying endpoint attributes added to a corresponding Characterization Record. This subset of the endpoint attributes included in the record is used as a target endpoint identifier, by which a specific target endpoint can be referenced. Depending on the available identifying attributes, this reference can be ambiguous and is a "best-effort" mechanism. Every distinct set of identifying endpoint attributes can be associated with a target endpoint label that is unique in a SACM domain.

**Target Endpoint Label:** An endpoint label that identifies a specific target endpoint.

**Target Endpoint Profile:** A bundle of expected or desired component composition, configurations and states that is associated with a target endpoint.

The corresponding task by which the association with a target endpoint takes places is the endpoint classification task. The task by which an endpoint profile is created is the endpoint characterization task. A type or class of target endpoints can be defined via a target endpoint profile. Examples include: printers, smartphones, or an office PC.

In respect to [RFC4949], a target endpoint profile is a protection profile as defined by Common Criteria (analogous to the target endpoint being the target of evaluation).

**SACM Task:** Is a task conducted within the scope of a SACM domain by one or more SACM functions that achieves a SACM-defined outcome.

A SACM task can be triggered by other operations or functions (e.g. a query from another SACM component or an unsolicited push on the data plane due to an ongoing subscription). A task is part of a SACM process chain. A task starts at a given point in time

and ends in a deterministic state. With the exception of a collection task, a SACM task consumes SACM statements provided by other SACM components. The output of a task is a result that can be provided (e.g. published) on the data plane.

The following tasks are defined by SACM:

Target Endpoint Discovery

Target Endpoint Characterization

Target Endpoint Classification

Collection

Evaluation [TBD]

Information Sharing [TBD]

SACM Component Discovery

SACM Component Authentication [TBD]

SACM Component Authorization [TBD]

SACM Component Registration [TBD]

**Timestamps :** Defined in [RFC4949] as "with respect to a data object, a label or marking in which is recorded the time (time of day or other instant of elapsed time) at which the label or marking was affixed to the data object".

A timestamp always requires context, i.e. additional information elements that are associated with it. Therefore, all timestamps wrt information elements are always metadata. Timestamps in SACM Content Elements may be generated outside a SACM Domain and may be encoded in an unknown representation. Inside a SACM domain the representation of timestamps is well-defined and unambiguous.

**Virtual Endpoint:** An endpoint composed entirely of logical system components (see [RFC4949]).

The most common example is a virtual machine/host running on a target endpoint. Effectively, target endpoints can be nested and at the time of this writing the most common example of target endpoint characteristics about virtual components is the EntLogicalEntry in [RFC6933].



**Vulnerability Assessment:** An assessment specifically tailored to determining whether a set of endpoints is vulnerable according to the information contained in the vulnerability description information.

**Vulnerability Description Information:** Information pertaining to the existence of a flaw or flaws in software, hardware, and/or firmware, which could potentially have an adverse impact on enterprise IT functionality and/or security.

Vulnerability description information should contain enough information to support vulnerability detection.

**Vulnerability Detection Data:** A type of imperative guidance extracted or derived from vulnerability description information that describes the specific mechanisms of vulnerability detection that is used by an enterprise's vulnerability management capabilities to determine if a vulnerability is present on an endpoint.

**Vulnerability Management Capabilities:** An IT management capability tailored toward managing endpoint vulnerabilities and associated metadata on an ongoing basis by ingesting vulnerability description information and vulnerability detection data, and performing vulnerability assessments.

**Vulnerability assessment capabilities:** An assessment capability that is tailored toward determining whether a set of endpoints is vulnerable according to vulnerability description information.

**Workflow:** A workflow is a modular composition of tasks that can contain loops, conditionals, multiple starting points and multiple endpoints.

The most prominent workflow in SACM is the assessment workflow.

### 3. IANA Considerations

This memo includes no request to IANA.

### 4. Security Considerations

This memo documents terminology for security automation. While it is about security, it does not affect security.

## 5. Acknowledgements

## 6. Change Log

Changes from version 00 to version 01:

- o Added simple list of terms extracted from UC draft -05. It is expected that comments will be received on this list of terms as to whether they should be kept in this document. Those that are kept will be appropriately defined or cited.

