INTERNATIONAL STANDARD

**ISO/IEC 19770-2**

Second edition
2015-10-01

Corrected version
2017-02

# Information technology — Software asset management —

## Part 2: Software identification tag

*Technologies de l'information — Gestion de biens de logiciel —*
*Partie 2: Étiquette d'identification du logiciel*



Reference number
ISO/IEC 19770-2:2015(E)



ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

 

# Contents

Page

 

# Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

This second edition cancels and replaces the first edition (ISO/IEC 19770-2:2009), which has been technically revised.

This corrected version of ISO/IEC 19770-2 incorporates the following corrections plus other minor editorial modifications:

— two subclauses have been added to 8.4; and

— the schema for the BaseElement type has been replaced in Annex B.

ISO/IEC 19770 consists of the following parts, under the general title *Information technology — Software asset management*:

— *Part 1: Processes and tiered assessment of conformance*

— *Part 2: Software identification tag*

— *Part 5: Overview and vocabulary*

The following parts are under preparation:

— *Part 3: Software entitlement schema*

— *Part 4: Resource Utilization Measurement (RUM)*

— *Part 7: Tag management*

The following part is planned:

— *Part 22: Guidance for the use of* ISO/IEC 19770-2 *Software Identification Tag information in Cyber Security*

# Introduction

## Overview

International Standards in the ISO/IEC 19770 family of standards for Information Technology (IT) asset management (ITAM) address both the processes and technology for managing software, hardware, and related IT assets. Because IT is an essential enabler for almost all activity in today's world, these standards must integrate tightly into all of IT. For example, software identification (SWID) tags have the capacity to assist in other management functions outside the scope of financial-focused or compliance-focused ITAM processes. From a technology perspective, ITAM standards for information structures provide not only the data interoperability of software management data, but also provide the basis for many related benefits such as more effective security in the management of software. ITAM standards for information structures also facilitate significant automation of IT functionality, such as improved authentication of software and automated linking to identify vulnerability information for more automated exposure identification and mitigation.

## Purpose of this part of ISO/IEC 19770

This part of ISO/IEC 19770 provides an International Standard for software identification tags. The software identification tag is a standardized data structure containing software identification information about a software product that supports new and automated management functions. Product information provided in the software identification tag structure will often be provided in an XML data file, but the same SWID tag product information may be accessible through other means depending on the computing device being managed.

SWID tags are created by a SWID tag producer, for example a software creator who develops and distributes software or a tool and/or service provider. SWID tag data is utilized by SWID tag consumers, for example a discovery tool or service that collects information from a computing device for a variety of purposes such as license compliance, software security, or logistics operations. Providing authoritative and detailed software identification information makes the management of software less expensive and provides support for significantly more automation for IT processes in the security, compliance, and logistics areas.

This part of ISO/IEC 19770 has been developed to facilitate automation of IT processes through the use of software identification tags and for applications which use those tags, for the purposes of security, compliance, and logistics automation. This part of ISO/IEC 19770 includes information which facilitates human intelligibility (such as edition and colloquial version name), but it is unrealistic to expect to create, manage, and use software identification tags without the use of automated capabilities built into specialist or generalist tools. The extent to which such capabilities are provided by specialist commercial products, open-source-type products, or platforms themselves, will depend on market developments over time.

This part of ISO/IEC 19770 supports software asset management processes as defined in ISO/IEC 19770-1. This part of ISO/IEC 19770 is also designed to work together with ISO/IEC 19770-3 which will provide an International Standard for software entitlement schema.

Software identification tags will benefit all stakeholders involved in the creation, licensing, distribution, releasing, installation, and on-going management of software. Key benefits associated with software identification tags include the following.

a) The ability to consistently and authoritatively identify software products that need to be managed for any purpose, such as for licensing, security, logistics, or for the specification of dependencies. Software identification tags provide the meta-data necessary to support more accurate identification than other software identification techniques.

b) The ability to identify groups or suites of software products in the same way as individual software products, enabling entire groups or suites of software products to be managed with the same flexibility as individual products.



 

c) The ability to automatically relate installed software with other information such as patch installations, configuration issues, or other vulnerabilities.

d) Facilitate interoperability of software information between different software creators, different software platforms, different IT management tools, and within software creator organizations, as well as between SWID tag producers and SWID tag consumers.

e) Facilitate automated approaches to license compliance, using information both from the software identification tag and from the software entitlement schema as specified in ISO/IEC 19770-3.

f) Provide a comprehensive information structure of the structural footprint of products, for example the list of software components of files and system settings associated with a product to identify if files have been modified.

g) Provide a comprehensive information structure that identifies different entities, including software creators, software licensors, packagers, distributors external to the software consumer, as well as various entities within the software consumer, associated with the installation and management of the product on an on-going basis.

h) Through the optional use of digital signatures by organizations creating software identification tags, the ability to validate that information is authoritative and has not been maliciously tampered with.

i) The opportunity for entities other than original software creators (e.g. independent providers or in-house personnel) to create software identification tags for legacy software, and for software from software creators who do not provide software identification tags themselves.

This part of ISO/IEC 19770 is divided into the following clauses and annexes:

— Clause 1 defines the scope;

— Clause 2 describes the normative references;

— Clause 3 describes the terms, definitions, and abbreviated terms used in this part of ISO/IEC 19770;

— Clause 4 defines conformance;

— Clause 5 provides interoperability guidance;

— Clause 6 describes the implementation of software identification tagging processes;

— Clause 7 contains platform implementation requirements and guidance;

— Clause 8 describes the elements of the tag;

— Annex A contains information on why the changes to the SWID tag schema are necessary;

— Annex B contains the XML schema document for the tag;

— Annex C provides a UML diagram of the SWID tag schema;

— Annex D provides sample tags.

# Information technology — Software asset management —

# Part 2:
# Software identification tag

## 1 Scope

This part of ISO/IEC 19770 establishes specifications for tagging software to optimize its identification and management.

This part of ISO/IEC 19770 applies to the following.

a) Tag producers: these organizations and/or tools create software identification (SWID) tags for use by others in the market. A tag producer may be part of the software creator organization, the software licensor organization, or be a third-party organization. These organizations and/or tools can broadly be broken down into the following categories.

   1) Platform providers: entities responsible for the computer or hardware device and/or associated operating system, virtual environment, or application platform, on which software may be installed or run. Platform providers which support this part of ISO/IEC 19770 may additionally provide tag management capabilities at the level of the platform or operating system.

   2) Software providers: entities that create, license, or distribute software. For example, software creators, independent software developers, consultants, and repackagers of previously manufactured software. Software creators may also be in-house software developers.

   3) Tag tool providers: entities that provide tools to create software identification tags. For example, tools within development environments that generate software identification tags, or installation tools that may create tags on behalf of the installation process, and/or desktop management tools that may create tags for installed software that did not originally have a software identification tag.

b) Tag consumers: these tools and/or organizations utilize information from SWID tags and are typically broken down into the following two major categories:

   1) software consumers: entities that purchase, install, and/or otherwise consume software;

   2) IT discovery and processing tool providers: entities that provide tools to collect, store, and process software identification tags. These tools may be targeted at a variety of different market segments, including software security, compliance, and logistics.

This part of ISO/IEC 19770 does not prescribe Information Technology Asset Management (ITAM) or other IT-related processes required for reconciliation of software entitlements with software identification tags or other IT requirements.

This part of ISO/IEC 19770 does not specify product activation or launch controls.

This part of ISO/IEC 19770 is not intended to conflict either with any organization's policies, procedures or standards or with any national or international laws and regulations.

## 2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.



 

ISO/IEC 19770–5, *Information technology — Software asset management — Part 5: Overview and vocabulary*

IEEE 1003.1:2013, *Standard for Information Technology — Portable Operating System Interface (POSIX(R))*

W3C Recommendation, *XML Schema Part 2: Datatypes (Second Edition)*

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*

# 3 Terms, definitions, and abbreviated terms

## 3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19770-5 and the following apply.

**3.1.1**
**patch**
software component that, when installed, directly modifies files or device settings related to a different software component without changing the version number or release details for the related software component

**3.1.2**
**platform provider**
organization responsible for the platform

Note 1 to entry: The platform provider is typically the vendor of the relevant operating system, virtual environment, or application platform.

**3.1.3**
**tagId**
globally unique value that shall be globally unique for every SWID tag created

Note 1 to entry: Globally unique values may use a 16 byte GUID, or other globally unique value as defined by the tag creator.

## 3.2 Abbreviated terms

| | |
|---|---|
| API | application programming interface |
| GUID | globally unique identifier |
| IETF | Internet Engineering Task Force |
| MD5 | message digest 5 |
| regid | registration identifier |
| RPC | remote procedure call |
| SAM | software asset management |
| SHA | secure hash algorithm |
| SWID | software identification, or software identification tag |
| URI | uniform resource identifier |
| URL | uniform resource locator |

VAR value added reseller

W3C World Wide Web Consortium

XML extensible markup language

XSD XML schema definition

# 4 Conformance

## 4.1 SWID tag conformance

A software identification tag is in conformance with this part of ISO/IEC 19770 if the tag data structure obeys all normative constraints specified in this part of ISO/IEC 19770.

## 4.2 Application conformance

Application conformance incorporates both syntax and semantics.

— A conforming tag consumer shall not reject any conforming SWID tag.

— A conforming tag producer shall be able to produce SWID tags conforming to this part of ISO/IEC 19770.

— A conforming tag consumer shall treat the information in SWID tag in a manner consistent with the semantic definitions given in this part of ISO/IEC 19770. An application's intended behavior need not require that application to process all of the information in a SWID tag. However, the information that it does process shall be processed in a manner that is consistent with the semantic definitions given in this part of ISO/IEC 19770.

— A conforming tag consumer shall, when necessary, be able to identify the version of the XML schema (XSD) used for a SWID tag and process information provided in older versions of SWID tags in a manner that is consistent with that version of the XSD.

## 4.3 Platform conformance

A platform is in conformance with this part of ISO/IEC 19770 if it provides a programmatic interface to add, retrieve, enumerate, and remove SWID tag data and/or if it provides support for SWID tags to be stored on and retrieved from a file storage environment on a specified device.

# 5 Interoperability guidance

## 5.1 Overview

It is critical that SWID tag producers create SWID tag data structures in a manner so that their tags can be consumed and used by tools and users in a consistent fashion and so that SWID tag consumers understand exactly how they should interpret the relationships defined within the SWID tags. This requires that tags be created in a way that is interoperable with tools and users requirements.

This Clause provides details on how SWID tags are created, what the various relationships defined in the SWID tag mean, and how that information can be interpreted by tools and users.

## 5.2 SWID tag modification

All SWID tags (regardless if they are primary or supplemental tags) shall only be modified by the organization that initially created the tag; this is to ensure that data, especially digitally signed data, is not modified in any way that the tag creator is not directly responsible. There are many instances

when additional data needs to be associated with an existing SWID tag, so the SWID structure allows any organization to create their own supplemental SWID tag that references another SWID tag (either primary or supplemental SWID tags can be referenced). This allows, for example, a software purchasing organization to associate specific licensing data for a user or device to a software title.

In many instances, the SWID tag will be created and maintained by the software developer. However, in cases where software is discovered that does not have a SWID tag, there will be instances where a discovery tool identifies software on a device and creates a SWID tag for that software. In either case, the entity that creates the initial SWID tag is creating a primary SWID tag and the data provided in the tag shall not be modified by any organization other than the organization specified as the "tagCreator" role of the Entity data attribute (see 8.5.2). For an example of a tag created by a discovery tool, see D.4.

There are many instances where additional information needs to be associated with an existing primary SWID tag. Since SWID tags may not be modified by anyone other than the Entity identified with the tagCreator role and securely signing a SWID tag update in the field may be difficult, supplemental tags are utilized to provide additional data. Supplemental tags may be used to provide details such as software activation information, or specific customer related information such as Information Technology Infrastructure Library (ITIL)-based release testing and rollout details for a specific software product.



## 5.3 SWID tag relationships

### 5.3.1 Overview

SWID tags are used to represent the software elements that are part of a software product. There may be multiple different SWID tags that define a software product, the various components that make up the software product, as well as which patches are related to the software product. The relationships between these multiple SWID tags are defined through SWID links that are part of the SWID data structure.

There are three special types of relationships that are uniquely defined in SWID tags: SoftwareIdentity attributes that is for pre-installation data; a software patch and for supplemental tag data.

### 5.3.2 Pre-installation data attribute

When software is distributed from a software publisher, it is typically provided in a "pre-installation" structure and includes an installation script. This type of distribution may be provided on removable media or through downloaded file. In these instances, there are often times a tag consumer will want to know details about the software that is available to install. SWID tags identifying pre-installation distributions of software can be provided and are identified if the attribute corpus is set to true (see 8.5.1).

### 5.3.3 SWID patch attribute

When a patch is being identified by a SWID tag, it shall include the attribute "patch" with the value being set to true (see 8.5.1).

SWID tags included with patches shall include a link to the product or products (see 8.5.4) it patches as well as links to any other patches to which the current patch may have a relationship. The relationships (see 8.6.7) that may be used by a patch are as follows:

— patches – every patch with a SWID tag must include a Link to the product(s) it patches and the link must use the rel value of "patches";

— requires – the new patch requires that the earlier patch be installed first. In this case, the new patch may only be installable if the earlier patch is installed;

— supersedes – the new patch includes all files from an earlier patch. The new patch (that supersedes the older patch) may be installed on its own, or after the earlier patch and either installation path will result in the same resulting state of the software product that is being patched.

