INTERNATIONAL STANDARD

**ISO/IEC 19770-2**

First edition
2009-11-15

# 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 19770-2:2009(E)



ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

# Table of Contents

Page

## Figures

Page



## Tables

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 19770-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

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

— *Part 1: Processes*

— *Part 2: Software identification tag*

— *Part 3: Software entitlement tag*

# Introduction

This part of ISO/IEC 19770 provides an International Standard for software identification tags. The software identification tag is an XML file containing authoritative identification and management information about a software product. The software identification tag is installed and managed on a computing device together with the software product. The tag may be created as part of the installation process, or added later for software already installed without tags. However, it is expected more commonly that the tag will be created when the software product is originally developed, and then be distributed and installed together with the software product. Having the tag available from the beginning allows for the more effective management of distribution and repackaging external to the software consumer, and then of release management within the software consumers organization.

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

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:

a) The ability to consistently and authoritatively identify software products that need to be managed for any purpose, such as for licensing, upgrading, packaging or for the specification of dependencies. Software identification tags provide the meta-data necessary to support more accurate identification which differentiates this approach from traditional file-oriented identification techniques.

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

c) Facilitation of de facto standardization between different software creators, and within software creator organizations, of how different versions of software are identified, allowing for better identification and management by software consumers of those different versions; for example, being able to distinguish between free-standing versions and versions which are components of suites, upgrade paths, etc.

d) Facilitation of automated approaches to license compliance, using information both from the software identification tag and from the software entitlement tag as will be specified in ISO/IEC 19770-3.

e) The ability to provide comprehensive information about the structural footprint of packages, i.e. the list of components such as files and system settings associated with that package, in order to link package-level management with file-level management.

f) The ability to provide information about how to identify if a particular software package is being actively used or not.

g) The ability to deal with the complexities of software installed on removable or shared storage, or in virtual environments (subject to the evolving ability of platforms and installers to identify devices and environments).

h) The ability to reflect within the software identification tag the identities and requirements of different entities, including software creators, software licensors, packagers, distributors external to the software consumer, release managers within the software consumer, and those responsible for installing and managing software on an on-going basis.

i) The ability to allow for the validation of any of this information through the optional use of digital signatures by anyone creating or modifying information in the software identification tag.

j) The ability for entities besides the software creators (e.g. independent providers, or in-house personnel) to create software identification tags for legacy software, and also for software from software creators who do not provide software identification tags themselves.

k) The ability of this International Standard to evolve in informal and formal ways, as common approaches become accepted throughout industry for dealing with additional types of information not currently covered by this part of ISO/IEC 19770, such as for product activation.

# Information technology — Software asset management —

# Part 2:
# Software identification tag



## 1 Scope

### 1.1 Purpose

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

### 1.2 Field of application

This part of ISO/IEC 19770 applies to:

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

b) Software providers: These are the entities that create ("software creators"), package ("software packagers") or license ("software licensors") software for distribution or installation. These include software manufacturers, independent software developers, consultants, and repackagers of previously manufactured software. They may also be in-house software developers.

c) Tag providers: These are the entities that create ("tag creators") or modify ("tag modifiers") software identification tags. A tag provider may be part of the software provider organization, or may be a 3rd party organization or the software consumer.

d) Tag tool providers: These are the entities that may provide any number of tools that create, modify or use software identification tags. These tools include development environments that provide automatically generated software identification tags, installation tools that may create and/or modify tags on behalf of the installation process as well as desktop management tools that may create tags for software that does not have a tag and/or modify tags with release details throughout the software lifecycle. See Annex C for details on how tool providers are likely to use software identification tags.

e) Software consumers: These are the entities that purchase, install and/or otherwise consume software, and who are intended as one of the major beneficiaries of the improved information provided by the software identification tag as specified in this part of ISO/IEC 19770. See Annex D for details on how software consumers are likely to use software identification tags.

### 1.3 Limitations

This part of ISO/IEC 19770 does not detail SAM processes required for reconciliation of software entitlements with software identification tags.

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 laws and regulations. Any such conflict should be resolved before using this part of ISO/IEC 19770.

# 2 Conformance

## 2.1 General

Conformance can apply to a product or an organization. For organizational conformance, the scope defined shall cover both the organizational scope as well as the products that are included in the scope.

If a claim of conformance is made for a product or organization, the claim shall specify the scope for which the conformance was tested.

Conformance throughout this clause is most often defined in terms of complying with the requirements of 6.1, 8.3, 8.4, and 8.5. Requirements for platform conformance are also specified in 7.2. There are also normative requirements specified in other subclauses of Clauses 6 and 7, indicated by the use of the word “shall”, but these are not included in the coverage of statements of conformance, except to the extent that they are also included in 6.1, 7.2, 8.3, 8.4, or 8.5. Statements including the word ‘should’ are recommendations but not mandatory.

## 2.2 Product conformance

### 2.2.1 Example reasons for product conformance

There are a number of reasons for an organization to seek individual product conformance to this part of ISO/IEC 19770. This may be sought when a specific product is being provided for a market that requires conformance (for example, if government organizations require products to conform to this part of ISO/IEC 19770 in order to be included on a project). It might also be desired by platform providers who want to provide a more secure and auditable tag storage that can be used to identify definitively which end-users installed which software packages.

### 2.2.2 Product scope

There shall be a clear statement for product scope describing, in unambiguous terms, the software products to which it applies and, where appropriate, clarifying the products to which it does not apply. The product conformance scope may be defined in any way considered appropriate, such as for a specific software product, for all software products, for all software products on specific platforms, for the software products of specified manufacturers and/or for all software products created after a specified date, as long as it is unambiguous. In the case of a product which creates or modifies software identification tags, the scope shall be the product itself and all software produced or modified by the product when tag-conformity functionality is enabled.

### 2.2.3 Software product conformance

Full conformance for a software product is achieved in one of two ways:

a) For a product which is installable, full conformance is achieved by demonstrating that all software identification tags installed by it at installation shall comply with all mandatory requirements of this part of ISO/IEC 19770, as specified in 6.1 and 8.3. If optional or extended tag elements are used these shall also comply with requirements as specified in 8.4 and 8.5.

   This conformance shall be demonstrated by performing equivalence partitioning with the exit criteria that all tests pass and 100 % equivalence partition coverage of the tag creation/installation is achieved. Equivalence partitions shall be derived from the statement of product scope.

If the software product consists of a package of other software products, then the software product shall retain all component tags and reference all child tag elements, which, under any circumstances, still need to be identified separately (for the purpose of licensing, security or other).

b) For a product that is distributable but not yet installed, full conformance is achieved by demonstrating that distributable builds are issued with a unique tag that shall comply with all mandatory requirements of this part of ISO/IEC 19770, as specified in 6.1 and 8.3. If optional or extended tag elements are used these shall also comply with requirements as specified in 8.4 and 8.5. The exception to this is that any mandatory elements which are installation-specific are not included.

   This conformance shall be demonstrated by performing equivalence partitioning with the exit criteria that all tests pass and 100 % equivalence partition coverage is achieved. Equivalence partitions shall be derived from the statement of product scope.

   If the software product consists of a package of other software products, then the software product shall retain all component tags and reference all child tag elements which under any circumstances still need to be identified separately (for the purpose of licensing, security or other).

### 2.2.4 Third party software identification tag conformance

Third party tag provider organizations may undertake the process of creating software identification tags for any software packages that do not include such tags. This may be done for older software products, shareware/freeware type products, or for companies that decide not to follow this part of ISO/IEC 19770. These tags may be provided to organizations to assist in their software discovery and identification procedures.

Full conformance for third party created software identification tags is achieved by demonstrating that all software identification tags produced by the organization comply with all mandatory requirements of this part of ISO/IEC 19770, as specified in 6.1 and 8.3. If optional or extended tag elements are used these shall also comply with requirements as specified in 8.4 and 8.5. Any new data that is added shall conform to the same standards as those required for installable software conformance.

Conformance for third party created software identification tags requires that the tag providers demonstrate that the software_ids they create are unique, and use consistent values for the identification of software providers. The expectation is that the tag providers will maintain a list of unique software providers for all tags created, and that the list includes a consistent software provider regid (that references the provider's domain) and a unique ID (which may be a GUID) for each reference and that these details are used consistently in the created tags.

This conformance shall be demonstrated by performing equivalence partitioning with the exit criteria that all tests pass and 100 % equivalence partition coverage of the tag production is achieved. Equivalence partitions shall be derived both from the range of software that the tag tool shall work on and the corresponding statements of product scope.

### 2.2.5 Software installer product conformance

Full conformance for a software installer product is achieved by demonstrating that all software identification tags installed by it at installation comply with all mandatory requirements of this part of ISO/IEC 19770, as specified in 6.1 and 8.3. If optional or extended tag elements are used these shall also comply with requirements as specified in 8.4 and 8.5.

This conformance shall be demonstrated by performing equivalence partitioning with the exit criteria that all tests pass and 100 % equivalence partition coverage of the tag creation/installation is achieved. Equivalence partitions shall be derived both from the range of software that is installed and the corresponding statements of product scope.

If the software being installed consists of a package of other software products, then the software product shall retain all component tags and reference all child tag elements which under any circumstances still need to be identified separately (for the purpose of licensing, security or other).

Existing tag values that are provided with distributable software shall not be modified in any way, with some specific exceptions. If a distributed software identification tag is found to be corrupted and that software identification tag does not provide a "validation" routine to fix the tag, a software product may provide options for handling this type of exception that a SAM practitioner can authorize. Based on actions specified by the SAM practitioner, the handling of such exceptions may include actions such as fixing the software identification tag if it is corrupt, deleting the software identification tag if it no longer belongs on the device, or modifying the software identification tag to specify that the software is no longer installed on the device. Should any modifications of the tag be specified by the user, these actions shall be logged and retained by the software product.

It is expected that such products will have the capability to turn this functionality on or off. A statement of product conformance shall apply only to the product with this functionality turned on.

### 2.2.6 Tag tool conformance

Full conformance for a tag tool is achieved in one of two ways:

a) Full conformance for a tag tool that installs or modifies installed software identification tags independent of software installation is achieved by demonstrating that all software identification tags installed or modified by the product comply with all mandatory requirements of this part of ISO/IEC 19770, as specified in 6.1 and 8.3. If optional or extended tag elements are used, these shall also comply with requirements as specified in 8.4 and 8.5. Any new data that is added shall conform to the same standards as those required for installable software conformance.

   This conformance shall be demonstrated by performing equivalence partitioning with the exit criteria that all tests pass and 100 % equivalence partition coverage of the tag production is achieved. Equivalence partitions shall be derived both from the range of software that the tag tool shall work on and the corresponding statements of product scope.

   If the software being installed consists of a package of other software products, then the software product shall retain all component tags and reference all child tag elements which under any circumstances still need to be identified separately (for the purpose of licensing, security or other).

   Existing tag values that are provided with distributable software shall not be modified in any way, with some specific exceptions. If a distributed software identification tag is found to be corrupted and that software identification tag does not provide a "validation" routine to fix the tag, a software product may provide options for handling this type of exception that a SAM practitioner can authorize. Based on actions specified by the SAM practitioner, the handling of such exceptions may include actions such as fixing the software identification tag if it is corrupt, deleting the software identification tag if it no longer belongs on the device, or modifying the software identification tag to specify that the software is no longer installed on the device. Should any modifications of the tag be specified by the user, these actions shall be logged and retained by the software product.

   It is expected that such products will have the capability to turn this functionality on or off. A statement of product conformance shall apply only to the product with this functionality turned on.

b) For a tag tool that discovers, collects, reports on and uses tags (such as discovery tools, desktop management tools or SAM reconciliation tools), full conformance is achieved by demonstrating the following.

   1) That all tags available on a computing device are collected. This includes tags that are stored in the common system location as well as tags that are located in the top level directories of software installations.

   2) That all tags collected from computing devices and stored in the tool's repository can be shown to include exactly the same information as the contents of the tag located on the computing device from which it was originally collected.

   3) If a tag is digitally signed and the corresponding public key is available, that the tool validates the signature and the information that has been signed.

This conformance shall be demonstrated by performing equivalence partitioning with the exit criteria that all tests pass and 100 % equivalence partition coverage of the tag collection/validation is achieved. Equivalence partitions shall be derived both from the range of software that the tool shall analyze and the corresponding statements of product scope.

If the software being installed consists of a package of other software products, then the software product shall retain all component tags and reference all child tag elements which under any circumstances still need to be identified separately (for the purpose of licensing, security or other).

Existing tag values that are provided with distributable software shall not be modified in any way, with some specific exceptions. If a distributed software identification tag is found to be corrupted and that software identification tag does not provide a "validation" routine to fix the tag, a software product may provide options for handling this type of exception that a SAM practitioner can authorize. Based on actions specified by the SAM practitioner, the handling of such exceptions may include actions such as fixing the software identification tag if it is corrupt, deleting the software identification tag if it no longer belongs on the device, or modifying the software identification tag to specify that the software is no longer installed on the device. Should any modifications of the tag be specified by the end-user, these actions shall be logged and retained by the software product.

It is expected that such products will have the capability to turn this functionality on or off. A statement of product conformance shall apply only to the product with this functionality turned on.

### 2.2.7 Platform conformance

Full conformance for a platform's tag functionality is achieved by demonstrating that it can store software identification tag data centrally and provide the following services with integrity, as specified in 7.2.

a) Basic functionality: add, modify, read, and delete tag data.

b) Security: determine which end-user can read, create, delete and modify software identification tags.

c) Audit functionality: identify which end-user installed, modified or removed a given software configuration item and when the modification occurred.

This conformance shall be demonstrated by performing equivalence partitioning with the exit criteria that all tests pass and 100 % equivalence partition coverage of the tag storage is achieved. Equivalence partitions shall be derived both from the range of software that the platform shall host and the corresponding statements of product scope.

## 2.3 Organizational conformance

### 2.3.1 Example reasons for organizational conformance

Organizations could want to conform to this part of ISO/IEC 19770 for a number of reasons. For example, software providers could want to promote their software products as being easier to manage. Also, software consumers could want to show that they are actively managing their software assets and that they can provide accurate information to any reconciliation or audit request.

### 2.3.2 Organizational scope

There shall be a clear statement for the organizational scope describing, in unambiguous terms, the organizational structure to which it applies and, where appropriate, clarifying the areas to which it does not apply. A statement of organizational scope shall be accompanied by a statement of software product scope.

### 2.3.3 Software provider conformance

Full conformance for a software provider is achieved by the organization demonstrating that all software within the scope meets the relevant product conformance requirements, as specified in 2.2.3.

#### 2.3.4 Tag tool provider conformance

Full conformance for a tag tool provider is achieved by an organization demonstrating that all software within the scope meets the relevant tool conformance requirements, as specified in 2.2.6.

Furthermore, in order to claim tag tool provider conformance, all tag tools produced by the organization shall be included in the product scope.

#### 2.3.5 Software consumer conformance

Full conformance for an organization that installs software is achieved by demonstrating that there are software identification tags in place for all software in the software consumer organization's product scope and that the software identification tags comply with all mandatory requirements of this part of ISO/IEC 19770, as specified in 6.1 and 8.3. If optional or extended tag elements are used, these shall also comply with requirements in 8.4 and 8.5.

### 2.4 Agreement compliance

This part of ISO/IEC 19770 may be used to help develop an agreement between a software provider and a software consumer, in which case clauses of this part of ISO/IEC 19770 can be selected for incorporation into the agreement, with or without modification. In such an instance, it is necessary for both parties to comply with their agreement rather than conform to this part of ISO/IEC 19770.