Changes from version 01 to version 02:

- o Added Vulnerability, Vulnerability Management, xposure, Misconfiguration, and Software flaw.

Changes from version 02 to version 03:

- o Removed Section 2.1. Cleaned up some editing nits; broke terms into 2 sections (predefined and newly defined terms). Added some of the relevant terms per the proposed list discussed in the IETF 89 meeting.

Changes from version 03 to version 04:

- o TODO

Changes from version 04 to version 05:

- o TODO

Changes from version 05 to version 06:

- o Updated author information.
- o Combined "Pre-defined Terms" with "New Terms and Definitions".
- o Removed "Requirements language".
- o Removed unused reference to use case draft; resulted in removal of normative references.
- o Removed introductory text from Section 1 indicating that this document is intended to be temporary.
- o Added placeholders for missing change log entries.

Changes from version 06 to version 07:

- o Added Contributors section.
- o Updated author list.
- o Changed title from "Terminology for Security Assessment" to "Secure Automation and Continuous Monitoring (SACM) Terminology".
- o Changed abbrev from "SACM-Terms" to "SACM Terminology".
- o Added appendix The Attic to stash terms for future updates.
- o Added Authentication, Authorization, Data Confidentiality, Data Integrity, Data Origin, Data Provenance, SACM Component, SACM Component Discovery, Target Endpoint Discovery.
- o Major updates to Building Block, Function, SACM Role, Target Endpoint.
- o Minor updates to Broker, Capability, Collection Task, Evaluation Task, Posture.
- o Relabeled Role to SACM Role, Endpoint Target to Target Endpoint, Endpoint Discovery to Endpoint Identification.
- o Moved Asset Targeting, Client, Endpoint Identification to The Attic.
- o Endpoint Attributes added as a TODO.
- o Changed the structure of the Change Log.

Changes from version 07 to version 08:

- o Added Assertion, Collection Result, Collector, Excluded Endpoint, Internal Collector, Network Address, Network Interface, SACM Domain, Statement, Target Endpoint Identifier, Target Endpoint Label, Timestamp.
- o Major updates to Attributes, Broker, Collection Task, Consumer, Controller, Control Plane, Endpoint Attributes, Expected Endpoint State, SACM Function, Provider, Proxy, Repository, SACM Role, Target Endpoint.
- o Minor updates to Asset, Building Block, Data Origin, Data Source, Data Provenance, Endpoint, Management Plane, Posture, Posture Attribute, SACM Component, SACM Component Discovery, Target Endpoint Discovery.

- o Relabeled Function to SACM Function.

Changes from version 08 to version 09:

- o Updated author list.
- o Added Data Plane, Endpoint Characterization, Endpoint Classification, Guidance, Interaction Model, Software Component, Software Instance, Software Package, Statement, Target Endpoint Profile, SACM Task.
- o Removed Building Block.
- o Major updates to Control Plane, Endpoint Attribute, Expected Endpoint State, Information Model, Management Plane.
- o Minor updates to Attribute, Capabilities, SACM Function, SACM Component, Collection Task.
- o Moved Asset Characterization to The Attic.

Changes from version 09 to version 10:

- o Added Configuration Drift, Data in Motion, Data at Rest, Endpoint Management Capability, Hardware Component, Hardware Inventory, Hardware Type, SACM Interface, Target Endpoint Characterization Record, Target Endpoint Characterization Task, Target Endpoint Classification Task, Target Endpoint Discovery Task, Vulnerability Description Information, Vulnerability Detection Data, Vulnerability Management Capability, Vulnerability Assessment
- o Added references to i2nsf definitions in Capability, SACM Component, SACM Interface, SACM Role.
- o Added i2nsf Terminology I-D Reference.
- o Major Updates to Endpoint, SACM Task, Target Endpoint Identifier.
- o Minor Updates to Guidance, SACM Component Discovery, Target Endpoint Label, Target Endpoint Profile.
- o Relabeled SACM Task
- o Removed Target Endpoint Discovery