 

Patches that do not have either link with a relationship of supersedes or requires may be installed independently of each other and in any order. Figure 1 provides an example of how the SWID tags for patches work.

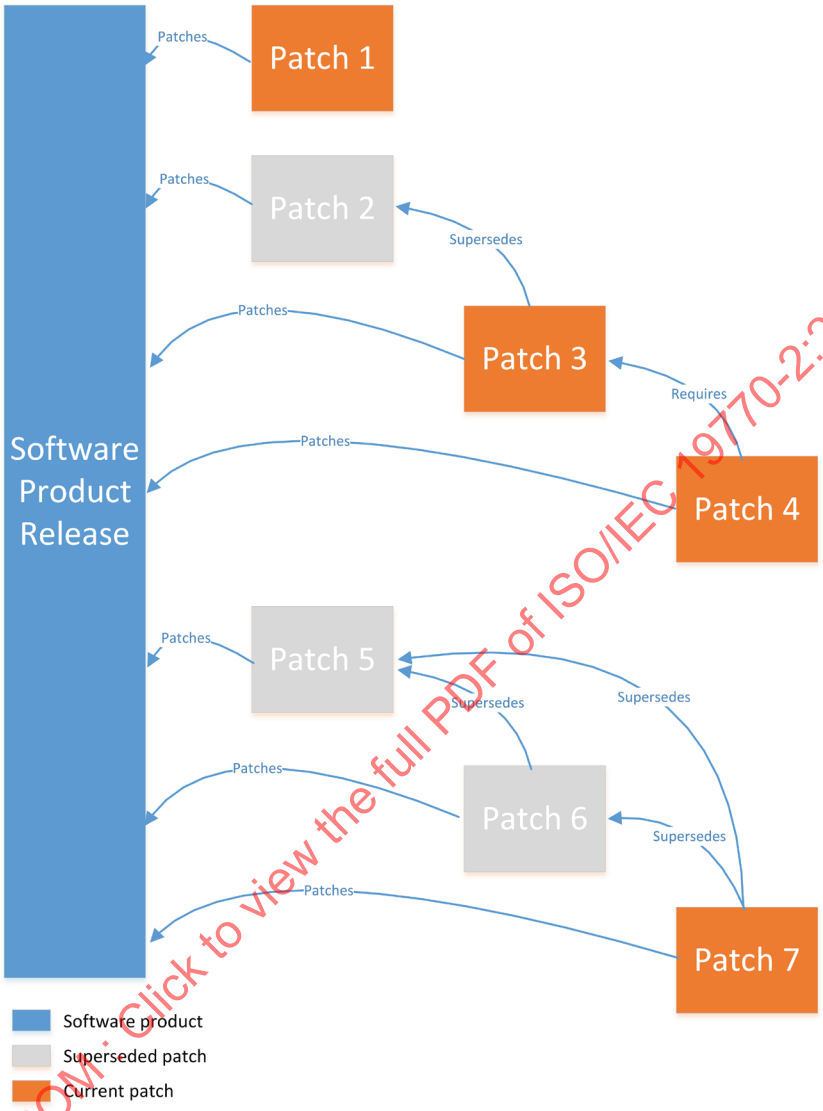


**Figure 1 — Representation of patch and product relationships**



### 5.3.4 SWID supplemental attribute

Supplemental tag data is data that is directly associated with a specific software product's primary tag but, for various reasons, the data included in the supplemental tag is not included in the primary SWID tag. SWID tag data may only be modified by the tagCreator; in other words, if a software creator provides a primary SWID tag for their product, the software consumer who installs and manages that software is not allowed to modify any data in the primary SWID tag. In this case, the software tag consumer can create a supplemental tag that provides specific details for the primary SWID tag they are referencing and they will set the attribute "supplemental" to the value of true (see 8.5.1). This supplemental tag can then be deployed with the installation of the software, or added after the fact as part of a device management process, or a software activation process.

Supplemental tags may also be provided by the tag creator to add additional information related to a specific installation of a software title.

# 6 Implementation of software identification tagging processes

## 6.1 General requirements and guidance

### 6.1.1 XML and XSD

The software identification tag file shall be defined as an XML data structure. The XML schema definition (XSD) as specified in this revision may be downloaded from

http://standards.iso.org/iso/19770/-2/2015/schema.xsd

In anticipation of possible defects which will be identified after publication, an alternative download location always with the latest defect-corrected version is

http://standards.iso.org/iso/19770/-2/2015-current/schema.xsd

### 6.1.2 SWID tags based on earlier revisions of this part of ISO/IEC 19770

A SWID tag (or tags) installed as part of a specific software product are assumed to continue to exist until the software components installed as part of that release are modified in some way. Tag consumers should be aware that earlier versions of the XML schema definition (XSD) may exist for older software products and that the older schema can be located at standards.iso.org.

Tag producers and tag consumers who are creating tags based on this revision are not required to also provide a SWID tag based on any older revisions.

### 6.1.3 SWID tag installation and removal

In instances where a software product is installed on a device, a software licensor conforming to this standard will ensure that a primary SWID tag is included on the installation media and installed at the same time the software is installed.

When software is uninstalled, or changed to a different release, the old SWID tags shall be removed from the device.

In instances where a patch is installed, the patch will include a patch SWID tag that will be installed when the patch is installed, and in most cases should be removed when either the patch is uninstalled, or when the product is uninstalled, or changed to a different release. The determination if a patch tag is to be removed, or not, is based on additional data provided in the ownership attribute (8.6.3).

Supplemental tags provided by the software publisher (which may be used to identify relationships between software products) shall be managed in a manner similar to primary and patch SWID tags such that the supplemental tags should be removed from a device when the software product is uninstalled.

As noted in 6.1.4, SWID tags reside in the same directory tree as the applications installation directory tree. It is expected that if an application directory tree is deleted when an application is uninstalled, that the SWID tags associated with that application (including primary, supplemental, and patch tags) are deleted as well.

### 6.1.4 SWID data storage and transmission

On devices with a file system, but no API defined to retrieve SWID tags, the SWID tag data shall be stored in an XML file and shall be located on a device's file system in a sub-directory named "swidtag" (all lower case) that is located in the same file directory or sub-directory of the install location of the software component with which they are installed. It is recommended, but not required, that the swidtag directory is located at the top of the application installation directory tree. Any payload information provided shall reference files using a relative path of the location where the SWID tag is stored.



 

On devices that do not have a file system, the SWID tag data shall be stored in a data storage location defined and managed by the platform provider for that device. SWID tag data stored in this manner may be transmitted using an alternative format in specific use cases. This may include a computing device that utilizes an API to store and retrieve SWID tag information and/or data-interchange formats other than XML.

On devices that utilize both a file system for software installation as well as API access to the SWID tag files, it is recommended that the SWID tag data be stored in the API managed repository as well as stored as a file on the system. This allows older discovery tools that may not utilize API support to continue to be able to identify SWID tag files on a device.

Finally, the SWID tag data may also be accessible via a URI, or other means (such as using a common information model, e.g. Reference [12]).

The platform provider may provide access to the software identification tag using methods that are available through APIs, RPCs, command line interface, or other programmatic means.

### 6.1.5 Unique registration ID (regid)

#### 6.1.5.1 Overview

Software identification tags may be created by different organizations and do not strictly require a centralized registration authority. Additionally, this part of ISO/IEC 19770 allows entities to create software identification tags for software components they did not create (such as an organization creating software identification tags for their internal software discovery processes). To accommodate these requirements, this part of ISO/IEC 19770 uses a regid. The regid provides a unique naming authority identifier.

#### 6.1.5.2 Structure of regid

A regid shall use a URI reference (see IETF RFC 3986). Once an organization specifies a regid for their organization's software, that regid shall be used consistently for all software from the organization.

To ensure interoperability, allow for open source project support and third party tag consistency, the following recommendations should be applied when creating a regid.

— Unless otherwise required, the URI should utilize the http scheme.

— If the http scheme is used, the "http://" may be left off the regid string (a string without a URI scheme specified is defined to use the "http://" scheme).

— Unless otherwise required, the URI should use an absolute-URI that includes an authority part, such as a domain name.

— To ensure consistency, the absolute-URI should use the minimum string required (for example, example.com should be used instead of www.example.com).

#### 6.1.5.3 Examples of regid

A regid for a company that creates and sells software is expected to be the HTTP reference to that company. So a regid for the Fabrikam Company is

"fabrikam.com"

A regid for an open source project called SampleProject that is hosted on soureceMyProject.net may be one of the following:

sampleproject.sourcemyproject.net

or

sourcemyproject.net/sampleproject

or

sourcemyproejct.net/?projectname=sampleproject

The appropriate regid to choose for the project is dependent on the default reference used by the hosting site for the particular project.

### 6.1.6 Tag identifier

For the purposes of uniqueness, tagId shall be a globally unique value that shall be unique for every SWID tag created. It is recommended that tagId be a minimum of a 16 byte globally unique identifier (GUID), but it may also be a string that is constructed by the tag provider or another globally unique value. If a string construct is used, the tagId shall follow the restrictions for URI character use as specified in IETF RFC 3986, characters. For more information on universally unique identifiers (of which a GUID is one type of UUID), see Reference [2].

### 6.1.7 Unique software identification tag file name

SWID tag files may be located on the installation medium or on a device where software has been installed. There is no requirement to use different filenames that identify software in a pre-installation state as compared to tags located on a computing device due to software being installed or discovered. The difference between SWID tags for software in a pre-installation state and software that is installed is that software in a pre-installation state will have the attribute corpus set to true.

The following rules and recommendations shall be used when creating filenames.

Filenames should be restricted to use only the characters listed in the Portable Filename Character Set defined in IEEE 1003.1:2013, 3.278 to maximize interoperability between platforms. If this limitation is too restrictive, the tag creator shall ensure that the characters used in the filename are valid characters for all platforms where their SWID tags may be stored on a file system.

SWID tag base filenames (i.e. the filename without the .swidtag extension) shall be structured to be globally unique for the tag creator and product.

SWID tag creators may use different approaches to defining the base portion of the SWID tag filename; however, if the filename aligns with the following structure, the filename will be unique for the product and recognizable by a system administrator

<name of the tag creator> + <product name>.swidtag

The .swidtag file extension shall be used for all software identification tags.

### 6.1.8 Software identification tag discovery

The primary requirement is that the SWID tag shall be discoverable and be able to be read on the machine, distribution medium, and/or virtual environment on which the software package is installed by a suitably privileged user and/or discovery process. This includes the ability to utilize SWID tag information prior to software being installed to allow for capabilities such as validating installation files prior to software being installed on a system.

### 6.1.9 Languages

This part of ISO/IEC 19770 recognizes that software creators produce software with builds that are specific for a particular language while other software creators produce software with one build that implements add-on “language packs”. However, this part of ISO/IEC 19770 does not require software identification tags to recognize different language versions of the same product. Language information will be more efficiently managed through the use of a supplemental tag where the supplemental tag indicates specific language support and references the primary SWID tag.



 

See Annex D for sample tags.

For encoding purposes, the use of UTF-8 (see Reference [15]) is the suggested methodology for software identification tags created based on this part of ISO/IEC 19770 (see Reference [14]).

### 6.1.10 Authenticity of software identification tags

Authenticity of an SWID tag, for example, to validate that the tag collected during a discovery process has not had specific elements of the tag altered, may be proven through the use of digital signatures within the tag.

Signatures are not a mandatory part of the software identification tag standard and can be used as required by any tag producer to ensure that sections of a tag are not modified and/or to provide authentication of the signer. If signatures are included in the software identification tag, they shall follow the W3C recommendation defining the XML signature syntax which provides message integrity authentication, as well as signer authentication services for data of any type.

The W3C Recommendation, XML Signature Syntax, and Processing Version 1.1 should be used with the canonicalization version 1.1 algorithm for digital signatures. Tool providers should be aware that Version 1.1 of the signing process is likely to have a signature corrupted if there are changes in whitespace within the signed elements of a SWID tag. This is primarily a consideration for discovery tools and systems. The recommendation is that at least one master, unaltered copy of each SWID tag be retained in the tools database for signature validation purposes.

When a signature is utilized for a SWID tag, the signature shall be an enveloped signature and the digital signature shall include a timestamp provided by a trusted timestamp server. This timestamp shall be provided using the XAdES-T form. Information on this form of timestamp can be found in Reference [13]. The SWID tag shall also include the public signature for the signing entity.

The requirement for a digitally signed SWID tag is that a SWID tag consumer shall be able to utilize the data encapsulated by the SWID tag to ensure that the digital signature was validated by a trusted certificate authority (CA), that the SWID tag was signed during the validity period for that signature and that no signed data in the SWID tag has been modified. All of these validations shall be able to be accomplished without requiring access to an external network. If a SWID tag consumer needs to validate that the digital certificate has not been revoked, then it is expected that there will be access to an external network or a data source that can provide revocation information.

For additional information on digital signatures, how they work and the minimum requirements for digital signatures used for US Federal Government processing, please review Reference [6].

### 6.1.11 File hash definitions

There are multiple different file hash algorithms in use throughout the industry. These algorithms have a variety of benefits and are used by a variety of different organizations. This part of ISO/IEC 19770 does not mandate that one version of a hash algorithm be utilized. The following are commonly used XML namespace references for typical hash algorithms currently in use:

— xmlns:MD5="http://www.w3.org/2001/04/xmldsig-more#md5";

— xmlns:SHA-1="http://www.w3.org/2000/09/xmldsig#sha1";

— xmlns:SHA-224="http://www.w3.org/2001/04/xmldsig-more#sha224";

— xmlns:SHA-256="http://www.w3.org/2001/04/xmlenc#sha256";

— xmlns:SHA-384="http://www.w3.org/2001/04/xmldsig-more#sha384";

— xmlns:SHA-512="http://www.w3.org/2001/04/xmlenc#sha512";

— xmlns:RIPEMD-160="http://www.w3.org/2001/04/xmlenc#ripemd160".