NOTE ISO/IEC's copyright and patent policy extends to all of this part of ISO/IEC 19770 and contents thereof. However, for the specific use of agreement compliance, there is no need to obtain copyright permission.

## 3 Normative references

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

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax,* 2005

IETF RFC 4646, *Tags for Identifying Languages,* 2006

W3C Recommendation, *XML Signature Syntax and Processing (Second Edition)*, 2008

*W3C Recommendation, XML Schema Part 2: Datatypes (Second Edition)*, 2004

UNSPSC, *The United Nations Standard Products and Services Code*

## 4 Terms, definitions and abbreviated terms

### 4.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

**4.1.1**
**application**
system for collecting, saving, processing, and presenting data by means of a computer

NOTE The term application is generally used when referring to a component of software that can be executed.

**4.1.2**
**bundle**
grouping of products which is the result of a marketing/licensing strategy to sell entitlements to multiple products as one purchased item

NOTE 1 A bundle can be referred to as a "suite", if the products are closely related and typically integrated (such as an office suite containing a spreadsheet, word processor, presentation and other related items).

NOTE 2 Bundles can also refer to software titles that are less closely related such as a game, a virus scanner and a utility "bundled" together with a new computer, or to groups of entitlements, such as multiple entitlements for a backup software product.

**4.1.3**
**component**
entity with discrete structure, such as an assembly or software module, within a system considered at a particular level of analysis

NOTE Component refers to a part of a whole, such as a component of a software product, a component of a software identification tag, etc.

**4.1.4**
**computing device**
functional unit that can perform substantial computations, including numerous arithmetic operations and logic operations without human intervention

NOTE A computing device can consist of a stand-alone unit, or several interconnected units. It can also be a device that provides a specific set of functions, such as a phone or a personal organizer, or more general functions such as a laptop or desktop computer.

**4.1.5**
**configuration item**
**CI**
item or aggregation of hardware or software or both that is designed to be managed as a single entity

NOTE Configuration items may vary widely in complexity, size and type, ranging from an entire system including all hardware, software and documentation, to a single module, a minor hardware component or a single software package.

**4.1.6**
**configuration management database**
**CMDB**
database containing all the relevant details of each configuration item and details of the important relationships between them

**4.1.7**
**customer**
end-users or organizations for which a software publisher designs and develops software and sells entitlements to use that software

**4.1.8**
**element**
component of a software identification tag that provides information related to the software represented by the tag

NOTE The different types of elements are defined in 8.3, 8.4 and 8.5.

**4.1.9**
**end-user**
person (or persons) who operate or interact directly with a computing device to manage or use software packages

**4.1.10**
**equivalence partitioning**
software testing process that identifies representative groups of input values that are processed in the same manner by software products allowing for a sampling of each representative group to validate the outcome resulting in a reduced number of test cases while ensuring full coverage of all test cases

**4.1.11**
**extensible markup language**
**XML**
license-free and platform-independent markup language that carries rules for generating text formats that contain structured data

**4.1.12**
**globally unique identifier**
**GUID**
16-byte string of characters that is generated in a manner that gives a high probability that the string is unique in any context

NOTE 1 Other globally unique identifier algorithms can be used in some situations. In general, alternative algorithms use Uniform Resource Identifier (URI) based structures, so the id owner's registration identifier (regid) is included in the identifier.

NOTE 2 GUID as an all capitalized term refers specifically to the 16 byte version. If the term is in lowercase (guid), it refers to a general algorithm that can use either a URI, or a 16-byte-based identifier.

**4.1.13**
**legacy software**
software originally created without software identification tags

**4.1.14**
**line of business application developer**
person or company specializing in developing applications providing specific functions for a particular business operation

**4.1.15**
**MD5**
**Message-Digest algorithm 5**
algorithm that is a widely-used cryptographic hash function with a 128 bit hash value often used to identify if two files contain the same data

**4.1.16**
**package**
set of related components that are combined into a single distributable item

NOTE For example, a software package would be a set of files that can be used to install software on a computing device and can be distributed via CD or electronic means.

**4.1.17**
**platform**
computer or hardware device and/or associated operating system, or a virtual environment, on which software can be installed or run

NOTE Examples of platforms include Linux™, Microsoft Vista®, and Java™.

**4.1.18**
**platform provider**
organization responsible for the platform

NOTE The platform provider is typically the vendor of the relevant operating system or virtual environment.

**4.1.19**
**product**
complete set of computer programs, procedures and associated documentation and data designed for delivery to a software consumer

NOTE The terms "product" and "software package" are used interchangeably depending on the context of the item described.

**4.1.20**
**registration identifier**
**regid**
identifier created from a domain name and the date when the domain was owned by a specific individual or company, allowing an individual or company to have their own unique namespace and be their own registration authority for all software configuration items they publish without requiring a separate industry based registration authority

**4.1.21**
**release**
collection of new and/or changed configuration items which are tested and introduced into a production environment together

**4.1.22**
**release manager**
individual responsible for the collection of new and/or changed configuration items which are tested and introduced into an organization's live production environment

**4.1.23**
**SAM owner**
individual at a senior organization-wide level who is identified as being responsible for SAM

**4.1.24**
**SAM practitioner**
individual involved in the practice or role of managing software assets

NOTE A SAM practitioner is often involved in the collection or reconciliation of software inventory and/or software entitlements.

**4.1.25**
**software**
all or part of the programs, procedures, rules, and associated documentation of an information processing system

**4.1.26 Software Asset Management**
**SAM**
effective use, control and protection of software assets within an organization

**4.1.27**
**software consumer**
organization or person who buys entitlements to use a software package

**4.1.28**
**software creator**
person or organization that creates a software product or package

NOTE This entity might or might not own the rights to sell or distribute the software.

**4.1.29**
**software developer**
person who creates software that can perform a specified set of actions

NOTE Often a software developer works with other developers for a software manufacturer to create commercial applications. A software developer can also often work as an in-house developer of software for use by the software developer's own organization.

**4.1.30**
**software entitlement**
legal ownership of software license use rights as defined through agreements between a software purchaser and the software copyright holder

NOTE Effective use rights take into account any contracts and all applicable licenses, including full licenses, upgrade licenses and maintenance agreements.

**4.1.31**
**software identification tag**
file comprised of mandatory elements, optional elements and extended information containing authoritative identification information about a software configuration item

NOTE For mandatory elements see 8.3, for optional elements see 8.4, for extended information see 8.5.

**4.1.32**
**software license**
legal rights to use software in accordance with terms and conditions specified by the software copyright owner

NOTE "Using a software product" can include: accessing, copying, distributing, installing and executing the software product, depending on that product's terms and conditions.

**4.1.33**
**software licensor**
person or organization who owns the rights to issue a software license for a specific software package

**4.1.34**
**software manufacturer**
group of people or organization that develops software, typically for distribution and use by other people or organizations

**4.1.35**
**software package**
complete and documented set of programs supplied for a specific application or function

NOTE In this part of ISO/IEC 19770, the term software package refers to the set of files associated with a specific set of business functionality that can be installed on a computing device and has a set of specific licensing requirements. In this part of ISO/IEC 19770, the terms "product" and "software package" are used synonymously depending on the context of the item described.

**4.1.36**
**software packager**
entity that re-packages or bundles software created by others

NOTE This can be done by a value added reseller who bundles a software package to work with an embedded system, or by a software reseller who is licensed to combine a number of different software products into a single bundle.

**4.1.37**
**software provider**
entity that creates (software creator), modifies (software modifier) or licenses (software licensor) software for distribution or installation

NOTE This includes software manufacturers, independent software developers, consultants and repackagers of previously manufactured software. IT can also represent in-house software developers.

**4.1.38**
**software publisher**
person, group or company that packages and distributes software and might or might not be the software manufacturer

**4.1.39**
**tag creator**
entity that initially creates the software identification tag

NOTE This entity can be part of the organization that created the software, in which case the tag creator and software creator will be the same. The tag creator can also be a third party organization unrelated to the software creator (such as in the case where tags are created for legacy software).

**4.1.40**
**tag modifier**
software packager or software consumer that modifies a tag after it has been created

NOTE Modification of any tag is limited to the elements that the software licensor has authorized and is done based on license or contractual agreements with the tag creator and/or software creator. The tag modifier can be allowed to add values to a software identification tag (such as the case of a reseller adding details about where the product was purchased), or can be allowed to modify existing portions of the tag (such as the case of a VAR making a set of software look like it comes from a single entity).

**4.1.41**
**tag provider**
entity that creates (tag creator) or modifies (tag modifier) software identification tags for software packages

NOTE A tag provider can be part of the software provider organization, or can be a third party organization or the software consumer.

**4.1.42**
**Uniform Resource Identifier**
**URI**
compact sequence of characters that identifies an abstract or physical resource available on the Internet

NOTE The syntax used for URIs is defined in IETF RFC 3986.

**4.1.43**
**valid**
<XML file> software identification tag data follows the specified XSD definition and the software identification tag file is valid from an XML perspective

NOTE See also 4.1.44.

**4.1.44**
**valid**
<software identification tag> process used to ensure the data included in an installed software identification tag is correct

NOTE See also 4.1.43.

**4.1.45**
**value-added reseller**
company licensed to repackage and support existing products as combined software packages

**4.1.46**
**version**
unique string of number and letter values indicating a unique revision of an item

NOTE Versions are often referred to in software to identify revisions of software that provide unique functionality or fixes. A version typically has multiple parts with at least a major version indicating large changes in functionality or user interface changes and a minor version indicating smaller changes in functionality or user interface changes.

**4.1.47**
**XML Schema Definition**
XML based language that specifies a set of rules and structure for the creation of XML documents

NOTE XML documents follow all rules defined in an XSD definition in order to be considered a "valid" document.

### 4.2 Abbreviated terms

| | |
|---|---|
| CI | configuration item |
| CMDB | configuration management database |
| GUID | globally unique identifier |
| IETF | Internet Engineering Task Force |
| MD5 | message digest 5 |
| regexp | regular expression |
| regid | registration identifier |
| SAM | software asset management |
| 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 |

## 5 Alignment and rationalization with prior standards

### 5.1 Statement of alignment for this part of ISO/IEC 19770

The contents of this part of ISO/IEC 19770 are intended to complement and align with prior ISO/IEC standard publications.

### 5.2 Alignment with ISO/IEC 19770-1:2006 Information technology — Software asset management — Part 1: Processes

The following areas of ISO/IEC 19770-1:2006 are supported by this part of ISO/IEC 19770.

a) ISO/IEC 19770-1:2006, clause 3 stipulates terms and definitions relevant to that document.

This part of ISO/IEC 19770 aligns with ISO/IEC 19770-1:2006, terms and definitions in clause 3 relevant to both parts have been reproduced here.

b) ISO/IEC 19770-1:2006, clause 4.4.2.2 stipulates: "Implementation of the software asset identification process will enable the organization to demonstrate that a) types of assets to be controlled and the information associated with them are formally defined. b) a register of stores and inventories exists,

clarifying which stores and types of information are held, with duplication allowed only if duplicate information can be traced back to the definitive source record."

This part of ISO/IEC 19770 affirms the necessity of formal definitions, stores and inventories for software configuration items. According to ISO/IEC 19770-1:2006, clause 4.4.2.2.a.2.i, software configuration items include "all platforms on which software can be installed or run." There is no explicit requirement for a "platform" element in 19770-2 because this part of ISO/IEC 19770 is focused on the identification of software found on computing devices and not on the requirements of a general software repository. It is expected that discovery tools collect platform information during their inventory process.

c) ISO/IEC 19770-1:2006, clause 4.4.2.2 stipulates: "Basic information required for all assets is i) Unique identifier ii) Name/description iii) Location iv) Custodianship (owner) v) Status (e.g., test/production status; development or build status) vi) Type (e.g., software, hardware, facility), vii) Version (where applicable)."

This part of ISO/IEC 19770 affirms these requirements for basic information. The location and custodianship of a software configuration item, however, are not included as values specified in this part of ISO/IEC 19770 as these are associated with the asset on which the software configuration item is discovered and not with the item itself.

The status of a software configuration item is defined by the Release values in the software identification tag. These values are optional and it is recommended that they are furnished alongside information pertaining to the sign off date and the operator who performed the process (8.4).

### 5.3 Alignment with ISO/IEC 20000-1:2005 Information technology – Service management – Part 1: Specification

a) ISO/IEC 20000-1:2005, clause 9.1 stipulates: "Changes to configuration items shall be traceable and auditable where appropriate, e.g. for changes and movements of software and hardware." This part of ISO/IEC 19770 affirms the usage of software identification tags for disclosure and definition of traceable and auditable information for software configuration items.

b) ISO/IEC 20000-1:2005, clause 9.1 stipulates: "Configuration control procedures shall ensure that the integrity of systems, services and service components are maintained."

This part of ISO/IEC 19770 upholds the necessity of configuration control procedures for integrity assurance purposes and therefore provides creators with the option to include a digital signature (6.1.11). The digital signature can be used to validate that specified mandatory element values have not been modified, this validation in turn allowing software providers or tag providers to authoritatively identify software identification tag tampering, or lack thereof.

c) ISO/IEC 20000-1:2005, clause 9.1 stipulates: "Master copies of digital configuration items shall be controlled in secure physical or electronic libraries and referenced to the configuration records, e.g. software, testing products, support documents."

This part of ISO/IEC 19770 that recommends software identification tags be included with respective software configuration items in the definitive software library.

d) ISO/IEC 20000-1:2005, clause 9.1 stipulates: "All configuration items shall be uniquely identifiable and recorded in a CMDB to which update access shall be strictly controlled."

This part of ISO/IEC 19770 aligns with the specification that all software configuration items be uniquely identifiable. A software configuration item is uniquely identifiable by product identifier, serial number, stock keeping unit, either of which can be related back to proof of license, purchase order and software configuration items stored in the CMDB (8.4.14, 8.4.20, 8.4.21 ).

The method by which a software identification tag is stored in the CMDB and referenced uniquely therein is not within the scope of this part of ISO/IEC 19770 (1.2).

e) ISO/IEC 20000-1:2005, clause 10.1 stipulates: "Release and distribution shall be designed and implemented so that the integrity of hardware and software is maintained during installation, handling, packaging and delivery."

This part of ISO/IEC 19770 affirms the necessity of integrity within the release process and therefore specifies that software identification tags fully conform to release processes detailed in ISO/IEC 20000-1:2005. This part of ISO/IEC 19770 recommends that optional elements pertaining to release details be included with software identification tags (8.4).

### 5.4 Alignment with ISO/IEC 20000-2:2005 Information technology — Service management — Part 2: Code of practice

a) ISO/IEC 20000-2:2005, clause 10.1.5 stipulates: "Release and distribution should be designed and implemented to: a) conform with the service provider's systems architecture, service management and infrastructure standards; b) ensure the integrity is maintained during build, installation, handling, packaging and delivery..."

This part of ISO/IEC 19770 affirms the importance of release and distribution conformance through the creation of the optional element "Release package" (8.4.17).

b) ISO/IEC 20000-2:2005, clause 10.1.6 stipulates: "The verification and acceptance processes should: a) verify that the controlled acceptance test environment matches the requirements of the target production environment; b) ensure that the release is created from versions under configuration management and installed in the acceptance test environment using the planned production process..."

This part of ISO/IEC 19770 affirms the importance of matching the controlled acceptance test environment to the requirements of the target production environment through the creation of the optional element "Release verification" (8.4.19).

c) ISO/IEC 20000-2:2005, clause 10.1.8 stipulates: "It is important that the release is delivered safely to its destination in its expected state."