Changes from version 10 to version 11:

- o Added Content Element, Content Metadata, Endpoint Label, Information Element, Metadata, SACM Component Label, Workflow.
- o Major Updates to Assessment, Capability, Collector, Endpoint Management Capabilities, Guidance, Vulnerability Assessment Capabilities, Vulnerability Detection Data, Vulnerability Assessment Capabilities.
- o Minor updates to Collection Result, Control Plane, Data in Motion, Data at Rest, Data Origin, Network Interface, Statement, Target Endpoint Label.
- o Relabeled Endpoint Management Capability, Vulnerability Management Capability, Vulnerability Assessment.

Changes from version 11 to version 12:

- o Added Configuration, Endpoint Characteristic, Event, SACM Content, State, Subject.
- o Major Updates to Assertion, Data in Motion, Data Provenance, Data Source, Interaction Model.
- o Minor Updates to Attribute, Control Plane, Data Origin, Data Provenance, Expected Endpoint State, Guidance, Target Endpoint Classification Task, Vulnerability Detection Data.

Changes from version 12 to version 13:

- o Added Virtual Component.
- o Major Updates to Capability, Collection Task, Hardware Component, Hardware Type, Security Automation, Subject, Target Endpoint, Target Endpoint Profile.
- o Minor Updates to Assertion, Data Plane, Endpoint Characteristics.

Changes from version 13 to version 14:

- o Handled a plethora of issues listed in GitHub.
- o Pruned some commonly understood terms.
- o Narrowing term labels per their definitions.
- o In some cases, excised expositional text.

- o Where expositional text was left intact, it has been separated from the actual definition of a term.

Changes from version 14 to version 16:

- o moved obsolete definitions into the Appendix (attic).

## 7. Contributors

David Waltermire  
National Institute of Standards and Technology  
100 Bureau Drive  
Gaithersburg, MD 20877  
USA

Email: david.waltermire@nist.gov

Adam W. Montville  
Center for Internet Security  
31 Tech Valley Drive  
East Greenbush, NY 12061  
USA

Email: adam.w.montville@gmail.com

David Harrington  
Effective Software  
50 Harding Rd  
Portsmouth, NH 03801  
USA

Email: ietfdbh@comcast.net

Brian Ford  
Lancope  
3650 Brookside Parkway, Suite 500  
Alpharetta, GA 30022  
USA

Email: bford@lancope.com

Merike Kaeo  
Double Shot Security  
3518 Fremont Avenue North, Suite 363  
Seattle, WA 98103  
USA

Email: merike@doubleshotsecurity.com

## 8. References

## 8.1. Normative References

- [RFC5792] Sangster, P. and K. Narayan, "PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC)", RFC 5792, DOI 10.17487/RFC5792, March 2010, <<https://www.rfc-editor.org/info/rfc5792>>.
- [RFC6933] Bierman, A., Romascanu, D., Quittek, J., and M. Chandramouli, "Entity MIB (Version 4)", RFC 6933, DOI 10.17487/RFC6933, May 2013, <<https://www.rfc-editor.org/info/rfc6933>>.

## 8.2. Informative References

- [I-D.ietf-i2nsf-terminology]  
Hares, S., Strassner, J., Lopez, D., Xia, L., and H. Birkholz, "Interface to Network Security Functions (I2NSF) Terminology", draft-ietf-i2nsf-terminology-06 (work in progress), July 2018.
- [I-D.ietf-netmod-entity]  
Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", draft-ietf-netmod-entity-08 (work in progress), January 2018.
- [I-D.ietf-sacm-vuln-scenario]  
Coffin, C., Cheikes, B., Schmidt, C., Haynes, D., Fitzgerald-McKay, J., and D. Waltermire, "SACM Vulnerability Assessment Scenario", draft-ietf-sacm-vuln-scenario-02 (work in progress), September 2016.
- [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>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC5209] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <<https://www.rfc-editor.org/info/rfc5209>>.
- [RFC6192] Dugal, D., Pignataro, C., and R. Dunn, "Protecting the Router Control Plane", RFC 6192, DOI 10.17487/RFC6192, March 2011, <<https://www.rfc-editor.org/info/rfc6192>>.