Additional hash algorithms may be developed and used over time and organizations may use the appropriate hash algorithm for their needs.

For security focused use, a minimum of the SHA-256 algorithm is recommended at the time of publication for this part of ISO/IEC 19770. The minimum algorithm for hashes may change as new algorithms are developed, validated, and programmatic libraries become available.

### 6.1.12 Use of standardized data types in XSD definition

There are a number of standardized types used in the software identification tag including specific date/time entries that shall follow formats as specified in the W3C recommendation titled, "*XML Schema Part 2: Datatypes Second Edition*". By using specific types as specified in the W3C recommendation, software identification tags can utilize an automated validation step that allows a much more consistent structure to the data provided.

### 6.1.13 Using Evidence or Payload

The evidence element (8.5.3) is similar to a payload element (8.5.6) in that these two elements contain information about resources (files, execution processes, and other system settings) that may be installed or discovered on a device. There is a key difference between Evidence and Payload.

The Payload element indicates the resources that may be installed on a device when the software is installed. Data in this subclause can be used, for example, to determine if files in a directory match the files that were designated as being installed with a software component or software product.

The Evidence element is used by discovery tools that identify software that does not include a SWID tag, and for which the tool creates a SWID tag based on discovery data. In this case, the element Evidence indicates what was discovered on the device, but it is not data that can be used to determine if files match what a publisher specified since there was no SWID tag to begin with. Evidence may be provided by systems that are discovering software dynamically and providing the data to a server, or may be used by a system to send data to a server as evidence of what was found. In the case where a SWID tag with the evidence element included is provided to the server, data from the evidence elements will be analysed either manually or in an automated fashion and a SWID tag identified as appropriate for the collected evidence. This SWID tag will be provided with the expectation that the server will provide that SWID tag back to the client to be placed on the device indicating that a specified software component or product was identified by the discovery tool.

### 6.1.14 Redistributable software components

Software creators regularly develop software components that may be redistributed by other organizations. Redistribution is done through entitlements and may include, but is not limited to, organizations making redistributable libraries and organizations making OEM software.

In either case, specifics of what is included in the redistributable package (for example, who is listed as the software creator and who is the tag creator) are agreed to by the parties involved and are not addressed in this part of ISO/IEC 19770.

Once agreed, SWID tags that are distributed through redistributable software components may be referenced just as any other SWID tag is referenced using the Link element. This allows, for example, a language library that is redistributed by an operating system manufacturer to be distributed by a tool provider with the tool including a "Link" element identifying a relationship to the library. See 8.5.4 for more information on the Link element and Annex D for examples of how redistributable packages may be referenced by a software product.

## 7 Platform requirements and guidance

The information contained within a software identification tag should be independent of the platform on which the software component that the tag references is installed. A platform, for example an



 

operating system, virtual environment, or application platform, should define processes to store and retrieve these software identification tags efficiently.

Software identification tags may be managed using different methods depending on the options a platform makes available for SWID tag management, as well as the options the software installation process utilizes for SWID tag management. The options shown in Table 1 may exist for every platform with the expectation that over time, more platforms and more software installation scripts will utilize the more effective options for SWID tag management and reporting:

**Table 1 — Platform guidance for access to SWID tags**

| Method | Description |
|---|---|
| Programmatic Interface<br><br>(SWID tag APIs) | A programmatic interface allows application software to call an operating system level service to manage a SWID tag from a particular platform.<br><br>If a programmatic interface is supported and the platform supports the concept of a file system, a SWID tag should still be stored as a file in the software applications program file directory (see 6.1.4) to ensure that tag consumers who rely on access to the files can still process SWID tag data properly. The data provided in the SWID tag can additionally be stored in a data storage location as specified and developed by the platform owner.<br><br>Using this approach, when an installation process uses the interface, additional information can be tracked and retained for all software installed, patched, upgraded, and removed from a system. Additional information that may be tracked includes the following:<br><br>— user id used to install or remove a software component;<br><br>— whether the installation process was initiated from the local system or remotely;<br><br>— installation instance id (if there is more than one installation);<br><br>— location of the installed software;<br><br>— date and time software was installed or removed from the device.<br><br>A programmatic interface can also provide a more effective, robust, and efficient software discovery process. Efficiencies may include the following:<br><br>— compressing all transmitted SWID tag data;<br><br>— transmitting SWID tag data only when the data changes;<br><br>— automatically validating that all software on a device has an appropriate level of SWID tag as defined by tag consumer policy (for example, ensuring that all SWID tags are digitally signed);<br><br>— automatically validating if executable files are stored on a device, but are not included in any SWID tag package_footprint. |
| SWID tag file stored with installed application | When a SWID tag file is stored in the file system of an application, the SWID tag file shall be located on the device's file system in a sub-directory named "swidtag" (all lower case) that is located in the same file directory or sub-directory of the install location of the software component the SWID tag represents. It is recommended, but not required, that the swidtag directory is located at the top of the application installation directory tree whenever possible. Any payload information provided must reference files using a relative path of the location where the SWID tag is stored.<br><br>EXAMPLE If a program called Fabrikam Suite is installed on a device in the "c:\Program Files\Fabrikam Suite" directory, the SWID tag will be located in the following directory:<br><br>C:\Program Files\Fabrikam Suite\swidtag |



# 8 Elements

## 8.1 General

It is recommended that tag producers maintain a central repository of all software identification tags created for all product releases. This repository can then be used to validate the uniqueness of GUIDs (see ISO/IEC 19770-5, as well as the details for tagId in 8.5.1) and validate that other elements

are normalized such as the roles; softwareCreator, tagCreator, and licensor. This part of ISO/IEC 19770 does not require an external registration agency for software identification tags, so it is up to each tag producer to ensure each of their tags is unique.

Data values and types are defined in 8.5 and 8.6. The examples are specified in XML syntax with the format that shall be used for software identification tag creation. The examples provide additional insight as to what information is to be included within data values.

Data values (see 8.5) are XML elements of the SoftwareIdentity element that provide the specific identification information about a software component. Data values include details such as the name of the software component, the version, if the software component is a patch, etc.

Data types (see 8.6) are XML types utilized by the data elements and attributes.

## 8.2 Minimum SWID tag data values required

Due to the multiple use cases identified for SWID tag creation, the minimum data requirements for a SWID tag are relatively sparse. The only values that are required for a SWID tag to be considered "valid" to meet the requirements of the XML schema shall be the following:

— SoftwareIdentity (8.5.1):
  — name;
  — tagId;
— Entity (8.5.2):
  — role of TagCreator (at a minimum);
  — regid of TagCreator (at a minimum);
  — name of TagCreator (at a minimum).

In addition, there are a number of SWID tag attributes in the SoftwareIdentity element that have default values. These include the following:

— SoftwareIdentity (8.5.1):
  — Entity.regid – default value is http://invalid.unavailable;
  — patch – default value is false;
  — supplemental – default value is false;
  — tagVersion – default value is 0;
  — version – default value is 0.0;
  — versionScheme – default value is multipartnumeric.

These default values are specified so that if no value is included for these attributes, the SWID tag is considered to be the first version of a primary tag and that the software product has the version number of 0.0.

Although a SWID tag specifies only the software component name, the tagId and tagCreator is a valid SWID tag file that does not mean that the information is sufficient for processes where SWID tags are used. 8.3 provides details on which SWID tag data IT organizations will expect to see included in the SWID tags they receive.



 

## 8.3 Recommended SWID tag data values

Due to the various use cases that apply to a SWID tag, there are a number of data attributes that are specified as optional because they may not be available in certain cases (for example, an organization may not specify a version number for a product – it is unusual, but possible that this element will be unspecified or unknown by a tool or service provider who may be creating a SWID tag for a software product).

For most software products, however, the data elements specified below should be made available whenever possible. This allows organizations to more completely automate their IT processes.

a) SoftwareIdentity (8.5.1):

   1) name;

   2) tagVersion;

   3) tagId;

   4) version;

   5) versionScheme;

b) Entity (8.5.2):

   1) name;

   2) regid;

   3) role (minimum requirement is that the tagCreator role is specified).

As well, the following data should be provided whenever possible to assist in IT automation processes:

a) SoftwareIdentity (8.5.1):

   1) patch (if this tag provides data about a patch);

   2) supplemental (if this tag provides supplemental information to another tag);

b) Entity (8.5.2 whenever possible, provide the softwareCreator and licensor role data as well);

c) Meta (8.5.5);

   1) product (this is the product name only, not including the edition, colloquial version, or services pack information).

If the software specified in this SWID uses the following type of naming structures, they should be provided both for human usability and so that other IT systems can utilize the information for automation purposes:

— edition;

— colloquialVersion;

— revision.

## 8.4 XML element and attribute names

### 8.4.1 Introduction

Software identification tag content shall be identified in accordance with the XML element and attribute names specified in 8.5. This naming requirement ensures consistent interoperability of software identification tag content, regardless of the creator or consumer of the tag data.

Data values in the SWID tag may be structured as XML elements (which contain XML attributes), or XML attributes (which contain the actual data values). The following convention for the left-hand column in the tabular structures in 8.5 and 8.6 differentiates attributes and Elements.

Elements are Pascal-cased (ThisIsAnExample)

Attributes are camel-cased (anotherExampleIsLikeThis)

Additionally, required elements are shown in bold.

Unless specified otherwise, elements may be repeated as many times as required by the SWID tag creator.

### 8.4.2 Additional attributes allowed

This part of ISO/IEC 19770-2 allows the following W3C attributes to be used for any element:

— lang – this attribute specifies the language of the string specified in the element where it is specified as well as defining the lang for all child elements unless specified otherwise.

— id – this attribute provides a unique id for the element that is unique within the XML file. The id attribute may be used for XML XPATH arguments, or for other XML processing as required by tool vendors.

— <namespace:any> - any other attribute can be included for an element if the attribute is defined with a namespace. This allows vendors, customers and tools to include unique attributes that may or may not be generally known others in the market.

The above changes clarify the intent of the attributes that are available to use, includes the id attribute and indicate to all readers that any other attributes can be included if desired.

## 8.5 Data values

### 8.5.1 SoftwareIdentity

| **Element name** | SoftwareIdentity | | |
|---|---|---|---|
| **Description** | Represents the root element specifying data about a software component.<br>NOTE A software product may be made up of one or multiple software components. Also, Software components may be atomic, or may be made up of multiple components. Each component will have its own SWID tag and only one SoftwareIdentity will exist for any one component. | | |
| **attribute or Element** | **Type** | **Required/ default** | **Definition** |
| **Entity** | **Entity**<br>(see 8.5.2) | **An entry for "tagCreator" is required, all others are optional** | **Element that specifies the organizations responsible for this SWID tag.** |

| | | | |
|---|---|---|---|
| Evidence | ResourceCollection<br>NOTE Evidence references the section titled Evidence to differentiate it from Payload; however, it uses the type ResourceCollection, and adds two values – date and device. (see 8.5.3) | optional | Element used to provide results from a scan of a system where software that does not have a SWID tag is discovered. This type of information is not provided by the software creator and is instead created when a system is being scanned and the evidence for why software is believed to be installed on the device is provided in the Evidence element.<br>NOTE Evidence is the same as payload with the exception that it also includes the date and device the evidence was collected. |
| Link | Link<br>(see 8.5.4) | optional | Element to reference any other item (can include details that are related to the SWID tag such as details on where software downloads can be found, vulnerability database associations, use rights, etc.).<br>NOTE This is modelled directly to match the HTML [LINK] element; it is critical for streamlining software discovery scenarios that these are kept consistent. |
| Meta | SoftwareMeta<br>(see 8.6.11) | optional | Element that allows arbitrary key/value pair data related to this SWID.<br>The attributes shown below are predefined attributes to ensure common usage across the industry. The schema allows for any additional attribute to be included in a SWID tag, though it is recommended that industry norms for new attributes are defined and followed to the degree possible. |
| Payload | ResourceCollection<br>NOTE Payload references the section "Payload" to differentiate it from Evidence; however, it utilizes the type "ResourceCollection".<br>Payload<br>(see 8.5.6) | optional | Element that specifies the items that may be installed on a device when the software is installed. Note that Payload may be a superset of the items installed and, depending on optimization systems for a device, may or may not include every item that could be created or executed on a device when software is installed.<br>In general, payload will be used to indicate the files that may be installed with a software product and will often be a superset of those files (i.e. if a particular optional component is not installed, the files associated with that component may be included in payload, but not installed on the device). |
| corpus | Boolean | false | If set to true, this attribute specifies that this SWID tag is a collection of information that describes the pre-installation data of software component. |
| patch | Boolean | false | If set to true, this attribute describes a product patch or modification to a different software element. |
| media | string<br>(Media) | optional | 'media' provides a way to describe the characteristics of the platform this SWID tag applies to (see the [Link] element 8.5.4 'media' attribute). |

| | | | |
|---|---|---|---|
| **name** | **String** | **required** | This attribute provides the software component name as it would typically be referenced. For example, what would be seen in the add/remove dialog on a Windows device, or what is specified as the name of a packaged software product or a patch identifier name on a Linux device. |
| supplemental | Boolean | false | If set to true, this tag specifies supplemental tag data that can be merged with primary tag data to create a complete record of the software information.<br><br>Supplemental tags will often be provided at install time and may be provided by different entities (such as the tag consumer, or a Value Added Reseller). |
| **tagId** | **String** | **required** | tagId shall be a globally unique identifier and should be assigned a GUID (see ISO/IEC 19770-5 definition for GUID).<br><br>The tagId provides a unique reference for the specific product, version, edition, revision, etc.<br><br>If two tagId's match and the tagCreator is the same, the underlying products they represent are expected to be the same. It is recommended that if a software release is available for multiple platforms, that each platform have its own unique tagId. However, it's recognized that some existing development processes may use the same tagId for the same release that may be available on different platforms.<br><br>This allows IT systems to identify if a software item (for example, a patch) is installed simply by referencing the specific tagId value which is likely to be readily available in a software inventory.<br><br>It is recommended, when possible, that a 16 byte GUID be used for this field; this provides global uniqueness without a significant amount of overhead for space.<br><br>If use of a 16 byte GUID is not possible, a text-based globally unique ID may be constructed, this ID should include a unique naming authority for the tagCreator and sufficient additional details that the tagId is unique for the software product, version, edition, revision, etc. This would likely look as follows (+ is used as a string concatenation symbol):<br><br>regid + productName + version + edition + revision + … |

| | | | |
|---|---|---|---|
| tagVersion | integer | 0 | The tagVersion indicates if a specific release of a software product has more than one tag that can represent that specific release. This may be the case if a software tag producer creates and releases an incorrect tag that they subsequently want to fix, but with no underlying changes to the product the SWID tag represents. This could happen if, for example, a patch is distributed that has a Link reference that does not cover all the various software releases it can patch. A newer SWID tag for that patch can be generated and the tagVersion value incremented to indicate that the data is updated. |

| | | | |
|---|---|---|---|
| version | string | 0.0 | Underlying development version for the software component. |
| versionScheme | VersionScheme (see 8.6.13) | multipartnumeric | Scheme used for the version number. |