This part of ISO/IEC 19770 upholds the need for efficient and secure release delivery by proposing the creation of the optional element "Release rollout" that allows an organization to validate who signed off on a software package as ready for production use and when the sign off occurred (8.4.18).



## 6 Implementation of software identification tagging processes

### 6.1 General requirements and guidance

#### 6.1.1 Software identification tag overview

Annex A provides an overview of software identification tagging principles from a more conceptual perspective, to assist in understanding.

#### 6.1.2 XML and XSD

The software identification tag shall be defined as an XML data structure. The XML Schema Definition (XSD) to be used shall be that defined in Annex G, or any (updated) version which may be downloaded from:

http://standards.iso.org/iso/19770/-2/2009/schema.xsd

Additional versions of the schema may be available with the version identifier of the schema included in the path to the schema. All prior versions of the schema shall be retained.

### 6.1.3 Unique registration ID (regid)

Software identification tags may be created by multiple 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 configuration items 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 will use a regid. The regid is based off of the iSCSI Qualified Name as defined in the IETF RFC 3720 – section 3.2.6.3.1 and the IETF RFC 3721 section 1.1 and provides a unique naming authority reference.

A regid can be created by any individual or organization that owns or has owned the registration for a domain name (as specified in IETF RFC 1034, section 3.5 and IETF RFC 1123, section 2.1). The domain name does not need to be active, nor does it need to resolve to an address. Domain names by themselves do not constitute a unique identifier since domains because they can expire and/or be acquired by other entities this means a regid must also include a date that the domain registration was owned by the entity. Finally, for entities that wish to further sub-divide unique naming sub-entities, an optional suffix is provided for the regid that may be used, for example, to provide large software publishers means to allow each of their business units to manage their own software identification tags independently.

The regid name shall consist of the following:

- The string "regid" – this qualifies the element as a registration id for software identification tags.
- A dot '.'
- A date code in YYYY-MM format. This date shall be a date during which the naming entity owned the domain. This date should be the first month in which the domain name was owned by this naming entity at 00:01 GMT of the first day of the month. This date code uses the Gregorian calendar and must include all four digits of the year and both digits of the month (where January = 01 and December = 12). The dash must be included.
- A dot '.'
- The reversed domain name of the naming entity (person or organization) creating a software identification tag
- An optional string that specifies sub-entities that may be their own unique naming authorities. This is specified by:
    - A comma ','
    - With the exception of the comma prefix, the owner of the domain name can assign text following the reversed domain name as desired as long as all characters are valid for use in filenames on any platforms the tag will be installed on. It is the responsibility of the naming entity to ensure that each sub-entity reference is unique within their organization.



An example of regids created by entities owning example.com or example.net looks as follows:

**Table 1 — Examples of regid values**

```
                  Naming      Additional
Type   Date       Auth        "example.com" naming authority
+---+ +-----+ +---------+ +--------------------------------+
|   | |     | |         | |                                |

regid.1995-09.com.example,AccountingSystems
regid.1995-09.com.example
regid.1995-09.net.example,WordProcessing
```

### 6.1.4 Software identification tag extension and location for installation

Software identification tag files shall include the ".swidtag" file extension in their names for the recognition purposes of platforms (4.1.7) and discovery tools.

Each platform provider (4.1.18), e.g. vendor of an operating system, should specify where software identification tags are to be located. Windows® installations, for instance, may have a Windows Management Instrumentation™ value specifying location; Linux™ distributions may use a Red Hat Package Manager value. If the platform provider does specify a location, then the software identification tag shall be installed in this location.

In the absence of specifications from the platform provider, software identification tags should be installed in commonly known shared locations that are used for collecting commonly used system information. The following examples provide information on which shared locations are expected to be used for various platforms, in the absence of an alternative specification from the platform provider.

**Table 2 — Examples of tag locations on different platforms**

| | |
|---|---|
| Apple Macintosh™ OS:X™ Leopard | <root>/Library/Application Support/**<software creator regid>** |
| Apple Macintosh™ OS X™ pre-Leopard<br><br>NOTE software identification tags should be included in the application directory by default for all operating systems (see below). Pre-Leopard OS X systems should also use this location as the default location. | Application Directory/<program.app package>/contents |
| UNIX® and Linux™ | usr/share/**<software creator regid>** |
| Windows® NT | C:\Winnt\All Users\Application Data**<software creator regid>** |
| Windows® 2000 Professional<br>Windows Server® 2000<br>Windows® XP<br>Windows Server® 2003 | %AllUsersProfile%\Application Data**<software creator regid>** |
| Microsoft Vista®<br>Microsoft Server® 2008 | %Program Data%**<software creator regid>** |



The platform provider may provide access to the software identification tag using methods that are independent from file access. For example, Microsoft Vista® includes 4 APIs that may be used to manage a software identification tag repository. These API's are:

**Table 3 — Microsoft Vista® APIs for software identification tag management**

| | |
|---|---|
| SLGetInstalledSAMLicenseApplications | Retrieves a list of applications that have a software identification tag installed by using SLInstallSAMLicense |
| SLGetSAMLicense | Gets information about a specific software identification tag installed by using SLInstallSAMLicense |
| SLUninstallSAMLicense | Removes a software identification tag for a specified application |
| SLInstallSAMLicense | Adds a software identification tag to the Microsoft Vista® repository |

In addition a copy of the software identification tag shall also be installed in the top level directory of the application itself. This allows the software identification tag to be discovered even if a removable storage device (such as a USB hard disk) is moved from one system to another (7.5). Should the same software be installed in two different locations for the same set of end-users, the expectation is that there would still be two (or more) instances of the software identification tag in the common system location specified above as well as two (or more) other tags that are located in the root directories of the installation directories. In this case, a software uninstaller needs to be aware of the multiple installations at uninstall time in order to ensure that a software identification tag is not removed from the common system location until all installations are uninstalled.

NOTE SAM tool providers should be aware that there will be cases where a software identification tag can be found in the common system location as well as in one or more top level directories of a software package installation. Using information in the installation_details element, SAM tools can associate tags located in the common system location and installation root directories with their corresponding installations. Rules need to be included in the SAM tools to deal with various permutations of tags which are found. In cases where the tag is found in the software package installation directory, and not in the common system location, additional rules are required to identify if software is located on removable storage media that may have been moved to another system. If a tag is found in the common system location as well as multiple tags found in installation directories, rules may need to be applied as appropriate for the organizational policies. Appropriate reporting and action can then be taken by the SAM practitioner.

The goal of a SAM tool should be to make it as easy as possible for a SAM practitioner to manage exceptions to organizational policy, so recognizing some of these issues and reporting on them based on the policies specified by the practitioner make overall SAM implementations significantly easier to manage.

### 6.1.5 Unique identifiers

For the purpose of uniqueness, there are two elements that, combined, shall create a globally unique ID called the software_id. These elements are

a) tag_creator_regid

b) unique_id that may be either a GUID, or any reference unique for the tag_creator_regid. The unique_id shall follow the restrictions for URI character use as specified in IETF RFC 3986, section 2, Characters.

The benefits of implementing a unique identifier during software identification tag creation include, but are not limited to, facilitation of the following:

a) Identification of parent-child relationships.

b) Explicit definition of dependencies and recognition of dependent software.

c) Identification of upgrade software and allowed upgrade packages.

d) Reference to identifying software identification tags from within software configuration items.

### 6.1.6 Unique software identification tag file name – distribution

When a software identification tag is created for distribution on installation media, it is not possible to provide additional installation-specific unique ids as described in section 6.1.7 since the tag has not been installed on any computing device as yet. In this case, the tag file is part of the master image for the installation process and shall align with the following structure:

<tag_creator_regid>_<software_id.unique_id>.swidtag

Following this structure provides a unique filename that can ship with the software.

A software identification tag on the installation media would thus typically have a filename such as:

regid.1986-12.com.adobe_fc3cc419-b5a1-9f16-ed203e537c40.swidtag

The installation routine shall then follow the process identified in section 6.1.7 when the tag is installed.

### 6.1.7 Unique software identification tag file name - installed

When software is installed on a computing device, the name of a software identification tag shall be unique at least to the specific code to be installed, and align with the following structure:

<tag_creator_regid>_<software_id.unique_id>_<unique_sequence_id>.swidtag

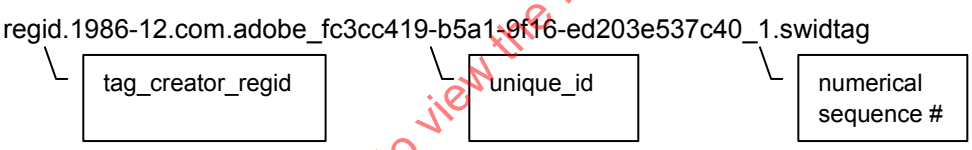
The unique_sequence_id is optional and shall be used to ensure that every software identification tag installed on a computing device has a unique filename. This unique_sequence_id may be a simple numerical sequence, or it may be created using an algorithm chosen by the organization that installs the software identification tag. The choice of how to create the unique_sequence_id is up to the organization installing the software identification tag, but the methodology used shall ensure that the filename is unique for a specific machine and/or virtual environment. The initial portion of a software identification tag filename would look as follows:

regid.1986-12.com.adobe_fc3cc419-b5a1-9f16-ed203e537c40

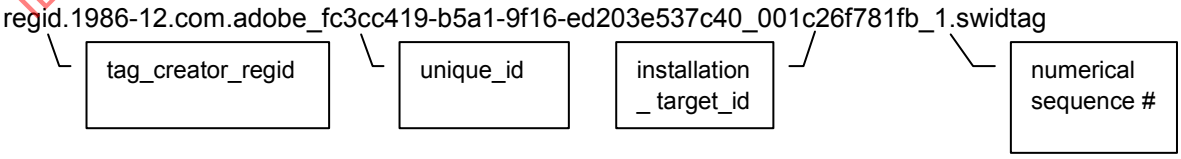
This would be followed by the unique_sequence_id. The algorithm to specify the unique_sequence_id may include some of the following considerations:

1. For basic tag filenames, the unique_sequence_id may simply be a sequential value that, if during installation of a software identification tag, a tag with the same name already exists, the sequential number is incremented. In this instance, the numerical sequence would start at a specified reference number and increment as other software identification tags for the same software ID are installed (this would happen if and only if the software allows multiple installations on a particular computing device). Note that the installation_instance within the optional installation_details element could be used as part of the filename in this way.

   An example of this software identification tag file name (for the first installation) is:

2. The preferred algorithm for the unique_sequence_id should include details specific to the computing device and/or media on which the software is installed (this may include device serial number, NIC MAC addresses, hard disk serial number, or other unique reference items) and/or details about the virtual environment where the software has been installed. Note that the installation_target_id within the optional installation_details element may be used as part of the filename in this way. Having device -specific details facilitates the tracking of software identification tags that may be installed on removable or shared media. Note that this algorithm may also require an additional unique sequence number to ensure that multiple installations on the same machine do not create the same filename. The installation_instance within the optional installation_details element could be used as part of the filename in this way. An example of this software identification tag file name is:

   Regardless of the algorithm chosen, the software identification tag filename shall be created by the installation routine that installs the tag and the name must not conflict with any existing filename. Additionally, the characters used in the filename shall not exceed 254 characters (or less if the targeted platform for the software requires shorter file names). Finally, the characters used in the filename shall meet all specific criteria required for the file systems that the tag is targeted for.

The .swidtag file extension shall be used for all software identification tags. This naming scheme allows for multiple software identification tags to be applied to the same product title, thereby providing support for

upgrades. It also allows for unique software identification tags to be created by organizations other than the original software creator, such as for legacy software.

### 6.1.8 Consistency among data values

One challenge for software asset management is the fact that marketing names used by software providers change frequently. Since the software identification tag will not typically be seen by the end-user, this tag can provide a consistent and reliable source of information to enable effective inventory and software entitlement reconciliation. Keeping values consistent from one version to another or across product lines is a large part of the value software identification tags provide the SAM practitioner.

Numerous elements in the software identification tag should remain consistent from tag to tag to optimize the overall software asset management process. It is recommended, for example, that tag providers maintain consistency in the following elements:

a) Software creator identity (8.3.4) – this element should remain consistent across all software packages created by a specific company.

b) Software licensor identity (8.3.5) – this element should remain consistent across all software packages licensed by a specific company.

c) Tag creator identity (8.3.7) – this element should remain consistent across all software identification tags created by a specific company.

d) License and channel information (8.4.8) - this optional element structure should remain consistent across product lines and it is recommended that the values are consistent across all software created by a specific company. Note that data in the 'license and channel information' elements of a software identification tag do not specify software license or software entitlement information – instead, data in these elements provides guidance to a SAM practitioner that will help them determine and perhaps automate software license reconciliation procedures.

e) Product category (8.4.12) – this optional element should remain consistent for a particular product unless the product adds or removes functionality to make it obvious that it belongs in a different category.

f) Product identifier (8.4.14) – this optional element should remain consistent especially for products that have maintenance agreements that provide for upgrade rights over a period of time. his element is used to identify a specific product from release to release – this element is not a product name, it is simply an identifier so the relationship for upgrade purposes can be done automatically.

### 6.1.9 Software identification tag discovery

Software is generally created as a gold master copy by software providers, then copied and distributed through different channels. Depending on the software provider's requirements, the software identification tag may be incorporated directly into the gold master, or it may be created by the installer, or even the software package itself. The primary requirement is that the software identification tag for the package shall be discoverable on the machine, media and/or virtual environment on which the package is installed. For more details, refer to the annex detailing guidance for software providers (Annex B).

### 6.1.10 Languages

Acknowledging that many software creators produce software with specific builds that are dependent on language while many others produce software with one build that implements add-on "language packs," this part of ISO/IEC 19770 does not require software identification tags to recognize different language versions of the same product. Consistent categorization through the use of the supported languages element, is however, strongly encouraged (8.4.24).

For encoding purposes, the use of utf-8 is the suggested methodology for software identification tags created based on this part of ISO/IEC 19770 (see http://www.w3.org/International/O-charset).

### 6.1.11 Ownership of elements within software identification tags

Because of the way that different entities may add or modify elements within the software identification tag, it may be necessary for any one of these entities to identify which elements it has created and/or modified. This capability is provided primarily via the elements_owner element, making use of the internal element ID as described in 6.1.12.

Alternatively, ownership of individual elements may be identified through the use of digital signatures as described in 6.1.13. However, the use of digital signatures does not directly identify an owner in the same way as the elements_owner does, so the use of the elements_owner element is still recommended even when digital signatures are used.

### 6.1.12 Internal element ID

There is a requirement to be able to cross-refer internally within the software identification tag to other individual elements within the software identification tag. One obvious situation where this is required is in the ability to identify the elements which have been created and/or modified by a specified tag creator or modifier, as listed in the elements_owner element. This cross-reference capability is provided by assigning internal element IDs to the individual elements which may then be used to cross-refer to those elements from other elements.