[X.1252] "ITU-T X.1252 (04/2010)", n.d..

#### Appendix A. The Attic

The following terms are stashed for now and will be updated later:

**Asset:** Is a system resource, as defined in [RFC4949], that may be composed of other assets.

Examples of Assets include: Endpoints, Software, Guidance, or X.509 public key certificates. An asset is not necessarily owned by an organization.

**Asset Management:** The IT process by which assets are provisioned, updated, maintained and deprecated.

**Asset Characterization:** Asset characterization is the process of defining attributes that describe properties of an identified asset.

**Asset Targeting:** Asset targeting is the use of asset identification and categorization information to drive human-directed, automated decision making for data collection and analysis in support of endpoint posture assessment.

**Client:** An architectural component receiving services from another architectural component.

**Endpoint Identification (TBD per list; was "Endpoint Discovery"):**  
The process by which an endpoint can be identified.

#### Authors' Addresses

Henk Birkholz  
Fraunhofer SIT  
Rheinstrasse 75  
Darmstadt 64295  
Germany

Email: [henk.birkholz@sit.fraunhofer.de](mailto:henk.birkholz@sit.fraunhofer.de)

Jarrett Lu  
Oracle Corporation  
4180 Network Circle  
Santa Clara, CA 95054  
USA

Email: jarrett.lu@oracle.com

John Strassner  
Huawei Technologies  
2330 Central Expressway  
Santa Clara, CA 95138  
USA

Email: john.sc.strassner@huawei.com

Nancy Cam-Winget  
Cisco Systems  
3550 Cisco Way  
San Jose, CA 95134  
USA

Email: ncamwing@cisco.com

Adam Montville  
Center for Internet Security  
31 Tech Valley Drive  
East Greenbush, NY 12061  
USA

Email: adam.w.montville@gmail.com

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 14, 2021

S. Banghart  
NIST  
B. Munyan  
A. Montville  
Center for Internet Security  
G. Alford  
Red Hat, Inc.  
July 13, 2020

Definition of the ROLIE configuration checklist Extension  
draft-mandm-sacm-rolie-configuration-checklist-02

Abstract

This document extends the Resource-Oriented Lightweight Information Exchange (ROLIE) core to add the information type categories and related requirements needed to support security configuration checklist use cases. Additional categories, properties, and requirements based on content type enables a higher level of interoperability between ROLIE implementations, and richer metadata for ROLIE consumers. Additionally, this document discusses requirements and usage of other ROLIE elements in order to best syndicate security configuration checklists.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 14, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. The 'configuration-checklist' information-type . . . . .	3
4. rolie:property Extensions . . . . .	4
5. Handling Existing Checklist Formats . . . . .	7
6. atom:link Extensions . . . . .	8
7. IANA Considerations . . . . .	8
7.1. configuration-checklist information-type . . . . .	8
7.2. checklist:contributor property . . . . .	9
8. Security Considerations . . . . .	9
9. Privacy Considerations . . . . .	10
10. References . . . . .	10
10.1. Normative References . . . . .	10
10.2. Informative References . . . . .	10
Appendix A. Examples . . . . .	10
Authors' Addresses . . . . .	11

## 1. Introduction

Default configurations for endpoints (operating systems, applications, etc.) are normally geared towards ease-of-use or ease-of-deployment, not security. As such, many enterprises operate according to guidance provided to them by a control framework ([CIS\_Critical\_Controls], [PCI\_DSS], [NIST\_800-53] etc.), which often prescribe that an enterprise define a standard, security-minded configuration for each technology they operate. Such standard configurations are often referred to as configuration checklists. This document defines an extension to the Resource-Oriented Lightweight Information Exchange (ROLIE) protocol [I-D.ietf-mile-rolie] to support the publication of configuration checklist information. Configuration checklists contain a set of configuration recommendations for a given endpoint. A configuration recommendation prescribes expected values pertaining to one or more discrete endpoint attributes.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The previous key words are used in this document to define the requirements for implementations of this specification. As a result, the key words in this document are not used for recommendations or requirements for the use of ROLIE.