**EXAMPLE**

```
<SoftwareIdentity
      xmlns=~"http://standards.iso.org/iso/19770/-2/2015/schema.xsd~"
      xmlns:xsi=~"http://www.w3.org/2001/XMLSchema-instance~"
      xmlns:ds=~"http://www.w3.org/2000/09/xmldsig#~"
      xsi:schemaLocation=~"http://standards.iso.org/iso/19770/-2/2015/schema.xsd~"
      name=~"ACME System Protection~"
      tagId=~"iso-sid-app-acme-endpoint-protection-v12-1-mp1~"
      version=~"12.1.1~"
      versionScheme=~"multipartnumeric~"/>
```

An example of a corpus tag would look something like the following:

```
<?xml version=~"1.0~" encoding=~"utf-8~"?>
<SoftwareIdentity
      xmlns=~"http://standards.iso.org/iso/19770/-2/2015/schema.xsd~"+
      xmlns:xsi=~"http://www.w3.org/2001/XMLSchema-instance~"
      xmlns:ds=~"http://www.w3.org/2000/09/xmldsig#~"
      xmlns:SHA256=~"http://www.w3.org/2001/04/xmlenc#sha256~"
      xsi:schemaLocation=~"http://standards.iso.org/iso/19770/-2/2015/schema.xsd~"
      name=~"ACME System Protection~"
      tagId=~"iso-sid-app-acme-endpoint-protection-v12-1-mp1~"
      version=~"12.1.1~"
      versionScheme=~"multipartnumeric~"
      corpus=~"true~">
<Payload>
<File name=~"EPV12.cab~" size=~"1024000~"
SHA256:hash=~"a314fc2dc663ae7a6b6bc6787594057396e6b3f569cd50fd5ddb4d1bbafd2b6a~" />
<File name=~"installer.exe~" size=~"524012~"
SHA256:hash=~"54e6c3f569cd50fd5ddb4d1bbafd2b6ac4128c2dc663ae7a6b6bc67875940573~" />
</Payload>
</SoftwareIdentity>
```



### 8.5.2 Entity

| **Element name** | Entity | | |
|---|---|---|---|
| **Description** | Specifies the organizations related to the software component referenced by this SWID tag. | | |
| **attribute or Element** | **Type** | **Required/ default** | **Definition** |
| Meta | Meta | optional | An open-ended collection of elements that can be arbitrary metadata about an entity. |

| **name** | **string** | **required** | The name of the organization claiming a particular role in the SWID tag. |
|---|---|---|---|
| regid | String<br>(see 6.1.5) | http://invalid.unavailable | The regid of the organization. If the regid is unknown, the value "http://invalid .unavailable" is the default value (see Reference [4]) for more details on this value). |
| **role** | **role**<br>**NMTOKEN**<br>(see 8.6.10) | **tagCreator role is required** | The relationship between this organization and this tag e.g. tag, softwareCreator, licensor, tagCreator, etc.<br>The role of tagCreator is required for every SWID tag.<br>Role may include any role value, but the pre-defined roles include the following:<br>— aggregator;<br>— distributor;<br>— licensor;<br>— softwareCreator;<br>— tagCreator.<br>Other roles will be defined as the market uses the SWID tags. |
| thumbprint | string | optional | If the SWID tag is signed, this value provides a hexadecimal string that contains a hash (or thumbprint) of the entities certificate. This allows the digital signature to be directly related to the entity specified. The SWID entity element that includes thumbprint shall be included in the signature for the relationship to have any validity. |

**EXAMPLE**

```
<Entity
   name=~"Acme, Inc.~"
   regid=~" acme.com~"
   role=~"softwareCreator licensor tagCreator~"/>
<Entity
   name=~"Fabrikam, Inc.~"
   regid=~"fabrikam.com~"
   role=~"distributor~"/>
```

### 8.5.3 Evidence

| Element name | Evidence<br>(type ResourceCollection with two additional data elements) | | |
|---|---|---|---|
| **Description** | Only one entry of the Evidence element may be included inside the SoftwareIdentity Tag.<br>Evidence is very similar in structure to the Payload element with both Evidence and Payload utilizing the ResourceCollection type (see 8.6.9). Evidence adds a date and deviceId for when the evidence was collected.<br>This element is used to provide results from a scan of a system where software that does not have a SWID tag is discovered. This information is not provided by the software creator, but is instead created when a system is being scanned and the evidence for why software is believed to be installed on the device is provided in the Evidence element.<br>SWID Tags that include the Evidence Element are based on the tool used and these tags may be temporary (if the tag with evidence is provided back to the server and a tool identified tag is subsequently installed back on the device), or permanent (if the tag data is utilized to identify what the SWID tag should be in a more dynamic or client centric manner). | | |
| **attribute or Element** | **Type** | **Required/ default** | **Definition** |
| date | datetime | optional | Date and time the evidence was gathered. |
| deviceId | string | optional | Identifier for the device the evidence was gathered from. |
| attributes from ResourceCollection | ResourceCollection (see 8.6.9) | optional | Elements from the ResourceCollection (8.6.9) shall be used when defining Payload data. |

**EXAMPLE** In this example, the following additional namespaces have been added into the use of the XML and hashes for the files are identified using the namespace definitions below.

```
- xmlns:SHA256=~"http://www.w3.org/2001/04/xmlenc#sha256~"
- xmlns:MD5=~"http://www.w3.org/2001/04/xmldsig-more#md5~"
- xmlns:SHA1=~"http://www.w3.org/2000/09/xmldsig#sha1~"
<Evidence date=~"201308-18T02:34Z~" deviceId=~"pc123.corp.somecompany.com~" >
      <File name=~"foo.txt~"
      SHA256:hash=
~"9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08~"
      MD5:hash=~"098f6bcd4621d373cade4e832627b4f6~"
      SHA1:hash=~"a94a8fe5ccb19ba61c4c0873d391e987982fbbd3~" />
</Evidence>
```

### 8.5.4 Link

| Element name | Link | | |
|---|---|---|---|
| **Description** | A reference to any another item (can include details that are related to the SWID tag such as details on where software downloads can be found, vulnerability database associations, use rights, etc).<br>NOTE This is modelled directly to match the HTML [LINK] element; it is critical for streamlining software discovery scenarios that these are kept consistent. See Reference [3]. | | |
| **attribute or Element** | **Type** | **Required/ default** | **Definition** |

| | | | |
|---|---|---|---|
| artifact | string | optional | For installation media (rel="installationmedia"); dictates the canonical name for the target resource.<br><br>Items with the same artifact name shall be considered mirrors of each other. |
| href | anyURI | required | The link to the item being referenced<br><br>Notes:<br><br>The href can point to several different things and can be any of the following:<br><br>— a relative uri (no scheme) - which is interpreted depending on context (e.g. ./folder/supplemental.swidtag" )<br><br>— a physical file location with any system-acceptable URI scheme (e.g. file:// http:// https:// ftp:// ... etc<br><br>— a URI with "swid:..." as the scheme, which refers to another swid by tagId. This URI would need to be resolved in the context of the system by software that can lookup other swidtags.( e.g., "swid:2df9de35-0aff-4a86-ace6-f7dddd1ade4c" )<br><br>— a URI with "swidpath:..." as the scheme , which contains an XPATH query. This URI would need to be resolved in the context of the system by software that can lookup other swidtags and select the appropriate one based on an XPATH query. Examples:<br><br>— swidpath://SoftwareIdentity[Entity/@regid='http://contoso.com'] would retrieve all swidtags that had an entity where the regid was Contoso<br><br>— swidpath://SoftwareIdentity[Meta/@persistentId='b0c55172-38e9-4e36-be86-92206ad8eddb'] would retrieve swidtags that matched a specific persistentId.<br><br>See W3C Recommendation, XML Path Language (XPath) 2.0 (Second Edition).<br><br>DMTF Specification, *CIM operations over HTTP* (http://www.dmtf.org/sites/default/files/standards/documents/DSP200.html)<br><br>World Wide Web Consortium references<br><br>Only one href attribute is allowed per Link element; however, multiple Link elements can be included to reference multiple URI's. |

| | | | |
|---|---|---|---|
| media | string (Media) | optional | An expression that the document evaluator can use to determine if the target of the link is applicable to the current platform (the host environment).<br>Used as an optimization hint to notify a system that it can ignore something when it's not likely to be used.<br>**8.7 The format of this string is modelled upon the W3C Recommendation, Media Queries** [DMTF References DMTF Specification, *CIM operations over HTTP (*http://www.dmtf.org/sites/default/files/standards/documents/DSP200.html*)*<br>World Wide Web Consortium references][12]<br>This is one or more EXPRESSIONs where the items are connected with an OPERATOR:<br>media="EXPRESSION [[OPERATOR] [EXPRESSION]...]"<br>EXPRESSION is processed case-insensitive and defined either :<br>(ENVIRONMENT)<br>indicates the presence of the environment<br>or<br>([PREFIX-]ENVIRONMENT.ATTRIBUTE:VALUE)<br>indicates a comparison of an attribute of the environment.<br>ENVIRONMENT is a text identifier that specifies any software, hardware feature, or aspect of the system the software is intended to run in.<br>Common ENVIRONMENTs include (but not limited to) the following:<br>— linux;<br>— windows;<br>— java;<br>— powershell;<br>— ios;<br>— chipset;<br>— peripheral. |

| | | | |
|---|---|---|---|
| media | string (Media) | optional | ATTRIBUTE is a property of an ENVIRONMENT with a specific value.<br>Common attributes include (but not limited to) the following:<br>— version;<br>— vendor;<br>— architecture.<br>PREFIX is defined as one of the following:<br>— MIN # property has a minimum value of VALUE;<br>— MAX # property has a maximum value of VALUE.if a PREFIX is not provided, then the property shall equal VALUE.<br>OPERATOR is defined of one of the following:<br>— AND;<br>— NOT.<br>Examples:<br>media="(windows)"<br># applies to only systems that identify themselves as 'Windows'<br>media="(windows) not (windows.architecture:x64)"<br># applies to only systems that identify<br># themselves as windows and are not for an x64 cpu<br>media="(windows) and (min-windows.version:6.1)"<br># applies to systems that identify themselves as<br># windows and at least version 6.1<br>media="(linux) and (linux.vendor:redhat) and (min-linux.kernelversion:3.0)"<br># applies to systems that identify themselves as<br># linux, made by redhat and with a kernel version of at least 3.0<br>media="(freebsd) and (min-freebsd.kernelversion:6.6)"<br># applies to systems that identify themselves as<br># freebsd, with a kernel version of at least 6.6<br>media="(powershell) and (min-powershell.version:3.0)"<br># applies to systems that have powershell 3.0 or greater<br>Properties are expected to be able to be resolved by the host environment without having to do significant computation. |
| ownership | Ownership (see 8.6.4) | optional | Determines the relative strength of ownership of the target piece of software. |
| rel | NMTOKEN (see Rel 8.6.7) | required | The relationship between this SWID and the target file.<br>Relationships can be identified by referencing the IANA Link Relations registration library, see Reference [7]. |

| | | | |
|---|---|---|---|
| type | string (MediaType) | optional | The IANA MediaType for the target href; this provides the SWID tag consumer an indicator of the resource type being referenced.<br>See Reference [8] for more details on link type. |
| use | Use (see 8.7.7) | optional | Determines if the target software is a hard requirement. |