Internal element IDs are used for intra-tag reference as well as for inter-tag references where a specific element from another tag must be identified (this may happen in the case where a digital signature is provided in a secondary file). These elements IDs must be unique within a software identification tag, but need not be unique between tags. For software identification tags, ID attributes are typically used to identify specific ownership of elements. The tag creator and modifier may use any defined methodology to specify and use ID attributes. However, it is useful if a default convention is provided and used when possible. he default convention suggested for this part of ISO/IEC 19770 is:

a) Every XML ID must start with an alphabetic character. The convention for this part of ISO/IEC 19770 is for the ID to start with the letter 'e' standing for element.

b) Each top level element should utilize the same value as the paragraph defining that element with each component of the paragraph denoted with an underscore. This results in the following attribute for the element product_version:

```
<swid:product_version ID=”e8_3_3”>
```

c) Every element may have sub-elements. In these cases, for each sub element, the ID should start with the top level paragraph identifier (as noted above), followed by "sub[number of sub element]". This results in the following IDs specified for the structure under product_version:

```
<swid:name ID=”e8_3_3sub1”>10.2</swid:name>
<swid:numeric ID=”e8_3_3sub2”>
  <swid:major ID=”e8_3_3sub2sub1”>10</swid:major>
  <swid:minor ID=”e8_3_3sub2sub2”>2</swid:minor>
  <swid:build ID=”e8_3_3sub2sub3”>0</swid:build>
  <swid:review ID=”e8_3_3sub2sub4”>0</swid:review>
</swid:numeric>
```

d) For values that may have multiple entries specified, such as abstract, the ID should start with the top level paragraph identifier (as noted above), followed by a sequence number as specified by "seq[number of instance]". This results in the following IDs specified for the structure under abstract:

```
<swid:abstract lang="en" ID=”e8_4_1_seq1”>This is the abstract written in English</swid:abstract>
<swid:abstract lang="fr" ID=”e8_4_1_seq2”>This is the abstract written in French</swid:abstract>
```

Following these recommendations for the ID references used by elements_owner provides a consistent approach to specifying IDs that may readily be understood by an engineer who may be reading the tag as well as being managed effectively through automated approaches.

### 6.1.13 Authenticity of software identification tags

Often, there will be a desire, or need to prove the authenticity of a software identification tag. For example, during an audit, a software vendor may want to validate that the software identification tag collected during a discovery process has not had key elements of the tag altered. Authentication is supported by allowing digital signatures within the software identification tag.

There is an existing recommendation published by the W3C that addresses the need to provide digital signatures in an XML document. The recommendation is "XML-Signature Syntax and Processing (Second Edition) – 10 June, 2008" found at the following location:

http://www.w3.org/TR/xmldsig-core/

NOTE W3C does not reference official documents as "standards". Instead, the officially supported documents are referred to as "recommendations". The reader should be aware that W3C follows a well defined process to get a document to a recommendation level and that W3C recommendations should be considered as authoritative documents.

The W3C recommendation provides integrity message authentication as well as signer authentication services for data of any type.

This part of ISO/IEC 19770 does not define the process for applying digital signatures to a software identification tag since the W3C recommendation already does that.

Signatures are not a mandatory part of the software identification tag, and can be used as required by any tag creator or modifier to ensure that sections of a tag are not modified and/or to provide authentication of the signer. If signatures are required for the software identification tag, they shall follow the W3C recommendation defining the XML Signature Syntax (http://www.w3.org/TR/xmldsig-core/).

NOTE Software identification tags will generally not require XML Advanced Electronic Signatures (XAdes), so this W3C recommendation is not referenced in this part of ISO/IEC 19770.

### 6.1.14 Standardization of XSD definition

There are a number of standardized types used in the software identification tag including specific date/time entries that shall follow a specified formats as specified in the W3C recommendation titled, "*XML Schema Part 2: Datatypes Second Edition*". Details for these data types can be found at the following location:

http://www.w3.org/TR/xmlschema-2/

By using specific types as specified by the above W3C recommendation, software identification tags can go through an automated validation step that allows a much more consistent structure to the data provided.

NOTE A few elements defined in this part of ISO/IEC 19770 provide for the use of regular expressions. The syntax that is supported for these regular expressions is also defined in the W3C recommendation listed above.

## 6.2 Software identification tagging life cycle: operational breakdown

### 6.2.1 Introduction

The following diagram shows the software identification tag lifecycle as it progresses from a software creator (or tag creator) to a software consumer organization.

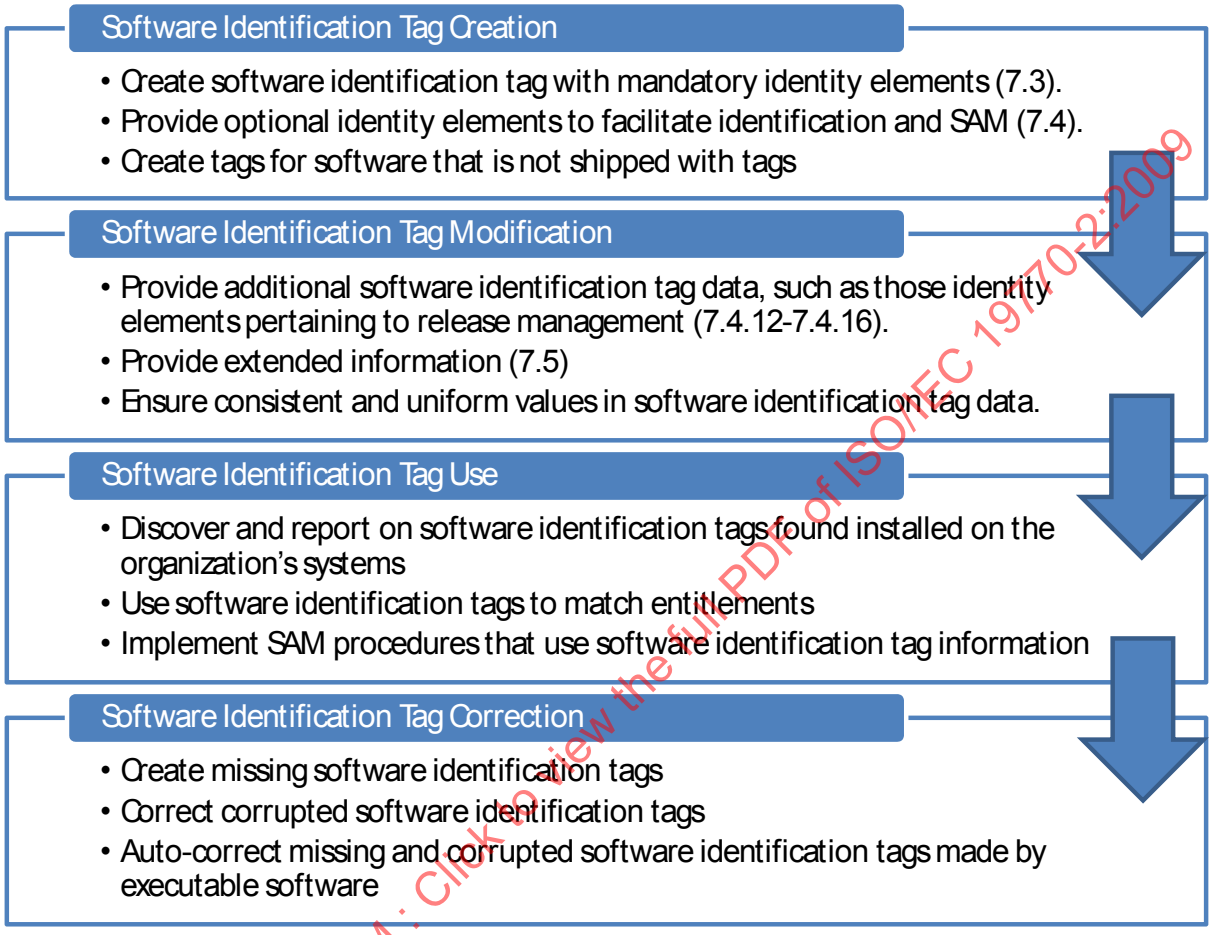



**Figure 1 — Software identification tag lifecycle**

### 6.2.2 Software identification tag creation

The software identification tag creator will often be the same as the software creator. As software is built, the software creator develops a corresponding software identification tag to identify the software package.

**Example software identification tag creators include:**

a) *Software manufacturers.* A software identification tag may be created when a software manufacturer develops a particular software configuration item. Subsequently, the software identification tag and software installation routine are shipped together. Software may be dispatched directly to software consumers, publishers or both. Arrangements as to who creates the software identification tag and who defines the software configuration item's origins may be agreed to between the various parties. It is possible that this process may have mixed ownership as it depends on the channel used for software distribution. Software manufacturers will likely use digital signatures regularly within software.

b) *Software publishers.* When software is developed by one organization and then published by another, during the packaging process the software publisher will create a software identification tag to include as part of the product installation process.

c) *Line of business application developers.* Line-of-business application developers will also create software identification tags to include as part of the product installation process.

d) *Distributors, repackagers, value-added resellers and other tag modifying organizations.* Organizations that distribute software that does not include standardized software identification tags may want to add them in order to accommodate the needs of their software consumers. When a software package does not include a software identification tag, SAM practitioners are encouraged to create and include their own software identification tag to optimize SAM processes.

Software identification tag creators may also be third party organizations that are not directly related to the software creator. This may be the case for legacy software that does not have software identification tags, or for software that is developed by an organization that chooses not to include software identification tags with their products. In these cases, the tag creator shall define element values in the software identification tag that indicate that they are not the creator of the software (such as the tag_creator_regid in the software_id element). These cases will be evident because the tag_creator_regid will be different from the software_creator_regid.

### 6.2.3 Software identification tag modification

Individuals or companies that alter software identification tags and/or add supplemental information to software identification tags are considered tag modifiers. This group may include software aggregators, VAR's and may also include groups within an organization that manage software release processes.

**Example software identification tag modifiers may include:**

*a) Distributors*

*b) Resellers*

*c) Value-added resellers*

*d) Republishers*

*e) Packagers*

*f) Discovery tool providers*

*g) Deployment tool providers*

*h) Release managers*

### 6.2.4 Software identification tag use

Although software consumers are the ultimate beneficiaries of standardized software identification tag content, discovery tools and operating systems are the primary systems that use the data. They read software identification tag data to gather information about a given set of software configuration items. When a software configuration item is deposited onto a particular computing device, a well-formed software identification tag provides authoritative identification of the specific software installed, the tag creator and all third-party providers who may have modified the software. This information is invaluable to the SAM practitioner.

**Example software identification tag end-users include:**

*a) SAM owners*

*b) IT support professionals*

*c) Owners of the software configuration item*

### 6.2.5 Software identification tag correction

Although software identification tags should be managed along with all other components associated with a specific software installation, there will be times when a tag may be removed, deleted or corrupted – either on purpose or accidentally.

In the case where a software identification tag becomes corrupt, there are optional methods available for a software application to validate that the software identification tag is up-to-date and correct and/or potentially to automatically self-heal the software identification tag. These methods help to ensure a higher level of integrity of the data collected by discovery tools.

In the case where a product installation does not include a software identification tag, release managers may choose to create a software identification tag to provide with the software to ease the SAM reconciliation process.

In all cases, there should be methods available for a discovery agent or service to cross check information in a manner that provides assurance about the accuracy of the data. For example, a discovery agent may collect software identification tags and all executable filenames from a system. Software identification tags have the ability to specify all the files that are associated with a particular piece of software. Using an appropriate algorithm, a SAM discovery tool can quickly validate that a system with a specific software identification tag also includes the necessary files that are associated with that title. The reconciliation engine can then filter out the known application filenames from the collected list of executable filenames. Any filenames that remain in the list at the end of this process would be considered exceptions and may require that a SAM practitioner investigates these files further.



## 7 Platform requirements and guidance

### 7.1 Types of platforms

For the purposes of this part of ISO/IEC 19770, the term platform refers to a computer or hardware device and/or associated operating system, or virtual environment, on which software can be installed or run (4.1.17). The Linux™ operating system, for example, is used on a wide variety of hardware, from cell phone devices to mainframe computers, and each variation can be considered a separate platform for the purposes of this part of ISO/IEC 19770. Additional example platforms include, but are not limited to:

a) Redhat™ Linux™ Enterprise 4.0 Intel x86

b) Novell SuSE™ Linux™ 10.2 Intel x64

c) Macintosh™ OS 10.4 Intel x64

d) Microsoft Vista® x64

e) HP-UX 11i Itanium™

A platform can also be exemplified by a virtual environment (i.e., such as Java™ or .NET™) and, in such cases, hardware support is generally unimportant. Versioning, however, is of great consequence in virtual environments. Software written and compiled for one version of Java™ or .NET™, for example, may not run in a prior version's environment. It is expected that virtual environments will also provide a software identification tag to identify specific details about the version of the environment installed.

NOTE Do not confuse the terms virtual environment and virtual machine. The latter may run within a host operating system platform but still represents a complete operating system environment by itself. The virtual machine, supplied with virtual hardware, should therefore be treated as a separate hardware instance comparable to that of a separate physical machine.

## 7.2 Basic platform services

Platforms exist independently of the software identification tag details of software configuration items they contain and should be indifferent to them. A platform, however, should define processes to store and retrieve these software identification tags efficiently.

It is recommended that platforms store and retrieve software identification tags in a process similar to how operating systems manipulate files, the exception being that software identification tags should be stored in a central location for ease of discovery. If a centralized repository is unavailable for software identification tags, they should then be stored in a common location related to the software configuration items they define (as defined in section 6.1.4) as well as in each software package installation's top level directory. This means that discovery tools need to collect software identification tags that may be located in multiple directories (such as in the top level directory of a software package's installed files) in order to provide a complete inventory of tags.

A platform meeting the requirements of this part of ISO/IEC 19770 shall provide the following services:

a) *Basic add, modify, read and delete operations.*

b) *Audit capabilities*

   1) Identify who installed a given software configuration item and when installation occurred.

   2) Identify who modified a given software configuration item and when modification occurred.

   3) Identify who uninstalled a given software configuration item and when uninstallation occurred.

   NOTE There is no requirement to retain software identification tags when software is uninstalled. For consistency, it is recommended that they be removed. Audit trails should be used instead to identify previous installations, where this information is desired.

c) *Security*

   1) Determine who can create and modify software identification tags.

   2) Determine who can read software identification tags.

## 7.3 Virtual environments

Virtual environments are typically installed on a computing device and should provide their own software identification tags to identify themselves to discovery tools.

EXAMPLE When a Java™ Virtual Machine (JVM) is installed on a computing device, it should be installed with a software identification tag just as any other software package should.

The globally unique identifier and other identifier information for a virtual environment's software identification tag can then be utilized by other packages to identify a software configuration item's compatibility with the virtual environment.

EXAMPLE Once the JVM is installed, the identifier information from that package can be referenced by Java™-based applications that utilize the JVM. This should be done by the Java™-based application specifying that the JVM is a dependency.

## 7.4 Virtual machines

Virtual machines provide a guest operating environment that is independent of the host operating environment. As far as software is concerned, the installation and discovery process will generally be exactly the same, with the exception that the discovery process will occur completely within the virtual machine or within the virtual machines disk (which is often just a file on the host operating system). In these environments, the software identification tag should be provided just as it would be for any other computer.

It is expected that discovery tools will collect information about the system on which the tag is discovered as part of the discovery process. If the system is a virtual machine, the details about the virtual machine, its host environment, etc should be collected as well. The SAM reconciliation process then will use the details of the software identification tags that are collected, in addition to details about the environment the tag is installed on (virtual machine type, host for the virtual machines, etc.) and use these pieces of information to aid in reconciliation.

There are numerous virtualization technologies in use today and this part of ISO/IEC 19770 cannot provide definitions for how software identification tags should be used on all existing, or future technologies. Virtualization vendors, however, should be aware that application installation and use should be tracked and monitored in order to comply with software entitlements. As such, these virtualization technologies should provide a means of discovering software packages or applications that are available for use and/or are being used on a particular computing device. This may be done by providing software identification tags that are discoverable along with a virtualized environment, or may be provided through a discovery process on a virtualized disk.