As an extension of [RFC8322], this document refers to many terms defined in that document. In particular, the use of "Entry" and "Feed" are aligned with the definitions presented in section TODO of ROLIE.

Several places in this document refer to the "information-type" of a Resource (Entry or Feed). This refers to the "term" attribute of an "atom:category" element whose scheme is "urn:ietf:params:rolie:category:information-type". For an Entry, this value can be inherited from its containing Feed as per [RFC8322].

Other terminology used in this document is defined below:

**Configuration Item** Generally synonymous with endpoint attribute.

**Configuration Checklist** A configuration checklist is an organized collection of rules about a particular kind of system or platform.

**Configuration Recommendation** A configuration recommendation is an expression of the desired posture of one or more configuration items. A configuration recommendation generally includes the description of the recommendation, a rationale statement, and the expected state of collected posture information.

TODO: There needs to be a "normative" reference to the SCAP 1.2/3 specifications and schema definitions

## 3. The 'configuration-checklist' information-type

This document registers a new information type for use in ROLIE repositories. The "configuration-checklist" information type represents a body of information describing a set of configuration recommendations. A configuration recommendation is, minimally, a single configuration item paired with a recommended value or range of

values. Depending on the source, a configuration recommendation may carry with it additional information (i.e. description, references, rationale, etc.). Provided below is a non-exhaustive list of information that may be considered as components of a configuration checklist.

- o A "Data Stream"
- o A "Benchmark"
- o A "Profile"
- o A "Value"
- o A "Rule" or "Group" of Rules
  - \* Description
  - \* Rationale
  - \* Remediation Instructions
  - \* Information, described in the dialect of a supported "check system", indicating the method(s) used to audit the checklist configuration item.
- o Applicable Platform Information
- o Information regarding a set of patches to be evaluated

#### 4. rolie:property Extensions

A breadth of metadata may be included with a configuration checklist as identifying information. A publishing organization may wish to recognize or attribute checklist authors or contributors, or maintain a revision/version history over time. Other metadata that may be included could indicate the various categories of products to which the checklist applies, such as Operating System, Network Device, or Application Server.

This document registers several new rolie:property elements to express this metadata in a more efficient and automatable form.

- o contributor (0..n)
  - \* name: urn:ietf:params:rolie:property:checklist:contributor

- \* value: Indicates those individuals noted as recognized contributors to the configuration checklist and/or the recommendations contained within. The value MUST be either a plaintext name of a entity, or a link to an <author> element that describes an entity.
- o checklist version
  - \* name: urn:ietf:params:rolie:property:checklist:version
  - \* value: Indicates the version/revision number of the configuration checklist. Implementations MAY choose to incorporate a semantic versioning scheme illustrating "major.minor.point" releases, such as "3.1.1".
- o title
  - \* name: urn:ietf:params:rolie:property:checklist:title
  - \* value: Indicates the document title of the configuration checklist, such as "CIS Benchmark for Microsoft Windows Server 2019".
- o overview
  - \* name: urn:ietf:params:rolie:property:checklist:overview
  - \* value: This property allows for a textual overview and/or introduction to the configuration checklist including, but not limited to, overview of the technology under assessment, limitations or caveats, or assumptions to be made when evaluating the checklist.
- o product name (0..n)
  - \* name: urn:ietf:params:rolie:property:checklist:product-name
  - \* value: This property allows for further refinement and identification of the configuration checklist using the name of the product or products to which the checklist applies, such as Microsoft Windows Server 2019, Red Hat Enterprise Linux, IBM WebSphere Application Server, Google Chrome, etc.
- o product category (0..n)
  - \* name: urn:ietf:params:rolie:property:checklist:product-category

- \* value: This property allows for further refinement and identification of the configuration checklist using the technology category. Examples of product category values may be (but aren't limited to):