**EXAMPLE**

```
<Link
    rel=~"component~"
    href=~"swid:iso-sid-cmp-acme-endpoint-protection-manager-v12-1-mp1~"
    use=~"required~"
    ownership=~"private~"/>


Example of referenced installation items with artifact. Each reference with
the same artifact value is considered an alternative location to access the
installation files. The example below contains two different installation
options.
<!-- link to installation media. items with same 'artifact' are alternate
locations for the same thing-->
<Link rel=~"installationmedia~" media=~"(windows) and (windows.architecture:
x64)~" artifact=~"windows-installer-x64~" href=~"http://fabrikam.com/downloads/
webserver-2.2-x64.msi~" />
<Link rel=~"installationmedia~" media=~"(windows) and (windows.architecture:
x64)~" artifact=~"windows-installer-x64~" href=~"http://mirror.fabrikam.com/downloads/
webserver-2.2-x64.msi~" />
<Link rel=~"installationmedia~" media=~"(windows) and (windows.architecture:
x64)~" artifact=~"windows-installer-x64~" href=~"http://downloads.contoso.com/
downloads/webserver-2.2-x64.msi~" />


<!-- link to installation media. items with same 'artifact' are alternate
locations for the same thing-->
<Link rel=~"installationmedia~" media=~"(windows) and (windows.architecture:
x86)~" artifact=~"windows-installer~" href=~"http://fabrikam.com/downloads/webserver-2
.2.msi~" />
<Link rel=~"installationmedia~" media=~"(windows) and (windows.architecture:
x86)~" artifact=~"windows-installer~" href=~"http://mirror.fabrikam.com/downloads/
webserver-2.2.msi~" />
<Link rel=~"installationmedia~" media=~"(windows) and (windows.architecture:
x86)~" artifact=~"windows-installer~" href=~"http://downloads.contoso.com/downloads/
webserver-2.2.msi~" />
```

### 8.5.5 Meta

| **Element name** | Meta<br>(type SoftwareMeta see 8.6.11) | | |
|---|---|---|---|
| **Description** | An open-ended collection of key/value data related to this SWID.<br>Meta, when used in the SoftwareIdentity scope allows for the use of the attributes specified in the SoftwareMeta type and will allow for any additional attributes regardless if they are defined in this standard/XSD, or not.<br>This gives a flexibility to the tagCreator to include additional meta data as required.<br>NOTE The processContents declaration for additional Meta attributes is defined as lax. | | |
| **attribute or Element** | **Type** | **Required/ default** | **Definition** |
| <any> | per definition | optional | Additional attributes that have been defined in their own namespace. This allows the the Meta values that can be defined to be extended by the tag creator. |
| attributes from SoftwareMeta | SoftwareMeta | optional | Attributes from the SofwareMeta (8.6.11) are pre-defined for normalization purposes when this element is used within the scope of the SoftwareIdentity Element. |

### 8.5.6 Payload

| **Element name** | Payload<br>(type ResourceCollection) | | |
|---|---|---|---|
| **Description** | Only one entry of the Evidence element may be included inside the SoftwareIdentity Tag.<br>The items that may be installed on a device when the software is installed. Note that Payload may be a superset of the items installed and, depending on optimization systems for a device, may or may not include every item that could be created or executed on a device when software is installed.<br>In general, payload will be used to indicate the files that may be installed with a software product and will often be a superset of those files (i.e. if a particular optional component is not installed, the files associated with that component may be included in payload, but not installed on the device). | | |
| **attribute or Element** | **Type** | **Required/ default** | **Definition** |
| attributes from ResourceCollection | ResourceCollection<br>(see 8.6.9) | optional | Elements from the ResourceCollection (8.6.9) shall be used when defining Payload data. |

**EXAMPLE** In this example, the following additional namespaces have been added into the use of the XML and hashes for the files are identified using the namespace definitions below:

```
- xmlns:SHA256=~"http://www.w3.org/2001/04/xmlenc#sha256~"
- xmlns:MD5=~"http://www.w3.org/2001/04/xmldsig-more#md5~"
-xmlns:SHA1=~"http://www.w3.org/2000/09/xmldsig#sha1~"
<Payload>
   <File name=~"foo.txt~"
    size=10354
    SHA256:hash=
~"9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08~"
    MD5:hash=~"098f6bcd4621d373cade4e832627b4f6~"
    SHA1:hash=~"a94a8fe5ccb19ba61c4c0873d391e987982fbbd3~" />
</Payload>
```

## 8.6 Type and attribute definitions

### 8.6.1 Directory

| **Name** | Directory |
|---|---|
| **Data type** | Complex type<br>Extension of FileSystemItem (see 8.6.3) |
| **Definition** | Provides the ability to apply a directory structure to the files defined in a Payload or Evidence element. |

| Defined values | Value | Type | Meaning |
|---|---|---|---|
| | Directory | Directory (see 8.6.1) | Directory element allows one or more directories to be defined in the file structure. |
| | File | File (see 8.6.2) | File element that allows one or more files to be specified for a given location. |
| | attributes from FileSystemItem | FileSystemItem (see 8.6.3) | Additional Meta data about a file. |

**EXAMPLE** The following example are created for a Microsoft windows platform, however, the same approach can be used on any platform. In the case where a system variable exists for "known" directories, those variables can be used, in other instances, environment variables may be used.

In the following example if %programdata% references c:\programdata, both of the examples below will reference the following file:

c:\programdata\printerco\printermodel\colordriver.dll

```
<Payload>
 <Directory root=~"%programdata%~" name=~"printerco~">
  <Directory name=~"printermodel~">
   <File name=~"colordriver.dll~" size=~"234824~"
   SHA256:hash=
~"186a46b8d7645ec498d5efc02f53e3f5e1df1710796db4e2b071108f654ba02a~" />
  </Directory>
 </Directory>
</Payload>
-or-
<Payload>
<Directory root=~"%programdata%~">
 <Directory name=~"printerco~">
  <Directory name=~"printermodel~">
   <File name=~"colordriver.dll~" size=~"234824~"
    SHA256:hash=
~"186a46b8d7645ec498d5efc02f53e3f5e1df1710796db4e2b071108f654ba02a~" />
   </Directory>
  </Directory>
 </Directory>
</Payload>
```



### 8.6.2 File

| Name | File |
|---|---|
| Data type | Complex Type<br>Extension of FileSystemItem (see 8.6.3) |
| Definition | Provides file information that can be used for a variety of IT requirements including validating files, identifying if files exist, and associating files with an application. Can be used for Evidence and/or Payload data. |

| Defined values | Value | Type | Meaning |
|---|---|---|---|
| | name | string (required) | The filename without any path characters. |
| | size | integer | The file size in bytes. |
| | version | string | If available, the version of the file. |
| | attributes from FileSystemItem | FileSystemItem (see 8.6.3) | Additional Meta data about a file. |

EXAMPLE Files are also expected to include a hash value that provides different levels of confidence that if the filename, file size and file hash code all match, then the file has not been modified in any fashion. These hash values are provided by including additional attributes and using specific namespaces to define the algorithm type. The following example provides details on how additional attributes and different hash values look like when included in the SWID tag file:

```
<File name=~"foo.txt~"
    size=10354
    SHA256:hash=
~"9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08~"
    MD5:hash=~"098f6bcd4621d373cade4e832627b4f6~"
    SHA1:hash=~"a94a8fe5ccb19ba61c4c0873d391e987982fbbd3~" />
```

### 8.6.3 FileSystemItem

| Name | FileSystemItem |
|---|---|
| Data type | Complex Type |
| Definition | Provides specific attributes for Directory and File, as well as allowing for any values to be used to extend the attributes (such as a variety of hash values, even hash values that have not yet been defined). |

| Defined values | Value | Type | Meaning |
|---|---|---|---|
| | key | boolean | A true value indicates that the directory is considered important or required for the use of a software component. Typical key directories would be those which, if not available on a system, would cause the software not to execute.<br>Key directories will typically be used to validate that software referenced by the SWID tag is actually installed on a specific computing device. |
| | location | string | The directory or location where a file was found or can expected to be located. |
| | name | String | The name of the filename or directory without any path characters. |
| | root | string | A system-specific root folder that the 'location' attribute is an offset from. If this is not specified the assumption is the 'root' is the same folder as the location of the SWID tag, or is the directory specified by an enclosing "Directory" SWID tag element. |
| | <any> | <any> | Additional values are allowed so that SWID tag producers can extend the roles they allow.<br>NOTE <any> is based on the fact that Meta is the base for FileSystemItem. |

EXAMPLE The following examples are created for a Microsoft windows platform; however, the same approach can be used on any platform. In the case where a system variable exists for "known" directories, those variables can be used, in other instances, environment variables may be used.

In the following example if %programdata% references c:\programdata, both of the examples below will reference the following file:

```
c:\programdata\printerco\printermodel\colordriver.dll
```

```
<Payload>
 <Directory root="%programdata%" name="printerco">
  <Directory name="printermodel">
   <File name="colordriver.dll" size="234824"
   SHA256:hash="186a46b8d7645ec498d5efc02f53e3f5e1df1710796db4e2b071108f654ba02a" />
  </Directory>
 </Directory>
</Payload>
-or-
<Payload>
 <Directory root="%programdata%">
  <Directory name="printerco">
   <Directory name="printermodel">
    <File name="colordriver.dll" size="234824"
     SHA256:hash="186a46b8d7645ec498d5efc02f53e3f5e1df1710796db4e2b071108f654ba02a" />
   </Directory>
  </Directory>
 </Directory>
</Payload>
```

### 8.6.4 Ownership

| | | |
|---|---|---|
| **Name** | Ownership (enumerated values) | |
| **Data type** | NMTOKEN | |
| **Definition** | Specifies how an entity relates to this SWID tag. See 8.5.4 for an example of how this type is used. | |
| **Defined values** | **Enumerationvalues** | **Meaning** |
| | abandon | If the item referenced by this SWID tag is uninstalled, then leave the [Link]'d software installed. |
| | private | If the item referenced by this SWID tag is uninstalled, then the [Link]'d software should be removed too. |
| | shared | If the item referenced by this SWID tag is uninstalled, then the [Link]'d software should be removed if no other software item is sharing it. |

EXAMPLE

```
<Link
      rel=~"component~"
      href=~"swid:iso-sid-cmp-acme-endpoint-protection-manager-v12-1-mp1~"
      use=~"required~"
      ownership=~"private~"/>
```

### 8.6.5 NMTOKEN and NMTOKENS

NMTOKEN and NMTOKENS are XML types as defined in W3C Recommendation, XML Schema Part 2: Datatypes (Second Edition)

DMTF Specification, CIM operations over HTTP (http://www.dmtf.org/sites/default/files/standards/documents/DSP200.html) World Wide Web Consortium references.

NMTOKEN is a string value or "token" made up of a limited set of specified ASCII characters and NMTOKENS is a set of values or "tokens" separated by the ASCII space (#0x20) character. See the W3C Recommendation for further details.

### 8.6.6 Process

| | | | |
|---|---|---|---|
| **Name** | Process | | |
| **Data type** | Complex Type | | |
| **Definition** | Provides process information for data that will show up in a devices process table. | | |
| **Defined values** | **Value** | **Type** | **Meaning** |
| | name | string (required) | The process name as it will be found in the devices process table. |
| | pid | integer (optional) | The process ID for the executing process; note that this will typically only be provided when the Process element is included as part of Evidence. |

### 8.6.7 Rel

| | |
|---|---|
| **Name** | rel (enumerated values) |
| **Data type** | NMTOKEN |
| **Definition** | Specifies how the target of a [Link] relates to this SWID. |

| Defined values | Value | Meaning |
|---|---|---|
| | ancestor | Refers to a SWID tag defining an ancestor of this software; used for upgrades. |
| | component | A component of this product. |
| | feature | A feature is a way of describing part of a software product that can be enabled or disabled separately without necessarily modifying any physical files. |
| | installationmedia | A [Link] to the installation media used to install the software identified in this SWID tag. |
| | packageinstaller | A tool or individual is required to install a software package referenced in the [Link]. |
| **Defined values** | parent | A reference to the SWID tag that [Link] is the parent to this SWID tag. |
| | patches | A reference to the product(s) a patch is applied to. |
| | requires | An additional software installation is required for the title referred to in this SWID tag to function properly The additional software is may, or may not be managed by the same group or organization as the one creating this SWID tag (example: C/C++ runtimes, virtual machines, etc).<br>If the SoftwareIdentity.patch is set to true, using a rel with requires indicates that this patch requires another patch to be installed first. |
| | see-also | Other pieces of software that may relate to the software identified in this SWID tag. These can be add-ons or other extensions that may be of interest to the user/administrator of the device. |
| | supersedes | This rel value is used with SoftwareIdentity.patch is equal to true and specifies the linkage to other patches that this patch supersedes. |
| | supplemental | Additional information that represents *this* SWID. This may be used to provide additional information for a software product such as the purpose for the installation that can provide a hint to compliance managers. |
| | <any> | Additional relationships can be used and identified by referencing Reference [7]. |

### 8.6.8 Resource

| Name | Resource | | |
|---|---|---|---|
| **Data type** | Complex Type | | |
| **Definition** | A container that can be used to provide arbitrary resource information about an application installed on a device, or evidence collected from a device. | | |
| **Defined values** | **Value** | **Type** | **Meaning** |
| | type | string (required) | The type of the resource could be information such as registrykey, port, rootUrl, etc. |
| | <any> | <any> | Resource allows any attributes to be included in the SWID tag. |