## 7.5 Support for software installed on removable media

Software can often be installed on removable media. In these cases, the software identification tag needs to identify that the software was installed on a specific computing device as well as provide tag information that follows the removable media. This shall be done by providing a software identification tag in the common system location (either the OS defined tag store, or tags located in common directories) for the computing device. A second copy of the software identification tag shall be included in the top level directory used to install the software package.

Tool providers should recognize that two or more identification tags may be discovered on the same computing device and be able to recognize that the computing device only has one installation of a software package. In the instance where the removable media is discovered on another computing device, the software entitlement rules for that particular package shall specify if the package should be considered a single entity (the installation on the removable media), or two entities (the system the software was installed on as well as the actual software on the removable media).

## 7.6 Hardware and platform identification

This part of ISO/IEC 19770 is focused on software identification tags. In most use cases, these software identification tags will be collected as part of a standard SAM or software discovery/inventory process. The tool used to handle the discovery is expected to collect and return hardware and other platform information (such as operating system details) that will be associated with any software identification tags collected. This provides an automated association of software with the hardware and platform on which it is installed.

As portable devices become more and more capable, virtual environments and virtual machines become more diverse and automated data collection is improved, it is expected that additional hardware and platform identification information may be required to assist with IT asset management processes. Should an official, or

de-facto standard be defined that provides additional unique hardware and platform asset identification information; details specified in that standard should be utilized to develop a consistent installation_target_id that may be used on any platform. Should some type of hardware identification become an official standard, it is expected that information from that standard will be included in the conformance section of a future version of this part of ISO/IEC 19770.

# 8 Elements

## 8.1 General

Elements describe common attributes of all or most software configuration items. Operating systems and discovery tools can use these attributes to identify software configuration items.

Elements may be altered by an assortment of tag modifiers (6.2.3, 6.2.4).

This part of ISO/IEC 19770 enumerates 37 distinct elements, and this list is not exhaustive. They are predefined to ensure consistency between software identification tags (8.2).

Elements are described with examples in clauses 8.3 and 8.4, respectively. The examples are specified in XML syntax, the format which shall be used for software identification tag creation. The examples provide insight as to what information is to be included within data elements. For extended examples of software identification tags, please refer to Annex H.

This part of ISO/IEC 19770 does not require a specific process for generating content for elements.

Mandatory elements (8.3) are required for a software identification tag to be considered valid or complete. Incomplete software identification tags should be flagged by discovery tools as invalid, notifying the software identification tag data SAM practitioner that these are not valid or complete. There are five mandatory elements.

It is recommended that tag providers maintain a central repository of all software identification tags created for all product releases containing at least the mandatory elements. This repository can then be used to validate the uniqueness of GUIDs as well as ensuring that the software creator name and GUID remain consistent throughout the product line. This part of ISO/IEC 19770 does not require an external registration agency for software identification tags, so it is up to each tag creator to ensure each of their tags is unique.

Optional elements (8.4) may or may not be provided in a software identification tag. The data elements that correspond to optional elements permit software identification tag creators additional opportunities to improve reliability of information for SAM practitioners and tool providers. If these optional elements are used, then they shall be used in accordance with the requirements in this section.

Extended elements (8.5) are provided in the software identification tag to allow the inclusion of additional values that have not been predefined. Extended elements shall be in an XML format and should include an XSD reference that can be used to validate the information in this section.

Unlike the mandatory and optional sections, there may be multiple extended sections in a tag. Each extended section should be specific to a particular tag creator or modifier, and not have mixed usage. For example, a tag creator may include extended elements that it wants included for its own discovery tool. A software consumer organization may want to include extended elements related to their overall software lifecycle policies and procedures.

## 8.2 Element names

Software identification tag content shall be identified in accordance with the element names specified in clauses 8.3 and 8.4 below. This naming requirement ensures consistent interpretation of software identification tag content, regardless of mandatory or optional nature.

If an element is optional, its description still shall adhere to this naming requirement and to limitations specified in the definitions of clause 8.4.

## 8.3 Mandatory elements

### 8.3.1 Entitlement required indicator ('entitlement_required_indicator')

| | |
|---|---|
| **XML Tag** | entitlement_required_indicator |
| **Type** | Boolean |
| **Definition** | This element is a Boolean tag that indicates if a software entitlement must match up against this item in order for a software reconciliation to be considered successful. Open Source software, for example, may not "require" a software entitlement in the reconciliation process to be legally installed and used. This does not mean that the software does not have a software entitlement; rather that it does not need a software entitlement specified in a SAM system for the reconciliation to be complete. This provides the ability for a practitioner to manage by exception and focus only on those items that are legally required to be in compliance. This does not mean that an organization will not manage compliance of items such as open source, or freeware products, simply that they can make that choice.<br><br>This element shall occur exactly once in the software identification tag. |
| **Example** | <entitlement_required_indicator>true</entitlement_required_indicator> |

### 8.3.2 Product title ('product_title')

| | |
|---|---|
| **XML Tag** | product_title |
| **Type** | XML character string |
| **Definition** | Name of product, as assigned by the software creator. This value is primarily used in end-user or computing device focused reports and is not typically going to be used as part of the process of reconciliation.<br><br>This element shall occur exactly once in the software identification tag. |
| **Example** | <product_title>Viewmaster Standard</product_title> |

### 8.3.3 Product version ('product_version')

<table>
<tr><td>XML Tag</td><td colspan="3">product_version</td></tr>
<tr><td>Type</td><td colspan="3">Complex type</td></tr>
<tr><td>Definition</td><td colspan="3">Version of the product defined as two elements – numeric version and version name.<br><br>This element allows software creators to provide purely numeric version information which is used for comparison purposes against software entitlement information and for grouping purposes. Additionally, the String version is provided so software creators have the ability to specify any textual representation they want an end-user to see in a report.<br><br>A good example of this is Microsoft Vista®. Most SAM practitioners will immediately recognize a text version for this OS if it is listed as Microsoft Vista®, Version SP1. However, many SAM practitioners will not recognize the numeric version of 6001.18063.<br><br>Each element is independent, but they should be related and consistent with each other.<br><br>The numeric-based version number consists of four levels: major and minor version numbers plus build and maintenance numbers. If a vendor does not choose to use all available levels, the non-used levels should be set to 0. The numeric version is expected to be used for comparison purposes against software entitlement information during the reconciliation phase of the software asset management process.<br><br>The string version of the product version may contain numeric and/or alphabetic characters. It is a more user friendly name of the product version than numeric-based version number. This value will typically be used in end-user or computing device oriented reports.<br><br>This element shall occur exactly once in the software identification tag.</td></tr>
<tr><td rowspan="3">Data Structure</td><td>XML tag</td><td>Type</td><td>Definition</td></tr>
<tr><td>name</td><td>XML character string<br><br>One entry</td><td>Textual name of the version</td></tr>
<tr><td>numeric</td><td>ProductVersionComplexType - Complex type consisting of four elements with numeric values: “major”, “minor”, “build”, “review”<br><br>One entry</td><td>Numeric version identifier</td></tr>
<tr><td>Example</td><td colspan="3">
<pre>
<product_version>
   <name>10.2 Fix Pack 1</name>
   <numeric>
      <major>10</major>
      <minor>2</minor>
      <build>0</build>
      <review>0</review>
   </numeric>
</product_version>

OR

<product_version>
   <name>6.2.1279.00</name>
   <numeric>
      <major>6</major>
      <minor>2</minor>
      <build>1279</build>
      <review>0</review>
   </numeric>
</product_version>
</pre>
</td></tr>
</table>

### 8.3.4 Software creator identity ('software_creator')

| | | | |
|---|---|---|---|
| **XML Tag** | software_creator | | |
| **Type** | Complex type – EntityComplexType | | |
| **Definition** | This element allows a discovery process to identify the specific software creator that produced the software package.<br><br>Software creator names in different countries may be exactly the same, but refer to separate legal entities. To ensure uniqueness, this element shall contain the regid of the software creator as well as the name.<br><br>This element shall occur exactly once in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | name | XML character string<br><br>One entry | This element provides the name of the entity defined in the tag. This name should be consistent between software products and software releases. |
| | regid | regid type<br><br>One entry | Regid of the software creator (as specified in section 6.1.3.) If the entity is unknown, or is no longer in business, this value may be set to "unknown". |
| **Example** | <software_creator><br><name>Example Corp</name><br><regid>regid.1995-09.com.example</regid><br></software_creator> | | |

### 8.3.5 Software licensor identity ('software_licensor')

| | | | |
|---|---|---|---|
| **XML Tag** | software_licensor | | |
| **Type** | Complex type – EntityComplexType | | |
| **Definition** | This element allows a discovery process to identify the specific software licensor that owns the copyright for the software package.<br><br>Software licensor names in different countries may be exactly the same, but can refer to different legal entities. To ensure uniqueness, this element shall contain the regid of the software licensor as well as the name.<br><br>This element shall occur exactly once in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | name | XML character string<br><br>One entry | This element provides the name of the entity defined in the tag. This name should be consistent between software products and software releases. |
| | regid | XML character string<br><br>One entry | Regid of the software licensor (as specified in section 6.1.3.) If the entity is unknown, or is no longer in business, this value may be set to "unknown". |
| **Example** | <software_licensor><br><name>Example Corp</name><br><regid>regid.1995-09.com.example</regid><br></software_licensor> | | |

### 8.3.6 Software unique identifier ('software_id')

<table>
<tr><td>XML Tag</td><td colspan="3">software_id</td></tr>
<tr><td>Type</td><td colspan="3">Complex type</td></tr>
<tr><td>Definition</td><td colspan="3">The software_id provides information that can be used to reference a specific version of a specific product. This element requires the tag creator to ensure that the unique_id is unique for each software title and version. Different upgrade levels of a software package shall be distinguished by unique software identifiers. To avoid the need for an external registration agency, each tag creator must use their own regid as specified in section 6.1.3.<br><br>The regid is provided along with a unique ID (unique_id) within that regid. Different platforms and/or development environments may have different methods of creating unique IDs. The unique_id could be a GUID, or it may be simply a unique reference within the development environment. For example, an organization could decide their unique_id would be something like <productname>_<version>_<releaseID>.<br><br>It will be possible for multiple tag creators to create their own unique software_ids for the same software product. This is likely to be the case where the software creator did not create a software identification tag (such as for legacy software), and multiple competitive service organizations then create their own tags for use with such software.<br><br>This element shall occur exactly once in the software identification tag.</td></tr>
<tr><td rowspan="3">Data Structure</td><td>XML tag</td><td>Type</td><td>Definition</td></tr>
<tr><td>tag_creator_regid</td><td>XML character string<br><br>One entry</td><td>This element specifies the identification of the organization that created the tag.<br><br>Regid of the tag creator as specified section 6.1.3. Note that the tag_creator_regid and the software_creator_regid may be the same values – this will be the case where the software creator is creating the tags. Including the tag_creator_regid helps ensure uniqueness of the software_id and also allows SAM practitioners and SAM tools to identify the provenance of any discovered software identification tag.<br><br>This element shall not contain characters that are inconsistent with filename use such as '/', '\', '[', etc.</td></tr>
<tr><td>unique_id</td><td>XML character string<br><br>One entry</td><td>Unique ID that identifies the specific version of a specific product.<br><br>The unique_id shall follow the restrictions for URI character use as specified in IETF RFC 3986, section 2, Characters.<br><br>Additionally, this element shall not contain characters that are inconsistent with filename use such as '/', '\', '[', etc.</td></tr>
<tr><td>Example</td><td colspan="3"><software_id><br>  <unique_id>fc3cc419-b5a1-9f16-ed203e537c40</unique_id><br>  <tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id></td></tr>
</table>

| | |
|---|---|
| | OR<br><br><software_id><br><unique_id>com.adobe.Acrobat-3D-Win-Multilingual-8.00</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id> |

### 8.3.7 Tag creator identity ('tag_creator')

| | | | |
|---|---|---|---|
| **XML Tag** | tag_creator | | |
| **Type** | Complex type – EntityComplexType | | |
| **Definition** | This element allows a discovery process to identify the specific tag creator for the software package.<br><br>Tag creator names in different countries may be exactly the same, but refer to separate legal entities. To ensure uniqueness, this element shall contain the regid of the tag creator as well as the name.<br><br>This element shall occur exactly once in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | name | XML character string<br><br>One entry | This element provides the name of the entity defined in the tag. This name should be consistent between software products and software releases. |
| | regid | XML character string<br><br>One entry | Regid of the software licensor (as specified in section 6.1.3.) If the entity is unknown, or is no longer in business, this value may be set to "unknown". |
| **Example** | <tag_creator><br><name>Example Corp</name><br><regid>regid.1995-09.com.example</regid><br></tag_creator> | | |

## 8.4 Optional elements

### 8.4.1 Abstract ('abstract')

| XML Tag | abstract | | |
|---|---|---|---|
| Type | Complex type | | |
| Definition | Summary that provides information for the software package that this tag applies to, including a language component to allow for multi-language support.<br><br>The abstract element may occur more than once in a software identification tag, but shall only occur once for each language specified.<br><br>If language is not specified, it is assumed to be English ("en").<br><br>This element may occur zero to unlimited times in the software identification tag. | | |
| Data Structure | XML tag | Type | Definition |
| | lang | XML character string. This is an optional tag attribute | The language the abstract is written in. Languages shall be specified as defined in IETF RFC 4646 (see http://www.rfc-editor.org/rfc/rfc4646.txt). |
| Example | <abstract lang="en">The View Master software enables viewing of all kinds of document formats.</abstract> | | |

### 8.4.2 Component association ('component_of')

| XML Tag | component_of | | |
|---|---|---|---|
| Type | Complex type | | |
| Definition | Component_of is an element that is used to show a child to parent relationship between packages (i.e. which parent does this package belong to). Typically, this element will be used when additional components are installed, but are related to an existing package on a computing device. This element is not used as part of a suite definition, rather it is used when a package is installed that adds functionality to an existing package on the computing device.<br><br>Typically either component_of or complex_of will be used to specify a product grouping, not both at the same time.<br><br>This element may occur zero to one time in the software identification tag. | | |
| Data Structure | XML tag | Type | Definition |
| | software_id | Software identification type described in 8.3.5<br><br>One to unlimited entries | List of unique software identifiers that define an association between this package and a parent package. |

| | |
|---|---|
| **Example** | <component_of><br><software_id><br><unique_id>fc3cc419-b5a1-9f16-ed203e537c40</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id><br></component_of><br><br>**Or**<br><br><component_of><br><software_id><br><unique_id>com.adobe.Acrobat-3D-Win-Multilingual-8.00</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id><br></component_of> |

### 8.4.3 Components list ('complex_of')

| | | | |
|---|---|---|---|
| **XML Tag** | complex_of | | |
| **Type** | Complex type | | |
| **Definition** | Complex_of is an element that specifies child relationships for this package (i.e. which packages belong to this one). This element is typically used to provide a list of products that are a part of a "suite". This element is made up of a list of unique identifiers that represent the products that make up the suite. Typically either complex_of or component_of will be used to specify a product grouping, not both at the same time.<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | software_id | Software identification type described in 8.3.5<br><br>One to unlimited entries | Unique Software identifier that defines an association between this package and its child packages. This item is generally used when defining the packages that make up a suite. |
| **Example** | <complex_of><br><software_id><br><unique_id>fc3cc419-b5a1-9f16-ed203e537c40</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id><br><software_id><br><unique_id>a584c19-b5a1-9f16-ed203e5ab45fc</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id><br></complex_of><br><br>**Or**<br><br><complex_of><br><software_id><br><unique_id>com.adobe.Acrobat-3D-Win-Multilingual-8.00</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id><br></complex_of> | | |