- + Antivirus Software
- + Application Server
- + Auditing
- + Authentication
- + Automation/Productivity Application Suite
- + Client and Server Encryption
- + Configuration Management Software
- + Database Management System
- + Desktop Application
- + Desktop Client
- + DHCP Server
- + Directory Service
- + DNS Server
- + Email Server
- + Encryption Software
- + Enterprise Application
- + File Encryption
- + Firewall
- + Firmware
- + Handheld Device
- + Identity Management
- + Intrusion Detection System



- + KVM
- + Mail Server
- + Malware
- + Mobile Solution
- + Monitoring
- + Multi-Functional Peripheral
- + Network Router
- + Network Switch
- + Office Suite
- + Operating System
- + Peripheral Device
- + Security Server
- + Server
- + Virtual Machine
- + Virtualization Software
- + Web Browser
- + Web Server
- + Wireless Email
- + Wireless Network

## 5. Handling Existing Checklist Formats

Today, checklists are distributed in a myriad of different formats, using a variety of organization schemes. This standard attempts to be as flexible as possible in its approach, in order to be usable by as many checklist distributors as possible.

Using the NIST National Checklist Program as a foundation, checklists consist of a primary set of content and a list of supporting content. These pieces of content come in a number of machine readable and

human readable formats, and it is out of scope of this standard to describe guidance for all them. Instead, a best effort should be made to use the available properties, elements, and attributes to describe the content. Moreover, the content is often a compressed file that consists of a package of other content. Likewise, describing this nested structure is out of scope for this standard. Each organization should use a description scheme that best matches their use and business cases, and this description scheme should be documented as thoroughly as possible for all users.

When existing identifiers, titles, authors, and dates are provided in machine-readable forms inside a ROLIE Entry, automated processes can find and acquire checklist content with more ease than the current state-of-the-art methodology. Fully solving the checklist automation problem will require a more significant effort touching on all parts of the checklist ecosystem.

## 6. atom:link Extensions

Name	Description
ancestor	Links to a configuration checklist supersceded by that described in this entry
target-platform	Links to a software descriptor resource defining the software subject to this configuration checklist entry
supporting	Links to a supporting document for the main content. The "title" attribute SHOULD be used to provide a human readable title for this document. Where possible, the "type" attribute MAY be used to describe the type of the supporting document. If the type is a simple IANA Media Type, the media type text should be used, otherwise, a short human readable description should be used.

## 7. IANA Considerations

### 7.1. configuration-checklist information-type

IANA has added an entry to the "ROLIE Security Resource Information Type Sub-Registry" registry located at <https://www.iana.org/assignments/rolie/category/information-type> .

The entry is as follows:

name: configuration-checklist

index: TBD

reference: This document, Section 3

## 7.2. checklist:contributor property

IANA has added an entry to the "ROLIE URN Parameters" registry located in <https://www.iana.org/assignments/rolie/>.

The entry is as follows:

name: property:checklist:contributor

Extension IRI:

urn:ietf:params:rolie:property:checklist:contributor

Reference: This document, Section 4

Subregistry: None

## 8. Security Considerations

Use of this extension requires understanding and managing the security considerations of the core ROLIE specification. Beyond that, there must be considerations made for the common use cases and data types that would be shared with this extension in particular.

Checklist information, while typically shared publicly, can have potential security impact if compromised. In these cases, the utmost care should be taken to secure the REST endpoint. Ensure that only authenticated users are allowed request access to any part of the ROLIE repository. Authentication schemes such as OAUTH or basic HTTP Auth provides a significant barrier to compromise. When providing checklist information as a paid service, security is valuable as a means to protect valuable data from being stolen or taken for free. In these cases, the above strategies still apply, but providers may want to make the Feed visible to non-authenticated users, with meaningful error messages sent to users that have not yet paid for the service.

Typical RESTful security measures applied commonly on the web would be effective to secure this ROLIE extension. As a flexible and relatively simple RESTful service, ROLIE server implementations have great flexibility and freedom in securing their repository.

## 9. Privacy Considerations

This extension poses no additional privacy considerations above and beyond those stated in the core ROLIE specification.