### 8.6.9 ResourceCollection

| Name | ResourceCollection |
|---|---|
| **Data type** | Complex Type |
| **Definition** | This type is used by Payload to provide details on what may be installed on a device and by Evidence to indicate what an inventory process discovered on a device. |

| Defined values | Value | Type | Meaning |
|---|---|---|---|
| | Directory<br>(see 8.6.1) | Directory | Directory element allows one or more directories to be defined in the file structure. |
| | File<br>(see 8.6.2) | File | Files that are included as part of the installation. |
| | Process<br>(see 8.6.6) | Process | Processes that may be found on the device. |
| | Resource<br>(see 8.6.8) | Resource | Additional generic resources that may be included as part of the installation (/dev entries, registry settings, etc.). |

### 8.6.10 Role

| Attribute | role | |
|---|---|---|
| Data type | NMTOKENS | |
| Definition | Specifies how an entity relates to this SWID tag.<br>NOTE Pre-defined values are recommended for use, but it is expected that new roles may be required as market needs change. This type includes the ability to define additional values as required. | |
| Defined values | Value | Meaning |
| | aggregator | 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). |
| | distributor | 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. |
| | licensor | Software licensor (see ISO/IEC 19770-5) |
| | softwareCreator | Software creator (see ISO/IEC 19770-5) |
| | tagCreator | Tag creator (see ISO/IEC 19770-5) |
| | <any> | Additional values are allowed so that SWID tag producers can extend the roles they allow. |

### 8.6.11 SoftwareMeta

| Element name | SoftwareMeta | | |
|---|---|---|---|
| Description | SoftwareMeta is an open-ended collection of key/value data related to this SWID.<br>The attributes shown below are predefined attributes to ensure common usage across the industry. The schema allows for any additional attribute to be included in a SWID tag, though it is recommended that industry norms for new attributes are defined and followed to the degree possible.<br>NOTE The processContents declaration for additional SoftwareMeta attributes is defined as lax. | | |
| attribute or Element | Type | Required/ default | Definition |
| activationStatus | string | optional | Identification of the activation status of this software title (e.g. Trial, Serialized, Licensed, Unlicensed, etc.). Typically, this is used in supplemental tags. |
| channelType | string | optional | Provides information on which channel this software was targeted for (e.g. Volume, Retail, OEM, Academic, etc.). Typically this is used in supplemental tags. |

| | | | |
|---|---|---|---|
| colloquialVersion | string | optional | The informal or colloquial version of the product (i.e. 2013). Note that this version may be the same through multiple releases of a software product where the version specified in SoftwareEntity is much more specific and will change for each software release.<br><br>Note that this representation of version is typically used to identify a group of specific software releases that are part of the same release/support infrastructure (i.e. Fabrikam Office 2013). This version is used for string comparisons only and is not compared to be an earlier or later release (that is done via the SoftwareEntity version). |
| description | string | optional | A longer, detailed description of the software. This description can be multiple sentences (differentiated from "summary" which is a very short, one-sentence description). |
| edition | string | optional | The variation of the product (Extended, Enterprise, Professional, Standard, etc.). |
| entitlementDataRequired | boolean | optional | An indicator to determine if there should be accompanying proof of entitlement when a software license reconciliation is completed. |
| entitlementKey | string | optional | A vendor-specific textual key that can be used to reconcile the validity of an entitlement (e.g. serial number, product or license key). |
| generator | string | optional | The name of the software tool that created a SWID tag. This element is typically used if tags are created on the fly, or based on a catalogue based analysis for data found on a computing device. |
| persistentId | string | optional | A GUID used to represent products installed where the products are related, but may be different versions. |
| product | string | optional | The base name of the product (e.g. Office, Creative Suites, Websphere, etc). |
| productFamily | string | optional | The overall product family this software belongs to. Product family is not used to identify that a product is part of a suite, but is instead used when a set of products that are all related may be installed on multiple different devices.<br><br>For example, an Enterprise backup system may consist of a backup server, multiple different backup systems that support mail servers, databases and ERP systems as well as individual software items that backup client devices. In this case all software titles that are part of the backup system would have the same productFamily name so they can be grouped together in reporting systems. |

| | | | |
|---|---|---|---|
| revision | string | optional | The informal or colloquial representation of the sub-version of the given product (e.g. SP1, R2, RC1, Beta 2, etc.). Note that the SoftwareIdentity.version will provide very exact version details, the revision is intended for use in environments where reporting on the informal or colloquial representation of the software is important (for example – if for a certain business process, an organization recognizes that it must have ServicePack 1 or later of a specific product installed on all devices, they can use the revision data value to quickly identify any devices that do not meet this requirement).<br>Depending on how a software organizations distributes revisions, this value could be specified in a primary (if distributed as an upgrade) or supplemental (if distributed as a patch) SWID tag. |
| summary | string | optional | A short (one-sentence) description of the software. |
| unspscCode | string | optional | An 8 digit code that provides UNSPSC classification of the software product this SWID tag identifies. For more information, see Reference [5]. |
| unspscVersion | string | optional | The version of the UNSPSC code used to define the UNSPSC code value. For more information, see Reference [5]. |
| <any> | per definition | optional | Additional attributes that have been defined in their own namespace. This allows the Meta values that can be defined to be extended by the tag creator. |

NOTE One representation of a persistentId values is through the use of what, in a windows installation process is referred to as an upgradeCode - http://msdn.microsoft.com/en-us/library/aa372375(v=vs.85).aspx as one example of the use of this value.

```
<Meta
    product=~"Office~"
    edition=~"Professional~"
    colloquilVersion=~"2013~"
    revision=~"SP1~"/>
```

### 8.6.12 Use

| | | |
|---|---|---|
| **Name** | Use (enumerated values) | |
| **Data type** | NMTOKEN | |
| **Definition** | Provides additional hints on when software identified in a [Link] should be installed. | |
| **Defined values** | **Eunumerated values** | **Meaning** |
| | required | The [Link]'d software is absolutely required for an operation software installation. |
| | recommended | Not absolutely required; the [Link]'d software is installed unless specified otherwise. |
| | optional | Not absolutely required; the [Link]'d software is installed only when specified. |

### 8.6.13 VersionScheme

| | | |
|---|---|---|
| **Name** | VersionScheme (enumerated values) | |
| **Data type** | string | |
| **Definition** | VersionScheme provides the types of version schemas that are represented in the version values. | |
| **Defined values** | **Value** | **Meaning** |
| | multipartnumeric | Numbers separated by dots, where the numbers are interpreted as integers (e.g. 1.2.3, 1.4.5, 1.2.3.4.5.6.7). |
| | multipartnumeric+suffix | Numbers separated by dots, where the numbers are interpreted as integers with an additional string suffix (i.e. 1.2.3a). |
| | alphanumeric | Strictly a string, sorting is done alphanumerically. |
| | decimal | A floating point number (e.g. 1.25 is less than 1.3). |
| | semver | Follows the semver.org specification. |
| | unknown | Other unknown version scheme, no attempt should be made to order versions of this type. |
| | <any> | Other version schemes that may be generally known in the market. |

# Annex A
(informative)

# XSD changes between revisions

## A.1 Overview

The XSD definition for SWID tags was significantly changed between the first publication of this part of ISO/IEC 19770 in 2009 and this revision. There are a number of reasons for these changes, but they can be grouped into the following three areas:

— data management and efficiency;

— market use and feedback;

— market adoption.

Each of these areas will be summarized in the clauses below.

During the development of this version of this part of ISO/IEC 19770, multiple approaches to managing the transition to the new XSD were analysed. During this analysis, it became obvious that existing software that was released with SWID tags defined by the original 2009 version of the XSD continue to use the same tags without change due to the fact that there would be no need to change the SWID data for existing released software. Additionally, tools and processes that are used for software discovery procedures will be able to identify that an older XSD structure is in use and will be able to utilize the SWID data from the older SWID tags. Since SWID tags are not typically modified until software is updated or patched, it will be important for tools that discover SWID tags to recognize that there may be different versions of the SWID data structures that could be in use at any time. For example, a software product released with a SWID tag based on the 2009 version of this part of ISO/IEC 19770 will continue using the 2009 version of the SWID tag structure and may also include a later patch release that utilizes the revised version of the standard.

Software products released with SWID tags using this revision of the SWID Tag schema shall not include duplicate tags that follow an earlier revision of this standard.

Additionally, tools that create SWID tags will very easily be able to adopt the new, more efficient XSD data structure and will have expanded abilities to add value to the SWID tags their customer's utilize. These tools will not be expected to create multiple different SWID tag structures rather, they only need to create one version of a SWID tag and they can then upgrade the version of the XSD they use as market conditions indicate that the tool needs to create a newer version of the SWID tag.



## A.2 Data management and efficiency

The 2009 version of the SWID XSD was not optimized for efficiency of managing or interpreting SWID tag data, nor was it designed to provide the smallest memory footprint. These issues may not be critical problems for smaller software creators or tool providers. However, they do have a major impact on tools and software creators focused on the enterprise market and those organizations focused on providing support for the Internet of Things (IoT).

This revision addresses these issues by consolidating data values that were previously separate elements in the 2009 version into XML attributes, as well as removing information that would be common to all SWID tags created by the software creator, further reducing the size of the SWID tag.

The use of the new element and attribute structure also allows for much more efficient processing of the SWID tag data when used for IT processes.

## A.3 Market use and feedback

Numerous tag creation tools and software creators have included SWID tags in their tools and processes. In most cases, the process is either automated or automation is desired. This means that implementing a new or upgraded tool is quite easy with very limited impact on processes or testing. The general feedback has been that as long as the inputs for the existing tool will work as inputs to any new tool, the overall change should be very low cost and easy to manage.

Additionally, there are third party tools that can be used to generate SWID tags for software currently installed on a device, as well as software recognition libraries that may not be able to provide all the minimum values specified in the older version of the XSD. The organizations managing these products and libraries have indicated a desire to create conforming SWID tags even if, for example, they do not have every piece of data to specify all mandatory elements specified in the 2009 version of the XSD.

Finally, there has been very strong support from software creators to include SWID tags to help their customers properly identify the exact software that is installed on a device. When utilizing the previous XSD structure for testing and modelling exercises, it became clear that there were a number of issues that needed to be addressed, three of which are identified below (and which are properly managed in this version of this part of ISO/IEC 19770).

— To support SWID tags effectively and efficiently, software creators should include the process to create (and optionally digitally sign) SWID tags as an automated sub-process during the software build process. One impact of this requirement that certain data values required by the 2009 version of the XSD may not be available to be provided automatically at the point in the software build process that it needs to be executed (for example, if an entitlement is required or not).

— Some software relationships deployed by software creators require more flexibility to identify the dependencies. These relationships may require logical operators such as a requirement that the software uses version X.Y or newer of a specified product (e.g. version ≥X.Y).

— A tremendous number of versioning strategies in use for software available today. There are some software products that do not include a version. This issue required a different approach to managing version details because the 2009 version of the XSD was limited to using one to four integer values for a version number.

These and other issues that are directly related to market feedback from organizations who have implemented the 2009 version of this part of ISO/IEC 19770, along with testing and data modelling efforts that have been undertaken by organizations interested in supporting SWID tags, are the reasons for making the changes to the XSD structure.

## A.4 Market adoption

Software creators and discovery tool providers have a strong interest in being able to demonstrate clear business cases for the provision of SWID tags by software creators. The more organizations that support SWID tags, the more value those tags provide to the whole market. There are a number of areas where SWID tags can provide additional value beyond asset management, resulting in interest in other market segments in adopting SWID tags.

A good example of this additional value is the ability for SWID tags to specify additional software that is related to software titles currently installed on a device. This could include details such as plug-ins that are available for an image editing or browser software product. For open source titles, the details could be the various options that may be available for a desktop management software product.

Providing additional reasons for software creators to include SWID tags provides an incentive to adopt this part of ISO/IEC 19770 more quickly and the value to the market increases dramatically.

## A.5 Conclusion

There are quite a number of benefits to making significant XSD changes in this revision of this part of ISO/IEC 19770 and very few issues that arise because of the changed data structure; only a few of the positive benefits are identified above. The benefits far outweigh the problems of making these changes.

# Annex B
(normative)

# XML schema definition (XSD)

## B.1 General

The following XSD provides the definition for the software identification tag. This XSD file should be available at the following location:

http://standards.iso.org/iso/19770/-2/2015/schema.xsd

In anticipation of possible defects which will be identified after publication, an alternative download location always with the latest defect-corrected version is:

http://standards.iso.org/iso/19770/-2/2015-current/schema.xsd

In the event of any identified or perceived inconsistencies between the XML schema documented in B.2 and the text in the body of the document, the text in B.2 shall take precedence (due to its greater precision). However, the file available at the above URL will be updated more frequently than this document and so should be consulted for the most up-to-date definitions.

## B.2 XML schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xmlns:swid="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"

  targetNamespace="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  elementFormDefault="qualified">

  <xs:import
    namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
  <xs:import
    namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:annotation>
    <xs:documentation>
      Schema for ISO-IEC 19770-2 Software Identification Tags
      http://standards.iso.org/iso/19770/-2/2015/schema.xsd

      Copyright 2015 ISO/IEC, all rights reserved

      Copyright notice: ISO and IEC grant the users of this Standard the right
      to use this XSD file free of charge for the purpose of implementing the
      present Standard.