### 8.4.4 Data source ('data_source')

| | |
|---|---|
| **XML Tag** | data_source |
| **Type** | XML character string |
| **Definition** | Basis for the data source of the final installation.<br><br>Values include, but are not limited to, strings such as the following: CD, MSDN CD, Electronic distribution and Definitive software library – Released for distribution. By providing such values in the software identification tagging process, SAM practitioners can rapidly assess what software configuration items are installed on which platforms and how each was installed. This element does not require normalization between different tag creators because many organizations have their own definitions for the type of data source and this information is informational to the SAM practitioner, and is not typically required.<br><br>Values used in this element should be consistent within an organization and across product lines.<br><br>This element may occur zero to one time in the software identification tag. |
| **Example** | <data_source>DVD</data_source> |

### 8.4.5 Dependency ('dependency')

| | | | |
|---|---|---|---|
| **XML Tag** | dependency | | |
| **Type** | Complex type | | |
| **Definition** | This element is provided in order to allow software to specify that it requires a different product in order to run. This is not necessarily related to software licensing or software entitlements, but simply to requirements. For example, a Java™ application may be dependent on a specific version of Java™ to run properly. An Excel® template requires Excel®. These dependencies are not necessarily strict dependencies that will be used by the software to validate that a specific software identification tag is available before the application runs, but rather guidance that is provided to the SAM database to assist with software relationships and/or potentially to provide information to a help desk environment.<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | software_id | Software identification type described in 8.3.5<br><br>One to unlimited entries | Unique Software identifier that defines the dependency tag. |
| **Example** | <dependency><br><software_id><br><unique_id>fc3cc419-b5a1-9f16-ed203e537c40</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></software_id><br></dependency> | | |

### 8.4.6 Element owner list ('elements_owner')

<table>
<tr><td>XML Tag</td><td colspan="3">elements_owner</td></tr>
<tr><td>Type</td><td colspan="3">Complex type</td></tr>
<tr><td>Definition</td><td colspan="3">This element provides the ability to specify who claims ownership over elements in the tag. This ownership claim is not as authoritative as using a digital signature, however, it does provide guidance to tag modifiers. By listing an element as being "owned", it indicates that the value specified in the element should not be changed unless explicitly agreed to by the existing owner of the element.<br><br>The tag creator should specify the elements that may not be modified by any tag modifier.<br><br>The element utilizes element IDs which the XSD supports. These element IDs are created by the tag creator and need only be uniquely specified and referenced for each software identification tag. IDs are utilized to reference specific other tag elements from within the tag itself. See the examples below as well as in section 6.1.12 to see how IDs are used.<br><br>This element may occur zero to unlimited times in the software identification tag.</td></tr>
<tr><td>Data Structure</td><td>XML tag</td><td>Type</td><td>Definition</td></tr>
<tr><td></td><td>owner_regid</td><td>XML character string<br><br>one entry</td><td>If the elements_owner is specified, this item specifies the company that owns the values. In general, commercial, off-the-shelf (COTS) software will use the regid as the full definition of the owner. Internally built applications may also specify the owner_name.</td></tr>
<tr><td></td><td>owner_name</td><td>XML character string<br><br>Zero or one entry</td><td>This element provides more detail on who owns the values specified in the software identification tag. In general, the owner_name will only be used for internally built applications where an individual or group also needs to be specified to know who should be contacted. If desired, commercial applications may also use the owner_name element to specify additional contact details related to the owned elements.</td></tr>
<tr><td></td><td>element_id</td><td>XML character string<br><br>Zero to unlimited entries</td><td>Element_id provides a list of IDs that are owned by the specified owner. IDs are specified by the element creator and must be unique for a specific tag, but there is no requirement that they be unique between different tags since they are only used to show relationship links within the tag, or to reference specific elements within a specific tag. See the example for how IDs may be specified and used by the elements_owner structure.</td></tr>
<tr><td>Example</td><td colspan="3"><swid:entitlement_required_indicator ID="e8_3_1">true</swid:entitlement_required_indicator><br><swid:product_title ID="e8_3_2">Adobe Photoshop CS3</swid:product_title><br><swid:product_version ID="e8_3_3"><br><swid:name ID="e8_3_3sub1">10.2</swid:name><br><swid:numeric ID="87_3_3sub2"><br><swid:major ID="e8_3_3sub2sub1">10</swid:major><br><swid:minor ID="e8_3_3sub2sub2">2</swid:minor><br><swid:build ID="e8_3_3sub2sub3">0</swid:build><br><swid:review ID="e8_3_3sub2sub4">0</swid:review><br></swid:numeric><br></swid:product_version><br><swid:software_creator ID="e8_3_4"><br><swid:name>Adobe Systems Incorporated</swid:name></td></tr>
</table>

```
  <swid:regid>regid.1986-12.com.adobe</swid:regid>
</swid:software_creator>
<swid:software_licensor ID="e8_3_5">
  <swid:name>Adobe Systems Incorporated</swid:name>
  <swid:regid>regid.1986-12.com.adobe</swid:regid>
</swid:software_licensor>
<swid:software_id ID="e8_3_6">
  <swid:unique_id>Photoshop-CS3-Win-GM-en_US</swid:unique_id>
  <swid:tag_creator_regid>regid.1986-12.com.adobe</swid:tag_creator_regid>
</swid:software_id>
<swid:tag_creator ID="e8_3_7">
  <swid:name>Adobe Systems Incorporated</swid:name>
  <swid:regid>regid.1986-12.com.adobe</swid:regid>
</swid:tag_creator>

<!-- Optional elements -->

<swid:elements_owner>
  <swid:owner_name>Adobe Systems Inc. </swid:owner_name>
  <swid:owner_regid>regid.1986-12.com.adobe</swid:owner_regid>
  <swid:elements_ID>e8_3_1</swid:elements_ID>
  <swid:elements_ID>e8_3_2</swid:elements_ID>
  <swid:elements_ID>e8_3_3</swid:elements_ID>
  <swid:elements_ID>e8_3_4</swid:elements_ID>
  <swid:elements_ID>e8_3_5</swid:elements_ID>
  <swid:elements_ID>e8_3_6</swid:elements_ID>
  <swid:elements_ID>e8_3_7</swid:elements_ID>
</swid:elements_owner>
```

### 8.4.7 Installation details ('installation_details')

| XML Tag | installation_details |
|---|---|
| Type | Complex type |
| Definition | This element provides specific details for the full path information on the locations of the software identification tags for a particular software package installation as well as installation instance details. Each software installation will have two software identification tags added to the system – one in the common platform directory (as specified in section 6.1.4) and one in the root directory of the installed software package.<br><br>It is strongly recommended that the tag locations be included whenever possible since this allows SAM tool providers to link two software identification tags together as a related tag.<br><br>On platforms where software products are allowed to be moved easily (such as the Apple Macintosh™ platform), it is highly recommended that the software application regularly validate that the installation_details element is defined properly and that SAM tool providers validate the location where the tags are discovered and compare that to the installation_details.<br><br>Installation instances are provided for software that may be installed multiple times on a single platform. This may be done in the case of installations for specific end-users, or it may be done to provide multiple copies of a particular software package for an end-user, or on the system in general.<br><br>This element may occur zero to one time in the software identification tag. |

| Data Structure | XML tag | Type | Definition |
|---|---|---|---|
| | location_platform | XML character string<br><br>Zero or one entry | This is the full path to the directory where the common software identification tag is located. If the common path is in a system specified location (such as in the case of Microsoft Vista®), then "system" may be specified. |
| | location_installation | XML character string<br><br>Zero or one entry | This is the full path to the software identification tag that is installed in the root directory of the software package. |
| | installation_instance | XML character string<br><br>Zero or one entry | If a software title allows multiple installations, the installation_instance allows organizations to provide a unique identifier for each installation.<br><br>For example, if multiple instances can be created for individual end-users, each installation might be identified by the end-user id. If the software may be installed multiple times for the same end-user or the system, this identifier may simply be a number that is incremented as other software identification tags are discovered during installation.<br><br>This element shall not contain characters that are inconsistent with filename specifications such as '/', '\', '[', etc. |
| | installation_locale | XML character string<br><br>zero to unlimited times | This element specifies the locale or locales supported by the installed software. Locales shall be specified as defined in IETF RFC 4646 (see http://www.rfc-editor.org/rfc/rfc4646.txt). If the installed version of software supports multiple locales, this can be identified by the software identification tag containing multiple installation_locale elements. |
| | installation_target_id | XML character string<br><br>Zero or one entry | A value which will allow the identification of the machine, storage device, and/or virtual environment on which software has been installed.<br><br>This element shall not contain characters that are inconsistent with filename specifications such as '/', '\', '[', etc.<br><br>The software installer (or self-installing programs) should provide for installation-specific parameters which determine the value to be put into this element. A specific value may be given in a parameter, or there may be a specification of the operating system call to use to obtain the required information. In the absence of any other specification by the software publisher, a default value shall be used, with the media serial number (of the media onto which the software has been installed) being the recommended default value. |

| | | | |
|---|---|---|---|
| | | | The installation_target_id may include asset number, media serial number for the media on which the software is installed, a NIC MAC address, or other unique reference items.<br><br>It is recognized that some of these values can change without the underlying identity truly changing, e.g. a network interface card may need to be changed which will change the NIC MAC address. Software asset management procedures will need to deal with such issues.<br><br>This value is a recommended component of the installed version of the filename for a software identification tag. (See 6.1.7)<br><br>It is anticipated that conventions will evolve as to what types of values will typically be placed into this element, and that these conventions will be supported by platform providers. |
| **Exam-ple** | <installation_details><br><location_platform>C:\Documents and Settings\All Users\Application Data \adobe\adobe.com-photoshop8.0pro.swidtag</location_platform><br><location_installation>C:\Program Files\Adobe\Photoshop CS\adobe.com-photoshop8.0pro.swidtag</location_installation><br><installation_instance>1<\installation_instance><br><installation_target_id>0018F8096CE1</installation_target_id><br><installation_locale>en-US</installation_locale><br><installation_locale>en-GB</installation_locale><br><installation_locale>en-AU</installation_locale><br></installation_details> | | |

### 8.4.8 Keywords ('keywords')

| | |
|---|---|
| **XML Tag** | Keywords |
| **Type** | Complex type |
| **Definition** | This element provides the ability for a tag creator or modifier to add specific keywords to the software identification tag. The keyword values are not specified in this part of ISO/IEC 19770, but are instead provided as a way for the tag creator or modifier to help search engines find software identification tags that relate to a particular subject.<br><br>Individual keywords may be added by tag modifiers, they may be specified in the "elements_owner" element and they may also be signed. Since the keywords element allows multiple keyword sub-elements, each sub-element may be owned or signed by its individual owner. |

| Data Structure | XML tag | Type | Definition |
|---|---|---|---|
| | Keyword | XML character string<br><br>Zero to unlimited entries | This is the element used to specify keywords that are applicable to a specific software identification tag. Keywords are entered one at a time, and may be identified as being owned (by elements_owner - 8.4.6) and may be included in a digital signature as well (see Authenticity of software identification tags 6.1.13).<br><br>Multiple tag modifiers may add their own keywords to the list as well. |
| **Example** | <keywords><br><keyword>Acme</keyword><br><keyword>Painter</keyword><br></keywords> | | |

### 8.4.9 License and channel information ('license_linkage')

| XML Tag | license_linkage | | |
|---|---|---|---|
| **Type** | Complex type | | |
| **Definition** | This element provides information that can be used to help determine the proper software entitlement structure for the product installation that is related to this tag. The elements that are part of the license_linkage element provide information on how the product may have been installed and its current license state on the particular system the tag is discovered on.<br><br>NOTE License state is not directly related to a software entitlement. License state is for an installation of a specific software package on a specific machine. Entitlements, on the other hand, specify the legal ownership of license use rights. Entitlement details are specified in part 3 of ISO/IEC 19770.<br><br>Elements provided as part of the license_linkage element can help SAM practitioners quickly identify when un-authorized software is installed in their environment. These tags are optional, but they will help the SAM practitioner by providing more information about where software may have come from. By providing these tags, SAM practitioners can build rules that help them manage by exception rather than having to monitor each and every change that may occur in the environment they work in.<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | activation_status | XML character string<br><br>Zero to unlimited entries | The values in this element are related to the various licensing levels that a specific software licensor may track for an individual machine. Every software licensor may have a different set of status values, but as much as possible, the values should be consistent for one software licensor. A representation of these values may include:<br><br>a) Trial – this indicates that the software is in a trial mode and this value may include the number of days the trial mode is valid, or that the trial has expired.<br><br>b) Serialized – this indicates that the end-user or |

| | | | |
|---|---|---|---|
| | | | software consumer has entered a valid serial number during the installation process, but that the product has not yet been activated.<br><br>c) Fully Licensed – this indicates that the product has been activated and as far as the software licensor is concerned, this is a fully capable installation for a specific system.<br><br>d) Unlicensed – this indicates that the software should no longer be able to be run on a device, or it is running in a limited mode. Software can get into this state by the following:<br><br>1) A trial period has expired<br><br>2) A time-based license has expired<br><br>3) Package was serialized, but never activated in the given timeframe.<br><br>Values used in this element should be consistent within an organization and across product lines. |
| | channel_type | XML character string<br><br>Zero to unlimited entries | Provides information on which channel this particular software was targeted for. The values used in this element may be unique to the software vendor, but should be consistent between products published by a particular vendor. A representation of these values may include:<br><br>a) Volume<br><br>b) Retail<br><br>c) OEM<br><br>d) Academic<br><br>If used by a software licensor, it allows a SAM practitioner to identify software that may be installed in an organizations environment but that doesn't follow organizational policy. For example, software that was destined for an academic channel is not generally considered appropriate for installation in a corporate setting.<br><br>Values used in this element should be consistent within an organization and across product lines. |
| | channel_name | XML character string<br><br>Zero or one entry | This element provides a location for the name of the channel. This allows reseller organizations to create software identification tags that include the name of a distribution or channel partner. |
| | customer_type | XML character string<br><br>Zero to unlimited | Customer type identifies the target customer, not the channel. The values used in this element may be unique to the software vendor, but should be consistent between products published by a particular vendor. A representation of these values may include: |

| | | | |
|---|---|---|---|
| | | entries | a) Government<br><br>b) Corporate<br><br>c) Educational<br><br>d) Retail<br><br>With current software entitlements, there is often a different cost associated with products that are targeted at different customers. However, there are also limitations placed on the installation of these products. For example, a copy of software that is created for an educational customer is typically sold at a lower cost and often not licensed for use in a corporate setting.<br><br>Values used in this element should be consistent within an organization and across product lines. |
| **Example** | <license_linkage><br><activation_status>Fully Licensed</activation_status><br><channel_type>Volume</channel_type><br><channel_name>Reseller name</channel_name><br><customer_type>Corporate</customer_type><br></license_linkage><br><br>**Or**<br><br><license_linkage><br><activation_status>Trial</activation_status><br><channel_type>Retail</channel_type><br><customer_type>Retail</customer_type><br></license_linkage> | | |