## 10. References

### 10.1. Normative References

- [I-D.ietf-mile-rolie]  
Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange", draft-ietf-mile-rolie-07 (work in progress), May 2017.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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>>.

### 10.2. Informative References

- [CIS\_Critical\_Controls]  
"CIS Critical Security Controls", August 2016, <<https://www.cisecurity.org/critical-controls/>>.
- [NIST\_800-53]  
Hanson, R., "NIST 800-53", September 2007, <<http://deusty.blogspot.com/2007/09/stunt-out-of-band-channels.html>>.
- [PCI\_DSS] "PCI Data Security Standard", April 2016, <[https://www.pcisecuritystandards.org/document\\_library?category=pcidss&document=pci\\_dss](https://www.pcisecuritystandards.org/document_library?category=pcidss&document=pci_dss)>.

## Appendix A. Examples

This section provides some brief examples of a Checklist Information Type ROLIE Entry.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="https://www.w3.org/2005/Atom"
  xmlns:rolie="urn:ietf:params:xml:ns:rolie-1.0" xml:lang="en-US">
  <id>c8db0a93-4dcb-426e-997f-ba43c100b863</id>
  <title>NIST National Checklist for Red Hat Virtualization Host 4.x</title>
  <published>2020-06-29T18:13:51.0Z</published>
  <updated>2020-06-29T18:13:51.0Z</updated>
  <category scheme="urn:ietf:params:rolie:category:information-type" term="checklist"/>
  <summary>SCAP content for evaluation of Red Hat Virtualization Host 4.x systems. The Red Hat content embeds multiple pre-established compliance profiles.</summary>
  <rolie:format ns="scap13namespace"/>
  <content type="application/zip" src="https://nvd.nist.gov/ncp/checklist/908/download/5615"/>
  <link rel="supporting" title="OpenControl-formatted NIST 800-53 responses for Red Hat Virtualization Host 4.x" href="https://github.com/ComplianceAsCode/redhat/tree/master/virtualization-host" type="Machine-Readable Format"/>
  <link rel="supporting" title="[DRAFT] DISA STIG for Red Hat Virtualization Host (RHVH)" href="https://galaxy.ansible.com/RedHatOfficial/rhv4_rhvh_stig" type="Ansible Playbook"/>
  <link rel="supporting" title="VPP - Protection Profile for Virtualization v. 1.0 for Red Hat Virtualization Hypervisor (RHVH)" href="https://galaxy.ansible.com/RedHatOfficial/rhv4_rhvh_vpp" type="Ansible Playbook"/>
  <rolie:property name="urn:ietf:params:rolie:property:content-id" value="908"/>
  <rolie:property name="urn:ietf:params:rolie:property:checklist:checklist-version" value="content v0.1.48"/>
  <rolie:property name="urn:ietf:params:rolie:property:content-published-date" value="2020-01-14T00:00:00+00:00"/>
  <rolie:property name="urn:ietf:params:rolie:property:content-updated-date" value="2019-06-14T00:00:00+00:00"/>
  <rolie:property name="urn:ietf:params:rolie:property:checklist:product-category" value="Virtual Machine"/>
</entry>
```

#### Authors' Addresses

Stephen Banghart  
NIST  
100 Bureau Drive  
Gaithersburg, Maryland 20877  
USA

Email: [stephen.banghart@nist.gov](mailto:stephen.banghart@nist.gov)

Bill Munyan  
Center for Internet Security  
31 Tech Valley Drive  
East Greenbush, NY 12061  
USA

Email: [bill.munyan.ietf@gmail.com](mailto:bill.munyan.ietf@gmail.com)

Adam Montville  
Center for Internet Security

31 Tech Valley Drive  
East Greenbush, NY 12061  
USA

Email: adam.w.montville@gmail.com

Banghart, et al.

Expires January 14, 2021

[Page 11]

Gabriel Alford  
Red Hat, Inc.  
100 East Davie Street  
Raleigh, North Carolina 27601  
USA

Email: [galford@redhat.com](mailto:galford@redhat.com)