      Disclaimer: In no event shall ISO and/or IEC be liable for any damages
      whatsoever (including, but not limited to, damages for loss of profits,
      business interruption, loss of information, or any other pecuniary
      loss) arising out of or related to the use of or inability to use the
      XSD file. ISO and IEC disclaim all warranties, express or implied,
      including but not limited to warranties of merchantability and fitness
      for a particular purpose.
    </xs:documentation>
    <xs:appinfo>
```

```
      <version>2.0</version>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="SoftwareIdentity" type="SoftwareIdentity">
    <xs:annotation>
      <xs:documentation>
        Represents the root element specifying data about a software component
      </xs:documentation>
    </xs:annotation>
  </xs:element>

  <xs:complexType name="BaseElement">
    <xs:annotation>
      <xs:documentation>
        Attributes common to all Elements in this schema
      </xs:documentation>
    </xs:annotation>

    <xs:attribute
      ref="xml:lang" >
      <xs:annotation>
        <xs:documentation>
          Allow xml:lang attribute on any element.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute
      ref="xml:id" >
      <xs:annotation>
        <xs:documentation>
          Allow xml:id attribute on any element.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:anyAttribute
      namespace="##other"
      processContents="lax">
      <xs:annotation>
        <xs:documentation>
          Allows any undeclared attributes on any element as long as the
          attribute is placed in a different namespace.
        </xs:documentation>
      </xs:annotation>
    </xs:anyAttribute>

  </xs:complexType>

  <xs:complexType name="Entity">
    <xs:complexContent>
      <xs:extension base="BaseElement">
        <xs:annotation>
          <xs:documentation>
            Specifies the organizations related to the software component
            referenced by this SWID tag.
          </xs:documentation>
        </xs:annotation>

        <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:element
            name="Meta"
            type="Meta">
            <xs:annotation>
              <xs:documentation>
                An open-ended collection of elements that can be used to attach
                arbitrary metadata to an Entity.
              </xs:documentation>
            </xs:annotation>
          </xs:element>
```

```
      </xs:sequence>

      <xs:attribute
        name="name"
        use="required"
        type="xs:string">
        <xs:annotation>
          <xs:documentation>
            The name of the organization claiming a particular role in the
            SWID tag.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="regid"
        type="xs:anyURI"
        use="optional"
        default="http://invalid.unavailable">
        <xs:annotation>
          <xs:documentation>
            The regid of the organization.  If the regid is unknown, the
            value "invalid.unavailable" is provided by default (see
            RFC 6761 for more details on the default value).
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="role"
        type="xs:NMTOKENS"
        use="required">
        <xs:annotation>
          <xs:documentation>
            The relationship between this organization and this tag e.g. tag,
            softwareCreator, licensor, tagCreator, etc.  The role of
            tagCreator is required for every SWID tag.

            EntityRole may include any role value, but the pre-defined roles
            include: aggregator, distributor, licensor, softwareCreator,
            tagCreator

            Other roles will be defined as the market uses the SWID tags.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="thumbprint"
        use="optional"
        type="xs:string">
        <xs:annotation>
          <xs:documentation>
            this value provides a hexadecimal string that contains a hash
            (or thumbprint) of the signing entities certificate.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Evidence">
  <xs:complexContent>
    <xs:extension base="ResourceCollection">
      <xs:annotation>
        <xs:documentation>
          The element is used to provide results from a scan of a system
          where software that does not have a SWID tag is discovered.
          This information is not provided by the software creator, and
```

```
            is instead created when a system is being scanned and the
            evidence for why software is believed to be installed on the
            device is provided in the Evidence element.
          </xs:documentation>
        </xs:annotation>
        <xs:attribute
          name="date"
          type="xs:dateTime">
          <xs:annotation>
            <xs:documentation>
              Date and time the evidence was gathered.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="deviceId"
          type="xs:string">
          <xs:annotation>
            <xs:documentation>
              Identifier for the device the evidence was gathered from.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="FilesystemItem">
    <xs:complexContent>
      <xs:extension base="Meta">
        <xs:annotation>
          <xs:documentation>
            Represents an individual file
          </xs:documentation>
        </xs:annotation>

        <xs:attribute
         name="key"
         type="xs:boolean">
          <xs:annotation>
            <xs:documentation>
              Files that are considered important or required for the use of
              a software component.  Typical key files would be those which,
              if not available on a system, would cause the software not to
              execute.

              Key files will typically be used to validate that software
              referenced by the SWID tag is actually installed on a specific
              computing device
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="location"
          type="xs:string">
          <xs:annotation>
            <xs:documentation>
              The directory or location where a file was found or can expected
              to be located.  does not include the filename itself.  This can
              be relative path from the 'root' attribute.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
        name="name"
        type="xs:string"
        use="required">
```

```
        <xs:annotation>
          <xs:documentation>
            The filename without any path characters
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="root"
        type="xs:string">
        <xs:annotation>
          <xs:documentation>
            A system-specific root folder that the 'location'
            attribute is an offset from. If this is not specified
            the assumption is the 'root' is the same folder as
            the location of the SWIDTAG.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:anyAttribute
        processContents="lax">
        <xs:annotation>
          <xs:documentation>
            Permits any user-defined attributes in file tags.
          </xs:documentation>
        </xs:annotation>
      </xs:anyAttribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Directory">
  <xs:complexContent>
    <xs:extension base="FilesystemItem">
      <xs:annotation>
        <xs:documentation>
          Provides the ability to apply a directory structure to the files
          defined in a Payload or Evidence element.
        </xs:documentation>
      </xs:annotation>

      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element
          name="Directory"
          type="Directory">
          <xs:annotation>
            <xs:documentation>
              A Directory element allows one or more directories to be
              defined in the file structure.
            </xs:documentation>
          </xs:annotation>
        </xs:element>

        <xs:element
          name="File"
          type="File">
          <xs:annotation>
            <xs:documentation>
              A File element that allows one or more files to be specified
              for a given location.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:choice>

    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="File">
```

```
    <xs:complexContent>
      <xs:extension base="FilesystemItem">
        <xs:annotation>
          <xs:documentation>
            Represents an individual file
          </xs:documentation>
        </xs:annotation>

        <xs:attribute
          name="size"
          type="xs:integer">
          <xs:annotation>
            <xs:documentation>
              The file size in bytes of the file
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="version"
          type="xs:string">
          <xs:annotation>
            <xs:documentation>
              The file version
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="Process">
    <xs:complexContent>
      <xs:extension base="Meta">
        <xs:annotation>
          <xs:documentation>
            Provides process information for data that will show up in a
            devices process table.
          </xs:documentation>
        </xs:annotation>

        <xs:attribute
          name="name"
          type="xs:string"
          use="required">
          <xs:annotation>
            <xs:documentation>
              The process name as it will be found in the devices process
              table.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="pid"
          type="xs:integer">
          <xs:annotation>
            <xs:documentation>
              The process ID for the executing process - note that this will
              typically only be provided when the Process element is included as part
              of Evidence.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="Resource">
```

```
    <xs:complexContent>
      <xs:extension base="Meta">
        <xs:annotation>
          <xs:documentation>
            A container that can be used to provide arbitrary resource
            information about an application installed on a device, or
            evidence collected from a device.
          </xs:documentation>
        </xs:annotation>

        <xs:attribute
          name="type"
          type="xs:string"
          use="required">
          <xs:annotation>
            <xs:documentation>
              The type of resource (ie, registrykey, port, rootUrl)
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="ResourceCollection">
    <xs:complexContent>
      <xs:extension base="BaseElement">
        <xs:annotation>
          <xs:documentation>
            This type is used by Payload to provide details on what may rbe
            installed on a device, and by Evidence to indicate what an
            inventory process discovered on a device.
          </xs:documentation>
        </xs:annotation>

        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element
            name="Directory"
            type="Directory">
            <xs:annotation>
              <xs:documentation>
                One or more directory elements
              </xs:documentation>
            </xs:annotation>
          </xs:element>

          <xs:element
            name="File"
            type="File">
            <xs:annotation>
              <xs:documentation>
                One or more File elements
              </xs:documentation>
            </xs:annotation>
          </xs:element>

          <xs:element
            name="Process"
            type="Process">
            <xs:annotation>
              <xs:documentation>
                One or more Process elements
              </xs:documentation>
            </xs:annotation>
          </xs:element>

          <xs:element
            name="Resource"
            type="Resource">
            <xs:annotation>
```

```
            <xs:documentation>
              One or more generic resource elements
            </xs:documentation>
          </xs:annotation>
        </xs:element>

      </xs:choice>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Link">
  <xs:complexContent>
    <xs:extension base="BaseElement">
      <xs:annotation>
        <xs:documentation>
          A reference to any another item (can include details that are
          related to the SWID tag such as details on where software
          downloads can be found, vulnerability database associations,
          use rights, etc).

          This is modeled directly to match the HTML [LINK] element; it is
          critical for streamlining software discovery scenarios that
          these are kept consistent.
        </xs:documentation>
      </xs:annotation>

      <xs:attribute
        name="artifact"
        type="xs:string"
        use="optional">
        <xs:annotation>
          <xs:documentation>
            For installation media (rel="installationmedia") - dictates the
            canonical name for the file.

            Items with the same artifact name should be considered mirrors
            of each other (so download from wherever works).
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="href"
        type="xs:anyURI"
        use="required" >
        <xs:annotation>
          <xs:documentation>
            The link to the item being referenced.

            The href can point to several different things, and can be any
            of the following:

            - a RELATIVE URI (no scheme) - which is interpreted depending on
            context (ie, "./folder/supplemental.swidtag" )

            - a physical file location with any system-acceptable
            URI scheme (ie, file:// http:// https:// ftp:// ... etc )

            - an URI with "swid:" as the scheme, which refers to another
            swid by tagId. This URI would need to be resolved in the
            context of the system by software that can lookup other
            swidtags.( ie, "swid:2df9de35-0aff-4a86-ace6-f7dddd1ade4c" )

            - an URI with "swidpath:" as the scheme, which refers to another
            swid by an XPATH query.  This URI would need to be resolved in
            the context of the system by software that can lookup other
            swidtags, and select the appropriate one based on an XPATH
            query. Examples:

            swidpath://SoftwareIdentity[Entity/@regid='http://contoso.com']
```

```
              would retrieve all swidtags that had an entity where the
              regid was Contoso

              swidpath://SoftwareIdentity[Meta/@persistentId='b0c55172-38e9-4e36-be86
-92206ad8eddb']
              would retrieve swidtags that matched the persistentId

              See XPATH query standard : http://www.w3.org/TR/xpath20/
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="media"
          type="Media"
          use="optional">
          <xs:annotation>
            <xs:documentation>
              An attribute defined by the W3C Media Queries Recommendation
              (see http://www.w3.org/TR/css3-mediaqueries/).

              A hint to the consumer of the link to what the target item is
              applicable for.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="ownership"
          type="Ownership"
          use="optional">
          <xs:annotation>
            <xs:documentation>
              Determines the relative strength of ownership of the target
              piece of software.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="rel"
          type="xs:NMTOKEN"
          use="required">
          <xs:annotation>
            <xs:documentation>
              The relationship between this SWID and the target file.

              Relationships can be identified by referencing the IANA
              registration library -
              https://www.iana.org/assignments/link-relations/link-relations.xhtml.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="type"
          type="MediaType"
          use="optional">
          <xs:annotation>
            <xs:documentation>
              The IANA MediaType for the target file; this provides the
              consumer with intelligence of what to expect.

              See http://www.iana.org/assignments/media-types/media-types.xhtml
              for more details on link type.
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

        <xs:attribute
          name="use"
```

```
          type="Use"
          use="optional">
          <xs:annotation>
            <xs:documentation>
              Determines if the target software is a hard requirement or not
            </xs:documentation>
          </xs:annotation>
        </xs:attribute>

      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="Meta">
    <xs:complexContent>
      <xs:extension base="BaseElement">
        <xs:annotation>
          <xs:documentation>
            An open-ended collection of key/value data related to this SWID.
          </xs:documentation>
        </xs:annotation>

        <xs:anyAttribute
          processContents="lax">
          <xs:annotation>
            <xs:documentation>
              Permits any user-defined attributes in Meta tags.
            </xs:documentation>
          </xs:annotation>
        </xs:anyAttribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="SoftwareIdentity">
    <xs:complexContent>
      <xs:extension base="BaseElement">
        <xs:choice maxOccurs="unbounded">

          <xs:element
            name="Entity"
            type="Entity"
            minOccurs="1"
            maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>
                Specifies the organizations related to the software component
                referenced by this SWID tag.

                This has a minOccurs of 1 because the spec declares that
                you must have at least a Entity with role='tagCreator'
              </xs:documentation>
            </xs:annotation>
          </xs:element>

          <xs:element
            name="Evidence"
            type="Evidence"
            minOccurs="0"
            maxOccurs="1">
            <xs:annotation>
              <xs:documentation>
                This element is used to provide results from a scan of a
                system where software that does not have a SWID tag is
                discovered.  This information is not provided by the
                software creator, but is instead created when a system
                is being scanned and the evidence for why software is
                believed to be installed on the device is provided in the
                Evidence element.
              </xs:documentation>
            </xs:annotation>
```

```
</xs:element>

<xs:element
  name="Link"
  type="Link"
  minOccurs="0"
  maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      A reference to any another item (can include details that
      are related to the SWID tag such as details on where software
      downloads can be found, vulnerability database associations,
      use rights, etc).

      Note:  This is modelled directly to match the HTML [LINK]
      element; it is critical for streamlining software discovery
      scenarios that these are kept consistent.
    </xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element
  name="Meta"
  type="SoftwareMeta"
  minOccurs="0"
  maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      An open-ended collection of key/value data related to this SWID.
    </xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element
  name="Payload"
  type="ResourceCollection"
  minOccurs="0"
  maxOccurs="1">
  <xs:annotation>
    <xs:documentation>
      The items that may be installed on a device when the software is
      installed.  Note that Payload may be a superset of the items
      installed and, depending on optimization systems for a device,
      may or may not include every item that could be created or
      executed on a device when software is installed.