### 8.4.10 Package footprint ('package_footprint')

| | |
|---|---|
| **XML Tag** | package_footprint |
| **Type** | Complex type |
| **Definition** | Specifies a set of files and other entries that indicate a product is installed. Also provides for a link to a package_footprint from an external URI. On the Windows® platform, other entries may include registry entries, WMI entries, and MSI data. Other platforms may include additional platform specific information that may be desirable to include.<br><br>Package_footprint information will be used by SAM tools and SAM practitioners to provide confidence levels on software identification tag information and cross reference details that a software creator indicates should be present against what is actually discovered during an inventory process. Note that the package_footprint is not intended to validate that a software installation is complete, nor that the software will actually run.<br><br>The items listed in the primary section can be used to validate that a software identification tag is installed on a device that does, in fact, have the software installed. Secondary and other items are provided by tag creator to ease the burden of SAM tools and SAM practitioners. By providing footprint information for an application, SAM tools and SAM practitioners can use this information to filter out the extensive list of discovered data they receive from all devices and can move towards an exception based SAM management practice as they have a more accurate list of what is authorized to be installed as opposed to |

| | | | |
|---|---|---|---|
| | what is discovered.<br><br>NOTE Examples of the various elements can be found in section 8.6.6 FootprintModuleComplexType which provides details on how the various types and structure of the type used in the package_footprint element work together.<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | external_description | URI string<br><br>Zero or one entry | This element allows a tag creator to keep the list of entities associated with a particular system available through an indirect reference to their site. By doing this, the tag creator owns the list and can update the list as necessary without having to distribute a patch, or a new version of the product with a new set of files listed. |
| | primary | FootprintModuleComplex type<br><br>Zero to one entry | Defines files and other items that are considered "primary" to a software package. If items in the primary element are found on a device, then software is considered to have a high probability of being installed and the tag should be considered to be valid in that the software to which the tag refers has a high likelihood of being installed on the device.<br><br>NOTE 1 A filename by itself is not unique. To ensure uniqueness, this element provides multiple characteristics that can be defined including: name, size, md5, version and "other" types that can be tag creator defined.<br><br>NOTE 2 There may be multiple entries for a particular file in the primary element. This allows a single footprint to be used for multiple patch releases where files may change size, version or MD5 sums. In these cases, the discovery tool only needs to discover a single unique file definition for each unique filename presented (i.e. if file "abc.com" has 3 different sizes and MD5 entries, and the SAM tool matches one discovered file against one of those 3 entries, than file "abc.com" is defined as existing on the device).<br><br>NOTE 3 If one version of each primary item (file, as well as os_configuration_record or other element in the FootPrintModuleComplex type) defined in the primary element is present, then SAM tools and SAM practitioners should have a high confidence that the software identification tag is properly identifying installed software. If some primary items are not discovered on a device, this would indicate that an exception has occurred and the SAM practitioner or release manager should investigate why the device does not have all primary items. |
| | secondary | FootprintModuleComplex type<br><br>Zero to one entry | Defines files and other items that are considered "secondary" to a software package. These items are not used to validate that the tag refers to software that is, in fact installed, but are instead provided so they can be used as a filter by software recognition algorithms that determine if a software package is installed based on files found. By providing a list of files that can be safely "filtered out", the software recognition engine will end up with many fewer unmatched files requiring research. |

| | | | |
|---|---|---|---|
| | | | Similar to the primary element above, files may include other specific characteristics to ensure uniqueness to the tag creator. |
| | related | FootprintModule Complex type<br><br>Zero to one entry | Defines files and other items that are "loosely coupled" to a software package, but that may also be installed with a particular software package. These file entries are also used to filter a complete list of files that a discovery agent may collect in order to remove files that are associated with "known" software.<br><br>An example of this is an installer that installs an Original Equipment Manufacturer (OEM) version of a software package. The files in the OEM installation should be able to be loosely associated with the primary package (so they can be appropriately filtered by a software recognition process), but if another software package claims ownership to those files, then the software claiming ownership gets precedence over any software that has a loose association. Other loose couplings would include components that are shared between software packages and/or components that may be installed as optional add-ons to a software package. |

**Example**

```
<package_footprint>
      <external_description>http://www.adobe.com/acrobat/(sofware_id)/filelist.xml</external_descriptio
n>
</package_footprint>

Or

<package_footprint>
      <primary>
            <file>
                  <name>acrobat.exe</name>
                  <size>349808</size>
                  <version>8.1.0.137</version>
                  <md5>9b67d80f752d9aba168f7795a5ffa7f6</md5>
            </file>
            <os_configuration_record>
                  <record_type>WMI</record_type>
                  <path>Root\CIMV2</path>
                  <name>Win32_Product</name>
                  <internal_path>Name= 'Adobe Acrobat 8 Professional'</internal_path>
                  <entry>
                              <name>Version</name>
                              <value>8.1.2</value>
                              <type></type>
                  </entry>
                  <entry>
                              <name>Vendor</name>
                              <value>Adobe Systems</value>
                              <type></type>
                   </entry>
            </os_configuration_record>
      </primary>
</package_footprint>

NOTE    The filelist.xml file would use the same structure as the software identification tag and the
"package_footprint" element would be used to provide the file definitions for a particular software_id
```

### 8.4.11 Packager ('packager')

<table>
<tr><td>XML Tag</td><td colspan="3">packager</td></tr>
<tr><td>Type</td><td colspan="3">Complex type</td></tr>
<tr><td>Definition</td><td colspan="3">Provides details of who modified a software package for a particular set of installation procedures. This element will most often be specified by a release manager within an organization as software is configured for installation within that company. In these cases, the packager element will often be associated with details such as release_id, and release_package.<br><br>The element may also be used by third-parties in cases where a product is OEM'd and repackaged, or if the software package is configured to install with a specific configuration.<br><br>This element may occur zero to one time in the software identification tag.</td></tr>
<tr><td rowspan="3">Data Structure</td><td>XML tag</td><td>Type</td><td>Definition</td></tr>
<tr><td>by</td><td>XML character string<br><br>One entry</td><td>Attribute which gives information about third party packaging product company. Examples of the types of values that could be beneficial here:<br><br>a) Packaging company name<br><br>b) Packaging technology used<br><br>c) Internal group name of the packager<br><br>d) Desktop management product used<br><br>Additional information on this item may be available on the web site: http://standards.iso.org/19770.</td></tr>
<tr><td>part</td><td>XML character string<br><br>One entry</td><td>Additional information about third party product reference details such as part number.</td></tr>
<tr><td>Example</td><td colspan="3"><packager><br><by>ACME Widget Corp</by><br><part>Photoshop CS3 – OEM'd into widget designer – P#345ABD</part><br></packager></td></tr>
</table>

### 8.4.12 Product category ('product_category')

| XML Tag | product_category | | |
|---|---|---|---|
| Type | Complex type | | |
| Definition | Means by which product titles are classified by high-level function. A standardized list of categories/groups is provided by the United Nations Standard Products and Services Code: UNSPSC (for more information see http://www.unspsc.org/), COMMODITY listing number 43230000). UNSPSC codes found in the section numbered 43230000 of the specification are where the bulk of commonly used software categories will be found. Product categorization shall be done using the UNSPSC codes.<br><br>This element may occur zero to one time in the software identification tag. | | |
| Data Structure | **XML tag** | **Type** | **Definition** |
| | UNSPSC_ver | XML character string<br><br>One entry | Version number of the UNSPSC code set used. The version is not required to use the code, however, if a tool uses the version to provide additional functionality (such as providing various names in one of 9 other languages), the version will be needed by the tool.<br><br>An example of the format of the UNSPSC versions is 10.0501. |
| | segment_title | XML character string<br><br>One entry | Name of the segment the product belongs to |
| | family_title | XML character string<br><br>One entry | Name enabling recognition of the family of the product |
| | class_title | XML character string<br><br>One entry | Name of the class |
| | commodity_title | XML character string<br><br>One entry | Name of the commodity |
| | code | Numerical value with 8 digits<br><br>One entry | Codes shall be specified as defined in the **UNSPSC** code list. |
| Example | <category><br><UNSPSC_ver>10.0501</UNSPSC_ver><br><segment_title>Information Technology Broadcasting and Telecommunications</segment_title><br><family_title>software</family_title><br><class_title>Finance accounting and enterprise resource planning ERP software</class_title><br><commodity_title>Enterprise resource planning ERP software</commodity_title><br><code>43231602</code><br></category> | | |

### 8.4.13 Product family ('product_family')

| | |
|---|---|
| **XML Tag** | product_family |
| **Type** | XML character string |
| **Definition** | Product family provides an element software publishers and software licensors can use to group related software products together for SAM practitioner reports. An example of the type of product that would use this element is a backup tool where the backup services, server backup and client backup portions for the tool are sold as independent products. In this case, if all products have the same product_family defined, a SAM tool can automatically group discovered software identification tag data appropriately as shown below:<br><br>Example Backup Utility<br>Backup Server - 20 installations discovered<br>Server Backup Utility – 240 installations discovered<br>Client Backup Utility – 10,240 installations discovered |
| **Example** | <product_family>Example Backup Utility</product_family> |

### 8.4.14 Product identifier ('product_id')

| | |
|---|---|
| **XML Tag** | product_id |
| **Type** | XML character string |
| **Definition** | Identification of the product. It is independent from its version.<br><br>Product_id should be a unique reference, but this can be unique within the software manufacture and does not need to be a globally unique ID.<br><br>It is recommended that the Product ID not be the product name, or other marketing term as these often change from release to release. Instead the product_id should be an identifier that can follow products through their lifecycle without requiring marketing changes.<br><br>Product_id is used to define a lineage between products for identification of allowed upgrades. This value may or may not be used by a software entitlement. If a software entitlement specifies that a product may allow upgrades during a certain period of time, the software entitlement document cannot know which future product names or product versions can be applied and will become available during that time. The product_id allows a software entitlement document to specify that a specific version of the product is entitled to be installed initially, and any updated products that have the same product_id are also entitled to be installed as long as the release_date falls within the range provided in the software entitlement.<br><br>NOTE There may be more than one entry for product_id. This may happen in the case where a creator comes out with a new product and allows end-users or software consumers using different older products to upgrade to the new one. For example:<br><br>Product A<br>product_id = 1234XYZ<br><br>Product B<br>product_id = ABCDPDQ<br><br>Product C (this product allows maintenance upgrades from Product A or Product B)<br>product_id = 9876HJK <- this is the new product ID for Product C…<br>product_id = 1234XYZ |

| | |
|---|---|
| | product_id = ABCDPDQ<br><br>If later releases of Product C only wanted to allow maintenance upgrades from earlier versions of Product C, the product_id would only include the new ID for that product – 9876HJK.<br><br>This element may occur zero to unlimited times in the software identification tag |
| **Example** | <product_id>fc3cc419-b5a1-9f16-ed203e537c40</product_id> |

### 8.4.15 Release date ('release_date')

| | |
|---|---|
| **XML Tag** | release_date |
| **Type** | XML dateTime type |
| **Definition** | This tag will typically be used by a software consumer organization as part of an ITIL release process.<br><br>Date software configuration item was released for installation. The software configuration item should use a single date of release in order to facilitate reconciliation.<br><br>This element may occur zero to one time in the software identification tag. |
| **Example** | <release_date>2008-01-21T12:00:00</release_date> |

### 8.4.16 Release identifier ('release_id')

| | |
|---|---|
| **XML Tag** | release_id |
| **Type** | XML character string |
| **Definition** | This tag will typically be used by a software consumer organization as part of an ITIL release process.<br><br>Data used in reconciliation to identify release package attributes upon installation and associated software entitlements. Entries for this element shall be kept consistent across all software identification tags for any given software configuration item.<br><br>This element may occur zero to one time in the software identification tag. |
| **Example** | <release_id>COE-Base-Ver 8, 2008-01-21</release_id> |

### 8.4.17 Release package ('release_package')

| | | | |
|---|---|---|---|
| **XML Tag** | release_package | | |
| **Type** | Complex type | | |
| **Definition** | This tag will typically be used by a software consumer organization as part of an ITIL release process.<br><br>Validation information that a release package has been built to conform to the service provider's systems architecture, service management and infrastructure specifications.<br><br>NOTE End-use software packages will almost always be customized to the needs of the service provider, with specific installation options and/or combinations of software bundling specified. (Release software identification tags are completely independent of any external software provider software identification tag).<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | Sign_off | XML character string<br><br>One entry | This entry indicates the person who authorized that the software was packaged properly and is ready to go into a testing phase. |
| | Sign_off_date | XML dateTime type<br><br>One entry | This entry indicates the date the software package was signed off. |
| | By | XML character string<br><br>One entry | This entry indicates the software developer who created the package. This information may be used if questions come up during the testing phase. |
| **Example** | <release_package><br><sign_off>Jane Doe</sign_off><br><sign_off_date>2008-01-10T12:00:00</sign_off_date><br><by>John Doe</by><br></release_package> | | |

### 8.4.18 Release rollout ('release_rollout')

| | |
|---|---|
| **XML Tag** | release_rollout |
| **Type** | Complex type |
| **Definition** | Validation information relevant to who signed off a release package as ready for production use and when the sign off occurred.<br><br>This element may occur zero to one time in the software identification tag. |

| Data Structure | XML tag | Type | Definition |
|---|---|---|---|
| | Sign_off | XML character string<br><br>One entry | This entry indicates the person who authorized that the software was properly tested in a production pilot and is ready to go into a production use. |
| | Sign_off _date | XML dateTime type<br><br>One entry | This entry indicates the date the software pilot was signed off. |
| | By | XML character string<br><br>One entry | This entry indicates the SAM practitioner who managed the pilot testing phase. This information may be used if questions come up once the software is in production. |
| **Example** | <release_rollout><br><sign_off>Mary Jane</sign_off><br><sign_off_date>2008-01-16T12:00:00</sign_off_date><br><by>John Smith</by><br></release_rollout> | | |

### 8.4.19 Release verification ('release_verification')

| XML Tag | release_verification | | |
|---|---|---|---|
| **Type** | Complex type | | |
| **Definition** | Validation information that a release package has been verified against a testing environment that matches the requirements of the target production environment.<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | Sign_off | XML character string<br><br>One entry | This entry indicates the person who authorized that the software was properly tested in a controlled environment and is ready to go into a pilot testing. |
| | Sign_off_date | XML dateTime type<br><br>One entry | This entry indicates the date the software testing was signed off. |
| | By | XML character string<br><br>One entry | This entry indicates the SAM practitioner who managed the controlled testing phase. This information may be used if questions come up once the software is in the pilot testing phase. |
| **Example** | <release_verification><br><sign_off>Jane Smith</sign_off><br><sign_off_date>2008-01-14T12:00:00</sign_off_date><br><by>Doug Johnson</by><br></release_verification> | | |

### 8.4.20 Serial number ('serial_number')

| XML Tag | serial_number |
|---|---|
| Type | XML character string |
| Definition | Unique identifying number; may be represented as a combination of numbers, letters or symbols. Serial Number is a commonly used unique number assigned for identification of a particular title and purchase. In the case of software identification tags, the unique_id becomes the primary unique key, but many organizations may still want to use the serial number where it is available.<br><br>NOTE 1 The serial number may be put through a one way hash that obfuscates the actual serial number – this is still useful to the SAM practitioner – especially if the same reference serial number is included on the purchase order, invoice or other details provided by the distributor to the software consumer.<br><br>NOTE 2 If the tag creator chooses not to provide a serial number, they may choose to provide some other referencable data value that may be used to associate information in purchase orders. This allows a tag creator to assist SAM providers in finding entitlement information.<br><br>This element may occur zero to one time in the software identification tag. |
| Example | <serial_number>1088-9015-2034-4567</serial_number><br><br>**Or**<br><br><serial_number>10PQR28FTQN2008</serial_number> |