      In general, payload will be used to indicate the files that
      may be installed with a software product and will often be a
      superset of those files (i.e. if a particular optional
      component is not installed, the files associated with that
      component may be included in payload, but not installed on
      the device).
    </xs:documentation>
  </xs:annotation>
</xs:element>

<xs:any
  namespace="##other"
  processContents="lax"
  minOccurs="0"
  maxOccurs="unbounded" >
  <xs:annotation>
    <xs:documentation>
      Allows any undeclared elements in the SoftwareIdentity element
      as long as the element is placed in a different namespace.

      As xs:any supercedes an xs:element declaration, this continues
      to support digital signatures using the ds:Signature element:

        Signatures are not a mandatory part of the software
        identification tag standard, and can be used as required
```

```
            by any tag producer to ensure that sections of a tag are not
            modified and/or to provide authentication of the signer. If
            signatures are included in the software identification tag,
            they shall follow the W3C recommendation defining the XML
            signature syntax which provides message integrity
            authentication as well as signer authentication services for
            data of any type.
        </xs:documentation>
      </xs:annotation>
    </xs:any>

</xs:choice>

<xs:attribute
  name="corpus"
  type="xs:boolean"
  use="optional"
  default="false">
  <xs:annotation>
    <xs:documentation>
      Set to true, if this attribute specifies that this SWID tag is a
      collection of information that describes the pre-installation
      data of software component.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
  name="patch"
  type="xs:boolean"
  use="optional"
  default="false">
  <xs:annotation>
    <xs:documentation>
      Set to true if this SWID describes a product patch or
      modification to a different software element.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
  name="media"
  type="Media"
  use="optional" >
  <xs:annotation>
    <xs:documentation>
      media is a hint to the tag consumer to understand what this
      SWID tag applies to (see the [Link] tags media attribute).
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
  name="name"
  type="xs:string"
  use="required">
  <xs:annotation>
    <xs:documentation>
      This attribute provides the software component name as it would
      typically be referenced.  For example, what would be seen in the
      add/remove dialog on a Windows device, or what is specified as
      the name of a packaged software product or a patch identifier
      name on a Linux device.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
  name="supplemental"
  type="xs:boolean"
  use="optional"
```

```
    default="false">
    <xs:annotation>
      <xs:documentation>
        Specifies that this tag provides supplemental tag data that can
        be merged with primary tag data to create a complete record of
        the software information.

        Supplemental tags will often be provided at install time and may
        be provided by different entities (such as the tag consumer, or
        a Value Added Reseller).
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <xs:attribute
    name="tagId"
    type="xs:string"
    use="required" >
    <xs:annotation>
      <xs:documentation>
        tagId shall be a globally unique identifier and should be
        assigned a GUID reference (see ISO/IEC 19770-5 definition
        for GUID).

        The tagID provides a unique reference for the specific product,
        version, edition, revision, etc (essentially, the same binary
        distribution).  If two tagIDs match and the tagCreator is the
        same, the underlying products they represent are expected to be
        exactly the same.

        This allows IT systems to identify if a software item (for
        example, a patch) is installed simply by referencing the
        specific tagID value which is likely to be readily available
        in a software inventory.

        It is recommended, when possible, that a 16 byte GUID
        be used for this field as this provides global uniqueness without
        a significant amount of overhead for space.

        If use of a 16 byte GUID is not possible, a text based globally
        unique ID may be constructed, this ID should include a unique
        naming authority for the tagCreator and sufficient additional
        details that the tagId is unique for the software product,
        version, edition, revision, etc.  This would likely look as
        follows (+ is used as a string concatenation symbol):

        regid + productName + version + edition + revision + ...
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <xs:attribute
    name="tagVersion"
    type="xs:integer"
    use="optional"
    default="0">
    <xs:annotation>
      <xs:documentation>
        The tagVersion indicates if a specific release of a software
        product has more than one tag that can represent that specific
        release.  This may be the case if a software tag producer creates
        and releases an incorrect tag that they subsequently want to fix,
        but with no underlying changes to the product the SWID tag
        represents.  This could happen if, for example, a patch is
        distributed that has a Link reference that does not cover all the
        various software releases it can patch.  A newer SWID tag for that
        patch can be generated and the tagVersion value incremented to
        indicate that the data is updated.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>
```

```
      <xs:attribute
        name="version"
        type="xs:string"
        use="optional"
        default="0.0" >
        <xs:annotation>
          <xs:documentation>
            Underlying development version for the software component.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="versionScheme"
        type="VersionScheme"
        default="multipartnumeric">
        <xs:annotation>
          <xs:documentation>
            Scheme used for the version number.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SoftwareMeta">
  <xs:complexContent>
    <xs:extension base="Meta">
      <xs:annotation>
        <xs:documentation>
          An open-ended collection of key/value data related to this SWID.

          The attributes included in this Element are predefined attributes
          to ensure common usage across the industry.  The schema allows for
          any additional attribute to be included in a SWID tag, though it is
          recommended that industry norms for new attributes are defined and
          followed to the degree possible.
        </xs:documentation>
      </xs:annotation>

      <xs:attribute
        name="activationStatus"
        type="xs:string"
        use="optional">
        <xs:annotation>
          <xs:documentation>
            Identification of the activation status of this software title
            (e.g. Trial, Serialized, Licensed, Unlicensed, etc).  Typically,
            this is used in supplemental tags.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="channelType"
        type="xs:string"
        use="optional">
        <xs:annotation>
          <xs:documentation>
            Provides information on which channel this particular
            software was targeted for (e.g. Volume, Retail, OEM,
            Academic, etc).  Typically used in supplemental tags.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="colloquialVersion"
```

```
    type="xs:string"
    use="optional">
    <xs:annotation>
      <xs:documentation>
        The informal or colloquial version of the product (i.e. 2013).
        Note that this version may be the same through multiple releases
        of a software product where the version specified in
        SoftwareEntity is much more specific and will change for each
        software release.

        Note that this representation of version is typically used to
        identify a group of specific software releases that are part of
        the same release/support infrastructure
        (i.e. Fabrikam Office 2013).  This version is used for string
        comparisons only and is not compared to be an earlier or later
        release (that is done via the SoftwareEntity version).
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <xs:attribute
    name="description"
    type="xs:string"
    use="optional">
    <xs:annotation>
      <xs:documentation>
        A longer, detailed description of the software.  This description
        can be multiple sentences (differentiated from summary which is
        a very short, one-sentence description).
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <xs:attribute
    name="edition"
    type="xs:string"
    use="optional">
    <xs:annotation>
      <xs:documentation>
        The variation of the product (Extended, Enterprise, Professional,
        Standard etc)
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <xs:attribute
    name="entitlementDataRequired"
    type="xs:boolean"
    use="optional">
    <xs:annotation>
      <xs:documentation>
        An indicator to determine if there should be accompanying proof
        of entitlement when a software license reconciliation is
        completed.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <xs:attribute
    name="entitlementKey"
    type="xs:string"
    use="optional">
    <xs:annotation>
      <xs:documentation>
        A vendor-specific textual key that can be used to reconcile the
        validity of an entitlement. (e.g. serial number, product or
        license key).
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>
```

```
<xs:attribute
  name="generator"
  type="xs:string"
  use="optional">
  <xs:annotation>
    <xs:documentation>
      The name of the software tool that created a SWID tag.  This
      element is typically used if tags are created on the fly, or
      based on a catalogue based analysis for data found on a
      computing device.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
      name="persistentId"
      type="xs:string"
      use="optional">
  <xs:annotation>
    <xs:documentation>
      A GUID used to represent products installed where the products
      are related, but may be different versions.  See one
      representation of this value through the use of what, in a
      windows installation process is referred to as an upgradeCode
      - http://msdn.microsoft.com/en-us/library/aa372375(v=vs.85).aspx
      as one example of the use of this value.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
  name="product"
  type="xs:string"
  use="optional">
  <xs:annotation>
    <xs:documentation>
      The base name of the product (e.g. Office, Creative Suites,
      Websphere, etc).
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
  name="productFamily"
  type="xs:string"
  use="optional">
  <xs:annotation>
    <xs:documentation>
      The overall product family this software belongs to.  Product
      family is not used to identify that a product is part of a
      suite, but is instead used when a set of products that are all
      related may be installed on multiple different devices.

      For example, an Enterprise backup system may consist of a backup
      server, multiple different backup systems that support mail
      servers, databases and ERP systems as well as individual software
      items that backup client devices.  In this case all software
      titles that are part of the backup system would have the same
      productFamily name so they can be grouped together in reporting
      systems.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>

<xs:attribute
  name="revision"
  type="xs:string"
  use="optional">
  <xs:annotation>
    <xs:documentation>
      The informal or colloquial representation of the sub-version of
```

```
            the given product (ie, SP1, R2, RC1, Beta 2, etc).  Note that the
            SoftwareIdentity.version will provide very exact version details,
            the revision is intended for use in environments where reporting
            on the informal or colloquial representation of the software is
            important (for example, if for a certain business process, an
            organization recognizes that it must have ServicePack 1 or later
            of a specific product installed on all devices, they can use the
            revision data value to quickly identify any devices that do not
            meet this requirement).

            Depending on how a software organizations distributes revisions,
            this value could be specified in a primary (if distributed as an
            upgrade) or supplemental (if distributed as a patch) SWID tag.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="summary"
        type="xs:string"
        use="optional">
        <xs:annotation>
          <xs:documentation>
            A short (one-sentence) description of the software.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="unspscCode"
        type="xs:string"
        use="optional">
        <xs:annotation>
          <xs:documentation>
            An 8 digit code that provides UNSPSC classification of the
            software product this SWID tag identifies.  For more
            information see, http://www.unspsc.org/
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute
        name="unspscVersion"
        type="xs:string"
        use="optional">
        <xs:annotation>
          <xs:documentation>
            The version of the UNSPSC code used to define the UNSPSC code
            value. For more information see, http://www.unspsc.org/.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="Media">
  <xs:annotation>
    <xs:documentation>
      An expression that the document evaluator can use to determine if the
      target of the link is applicable to the current platform (the host
      environment)

      Used as an optimization hint to notify a system that it can
      ignore something when it's not likely to be used.

      The format of this string is modeled upon the MediaQuery definition at
      http://www.w3.org/TR/css3-mediaqueries/

      This is one or more EXPRESSIONs where the items are connected
      with an OPERATOR:
```

```
      media="EXPRESSION [[OPERATOR] [EXPRESSION]...]"

      EXPRESSION is processed case-insensitive and defined either :
      (ENVIRONMENT)
      indicates the presence of the environment
      or
      ([PREFIX-]ENVIRONMENT.ATTRIBUTE:VALUE)
      indicates a comparison of an attribute of the environment.

      ENVIRONMENT is a text identifier that specifies any software,hardware
      feature or aspect of the system the software is intended to run in.

      Common ENVIRONMENTs include (but not limited to):
      linux
      windows
      java
      powershell
      ios
      chipset
      peripheral

      ATTRIBUTE is a property of an ENVIRONMENT with a specific value.
      Common attributes include (but not limited to):
      version
      vendor
      architecture

      PREFIX is defined as one of:
      MIN    # property has a minimum value of VALUE
      MAX    # property has a maximum value of VALUE

      if a PREFIX is not provided, then the property should equal VALUE

      OPERATOR is defined of one of:
      AND
      NOT

      Examples:
      media="(windows)"
      # applies to only systems that identify themselves as 'Windows'

      media="(windows) not (windows.architecture:x64)"
      # applies to only systems that identify
      # themselves as windows and are not for an x64 cpu

      media="(windows) and (min-windows.version:6.1)"
      # applies to systems that identify themselves as
      # windows and at least version 6.1

      media="(linux) and (linux.vendor:redhat) and (min-linux.kernelversion:3.0)"
      # applies to systems that identify themselves as
      # linux, made by redhat and with a kernel version of at least 3.0

      media="(freebsd) and (min-freebsd.kernelversion:6.6)"
      # applies to systems that identify themselves as
      # freebsd, with a kernel version of at least 6.6

      media="(powershell) and (min-powershell.version:3.0)"
      # applies to systems that have powershell 3.0 or greater

      Properties are expected to be able to be resolved by the host
      environment without having to do significant computation.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>

<xs:simpleType name="MediaType">
  <xs:annotation>
    <xs:documentation>
```

```
        The IANA MediaType for the target href; this provides the SWID tag
        consumer with intelligence of what to expect.

        See http://www.iana.org/assignments/media-types/media-types.xhtml
        for more details on link type.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string”/>
</xs:simpleType>

<xs:simpleType name="Ownership">
  <xs:restriction base="xs:NMTOKEN">

    <xs:enumeration value="abandon">
      <xs:annotation>
        <xs:documentation>
          Determines the relative strength of ownership of the target
          piece of software.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>

    <xs:enumeration
      value="private">
      <xs:annotation>
        <xs:documentation>
          If this is uninstalled, then the [Link]'d software should be removed
          too.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>

    <xs:enumeration
      value="shared">
      <xs:annotation>
        <xs:documentation>
          If this is uninstalled, then the [Link]'d software should be removed
          if nobody else is sharing it
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>

  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Use">
  <xs:annotation>
    <xs:documentation>
      Determines if the target software is a hard requirement.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:NMTOKEN">

    <xs:enumeration
      value="required">
      <xs:annotation>
        <xs:documentation>
          The [Link]'d software is absolutely required for installation
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>

    <xs:enumeration
      value="recommended">
      <xs:annotation>
        <xs:documentation>
          Not absolutely required, but install unless directed not to
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
```