### 8.4.21 SKU ('sku')

| XML Tag | sku |
|---|---|
| Type | XML character string |
| Definition | A Stock Keeping Unit (SKU) is a unique identifying number for a software provider. The SKU may be represented as a combination of numbers, letters or symbols. SKU is a commonly used unique number assigned for identification of a particular title and purchase. In the case of software identification tags, the unique_id becomes the primary unique key, but many organizations may still want to have direct access to the SKU value.<br><br>NOTE If the tag creator chooses not to provide a SKU, they may choose to provide some other referencable data value that may be used to associate information in purchase orders. This allows a tag creator to assist SAM providers in finding entitlement information.<br><br>This element may occur zero to one time in the software identification tag. |
| Example | <sku>065-04940</sku> |

### 8.4.22 Software creator alias ('software_creator_alias')

<table>
<tr><td>XML Tag</td><td colspan="3">software_creator_alias</td></tr>
<tr><td>Type</td><td colspan="3">Complex type – EntityDataComplexType</td></tr>
<tr><td>Definition</td><td colspan="3">Provides additional software creator information enabling SAM practitioners and SAM tool providers to identify previous entities who were related to the creation of the software identified in the tag. Though not strictly required for software discovery purposes, this entry will ease the burden of a SAM practitioner by providing them with previous software creator details which can be used to more easily find an older software entitlement. This is especially important in the case where an upgrade is allowed from a previous software provider's version of a product to the current provider's version.<br><br>This element may occur zero to one time in the software identification tag.</td></tr>
<tr><td>Data Structure</td><td>XML tag</td><td>Type</td><td>Definition</td></tr>
<tr><td></td><td>alias</td><td>Complextype - AliasDetailsComplexType<br><br>Zero to unlimited entries</td><td>Details of previous creators who may have a relationship to the software title identified by the software identification tag.</td></tr>
<tr><td>Example</td><td colspan="3">The following example is appropriate for a Macrovision product that was purchased by and is now owned by Adobe®<br><br><software_creator_alias><br><alias><br><alias_name>Macrovision</alias_name><br><alias_regid>regid.1998-02.com.macrovision</alias_regid><br></alias><br></software_creator_alias><br><br>Or, if the regid of the alias entity is unknown:<br><br><software_creator_alias><br><alias><br><alias_name>Macrovision</alias_name><br><alias_regid>unknown</alias_regid><br></alias><br></software_creator_alias></td></tr>
</table>

### 8.4.23 Software licensor alias ('software_licensor_alias')

<table>
<tr><td>XML Tag</td><td colspan="3">software_licensor_alias</td></tr>
<tr><td>Type</td><td colspan="3">Complex type – EntityDataComplexType</td></tr>
<tr><td>Definition</td><td colspan="3">Provides additional software licensor information enabling SAM practitioners and SAM tool providers to identify previous entities who were related to the licensing of the software identified in the tag. Though not strictly required for software discovery purposes, this entry will ease the burden of a SAM practitioner by providing them with previous software licensor details which can be used to more easily find an older software entitlement. This is especially important in the case where an upgrade is allowed from a previous software provider's version of a product to the current provider's version.<br><br>This element may occur zero to one time in the software identification tag.</td></tr>
<tr><td rowspan="2">Data Structure</td><td>XML tag</td><td>Type</td><td>Definition</td></tr>
<tr><td>alias</td><td>Complextype - AliasDetailsComplexType<br><br>Zero to unlimited entries</td><td>Details of previous licensors who may have a relationship to the software title identified by the software identification tag.</td></tr>
<tr><td>Example</td><td colspan="3">
<pre>
<software_licensor_alias>
  <alias>
    <alias_name>Adobe Systems</alias_name>
    <alias_regid>regid.1986-12.com.adobe</alias_regid>
  </alias>
</software_licensor_alias>
</pre>
Or, if the regid of the alias entity is unknown:
<pre>
<software_licensor_alias>
  <alias>
    <alias_name>Adobe Systems</alias_name>
    <alias_regid>unknown</alias_regid>
  </alias>
</software_licensor_alias>
</pre>
</td></tr>
</table>

### 8.4.24 Supported languages ('supported_languages')

| | | | |
|---|---|---|---|
| **XML Tag** | supported_languages | | |
| **Type** | Languages as specified in IETF RFC 4646 | | |
| **Definition** | Languages that the program interface presents to the user. Languages shall be specified as defined in IETF RFC 4646.<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | Language | XML character string<br><br>One to unlimited entries | Languages supported by this software package. Language may occur multiple times. Specification of the language shall be specified as defined by IETF RFC 4646 (see http://www.ietf.org/rfc/rfc4646.txt) and the process used for matching language tags is specified in IETF RFC 4647 (see http://www.ietf.org/rfc/rfc4647.txt). |
| **Example** | <supported_languages><br><language>en</language><br><language>fr</language><br></supported_languages> | | |

### 8.4.25 Tag creator alias ('tag_creator_alias')

| | | | |
|---|---|---|---|
| **XML Tag** | tag_creator_alias | | |
| **Type** | Complex type – EntityDataComplexType | | |
| **Definition** | Provides additional tag creator information enabling SAM practitioners and SAM tool providers to identify previous entities who were related to the creation of the software identification tag. Though not strictly required for software discovery purposes, this entry will ease the burden of a SAM practitioner by providing them with previous tag creator details which can be used to more easily find an older software entitlement.<br><br>This element may occur zero to one time in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | alias | Complextype - AliasDetailsComplexType<br><br>Zero to unlimited entries | Details of previous tag creators who may have a relationship to the software title identified by the software identification tag. |
| **Example** | The following example is appropriate for a Macrovision product that was purchased by and is now owned by Adobe®<br><br><tag_creator_alias><br><alias><br><alias_name>Macrovision</alias_name><br><alias_regid>regid.1998-02.com.macrovision</alias_regid> | | |

| | |
|---|---|
| | </alias><br></tag_creator_alias><br><br>**Or**, if the regid of the alias entity is unknown:<br><br><tag_creator_alias><br><alias><br><alias_name>Macrovision</alias_name><br><alias_regid>unknown</alias_regid><br></alias><br></tag_creator_alias> |

### 8.4.26 Tag creator copyright ('tag_creator_copyright')

| | | | |
|---|---|---|---|
| **XML Tag** | tag_creator_copyright | | |
| **Type** | XML character string | | |
| **Definition** | This element is provided in order to enable the tag creator the chance to specify the copyright for this particular tag. It is expected that software creators will allow their tag to be collected and distributed as long as creator-specified contents of the tag are not modified. This allows SAM tool providers and others to access and use software identification tags easily within their tools.<br><br>An independent 3rd party that creates tags may put more limitations on the use and/or redistribution of the software identification tag data.<br><br>See Annex F for more details on copyright information.<br><br>The abstract element may occur more than once in a software identification tag, but shall only occur once for each language specified.<br><br>If language is not specified, it is assumed to be English ("en").<br><br>This element may occur zero to unlimited times in a software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | lang | XML character string. This is an optional tag attribute | The language the abstract is written in. Languages shall be specified as defined in IETF RFC 4646 - http://www.ietf.org/rfc/rfc4646.txt. |
| **Example** | <tag_creator_copyright lang="en">This tag may be used by used, stored, referenced and distributed by any software tool provider and or third party tag collection agency as long as the following elements are not modified:<br><br>- entitlement_required_indicator<br>- product_title<br>- product_version<br>- software_creator<br>- software_licensor<br>- software_id<br>- tag_creator<br><br>Extended information may also be added to the tag.<br></tag_creator_copyright> | | |

### 8.4.27 Tag version ('tag_version')

| | | | |
|---|---|---|---|
| **XML Tag** | tag_version | | |
| **Type** | Complex Type | | |
| **Definition** | This element provides a data element for tag creators or tag modifiers to provide tag version information. A properly defined software identification tag does not need to have a version specified by the tag creator since every software identification tag is unique. However as tags move through the software lifecycle, multiple tag modifiers may want to make changes to elements they are allowed to modify and/or to add extended elements to a software identification tag. In these cases, a version reference is required. There is a need to allow multiple entities to provide their own version information, meaning this element may be included multiple times within a single software identification tag. Each time a version element is provided all elements within the version element are required items to ensure uniqueness.<br><br>This element may occur zero to unlimited times in the software identification tag. | | |
| **Data Structure** | **XML tag** | **Type** | **Definition** |
| | name | XML character string<br><br>One entry | This element provides the name of the entity defined in the tag. This name should be consistent between software products and software releases. |
| | regid | regid type<br><br>One entry | Regid of the software creator (as specified in section 6.1.3.) If the entity is unknown, or is no longer in business, this value may be set to "unknown". |
| | numeric_version | ProductVersion ComplexType - Complex type consisting of four elements with numeric values: "major", "minor", "build", "review"<br><br>One entry | Numeric version identifier |
| **Example** | <tag_version><br><name>My Example Corp</name><br><regid>regid.1995-09.com.example</regid><br><numeric_version><br><major>1</major><br><minor>0</minor><br><build>0</build><br><review>0</review><br></numeric_version><br></tag_version> | | |

### 8.4.28 Upgrade for ('upgrade_for')

| XML Tag | upgrade_for | | |
|---|---|---|---|
| Type | Complex type | | |
| Definition | Product title that represents an upgrade for an earlier, down-level version, providing specific details about what is upgraded.<br><br>This element may occur zero to unlimited times in the software identification tag. | | |
| Data Structure | XML tag | Type | Definition |
| | upgrade_id | Software identification type described in 8.3.5<br><br>One to unlimited entries | This refers to the software_ids that may be upgraded to the version indicated in this tag. If the software_ids are provided, SAM managers can then do a completely automated reconciliation to ensure that all upgrades are done appropriately. |
| | upgrade_description | XML character string<br><br>Zero or one entry | Optional description of upgrade |
| Example | <upgrade_for><br><upgrade_id><br><unique_id>fc3cc419-b5a1-9f16-ed203e537c40</unique_id><br><tag_creator_regid>regid.1986-12.com.adobe</tag_creator_regid><br></upgrade_id><br><upgrade_description>{Optional description of upgrade}</upgrade_description><br></upgrade_for> | | |

### 8.4.29 Usage identifier ('usage_identifier')

| XML Tag | usage_identifier |
|---|---|
| Type | Complex Type |
| Definition | Provides information specifying which running process should be used to validate usage of the product. The usage element does not need to specify software components that load at startup, but can for example specify those components that indicate that an end-user is actually using the product.<br><br>This element may occur zero to unlimited times in the software identification tag. |

| Data Structure | XML tag | Type | Definition |
|---|---|---|---|
| | filename | XML character string<br><br>Zero to unlimited entries | This defines the filename that is executed to start an application.<br><br>This element may occur zero to unlimited times in the software identification tag. |
| | processname | XML character string<br><br>Zero to unlimited entries | This defines the process name as it would be found in the process table for the application.<br><br>By default, this value will be interpreted as a literal expression. Processname has an attribute called "type" that allows for the definition of a "literal" or "regexp" (regular expression). If the type for processname is "regexp", the value provided shall follow the regular expression syntax as defined at http://www.w3.org/TR/xmlschema-2/#regexs.<br><br>This element may occur zero to unlimited times in the software identification tag. |
| | URI | XML URI type<br><br>Zero to unlimited entries | This defines a web based location that needs to be accessed at runtime to determine application usage. This may be used in the case of an on-line tool that an organization wants to track usage on, or it may be used in a case where an application requests information from a URI during execution.<br><br>By default, this value will be interpreted as a literal expression. URI has an attribute called "type" that allows for the definition of a "literal" or "regexp" (regular expression). If the type for URI is "regexp", the value provided shall follow the regular expression syntax as defined at http://www.w3.org/TR/xmlschema-2/#regexs.<br><br>This element may occur zero to unlimited times in the software identification tag. |
| Example | <usage><br><filename>winword.exe</filename><br><processname>windword.exe</processname><br></usage><br><br>**Or**<br><br><usage><br><URI type="regexp">https?://[^/]*/MyWebApp/</URI><br></usage> | | |

### 8.4.30 Validation ('validation')

| XML Tag | validation | | |
|---|---|---|---|
| Type | Complex Type | | |
| Definition | Provides a callout that a discovery agent can use to validate the software identification tag. Due to system issues, installation and uninstallation defects or other issues, a software identification tag may not be fully in sync with the software installation. This element allows a software identification tag to include a validation callout that may be used, if required, to validate that the software identification tag is correct.<br><br>It is expected that software applications will regularly validate software identification tags with which they are associated at the time they are executed – this would be considered a self-healing process. During this self-healing, the tag sub-elements that are part of the validation element, last_validated_by and last_validated_date would be updated to show that the tag has been compared and found to be accurate.<br><br>In those cases where a software package is not run for an extended period, the last_validated_by and last_validated_date would not be updated. A discovery agent, by policy, may require that a software identification tag be validated within a specified period (for example, if a software identification tag has not been validated in 3 months, it may be considered suspect and possibly out of sync). If the last_validated_date is older than the specified period, the discovery agent can call the routine defined in validation_call to ensure the software identification tag is up-to-date.<br><br>This element may occur zero to one time in the software identification tag. | | |
| Data Structure | **XML tag** | **Type** | **Definition** |
| | validation_call | XML character string<br><br>One entry | This is a call that a discovery agent can make to validate that the software identification tag is valid. It is expected that this call would be to one of the executable applications in the software package with a command line parameter that specifies that the tag should be validated.<br><br>The validation_call could also be specified as a URL reference. |
| | last_validated_by | XML character string<br><br>Zero or one entry | This element identifies the process that was used to validate the software identification tag. It is expected that the software creator will create an ID for any validation routines and include that ID in this element.<br><br>Depending on the software creator, this reference may be the same for all software creator's titles (i.e. ACME's validation routine), or it may be unique per package. It may even be possible that an installation routine may be used to do a tag validation. |
| | last_validated_date | XML dateTime type<br><br>Zero or one entry | The last date and time that this tag was validated. |
| | last_validated_result | XML character string | This element will be "True", "False" or "Unknown" (case of the string value does not matter). The purpose of this element is to provide a method that can identify if the |

| | | | |
|---|---|---|---|
| | | Zero or one entry | Validation_call returns with a "True" – that the tag was found to be valid, "False" - that the tag was not valid and should be reviewed, or "Unknown" – that the validation call could not be made, or failed to return a result, so it is unknown if the tag is valid or not.<br><br>For the SAM practitioner, if they receive software identification tag information that includes "Validation_result=false", that will provide an exception that needs to be investigated. It does not indicate that the tag is not valid, it simply indicates that there is an issue with the validation process and lets the SAM practitioner become aware of a problem. |
| **Example** | <validation><br><validation_call>c:\program files\ACME\ACME_validator.exe /tag-validate</validation_call><br><last_validated_by>ACME_validator.exe</last_validated_by><br><last_validated_date>2008-03-31T12:00:00</last_validated_date><br><last_validated_result>true</last_validated_result><br></validation> | | |

## 8.5 Extended elements

### 8.5.1 Extended information ('extended_information')

| | |
|---|---|
| **XML Tag** | extended_information |
| **Type** | Sequence of elements of any type |
| **Definition** | Supplemental information that may be provided by the software or tag creators, the purchaser of the software, or a 3rd party (such as a distributor, SAM tool or desktop management tool).<br><br>Data shall be provided in an XML structure.<br><br>This element contains any extended information required. Data provided in this section shall be provided in an XML-compliant structure. Additionally, an XSD should be provided so this section can be properly validated. The XSD file shall be referenced properly in the software identification tag XML file as per standard XML definitions.<br><br>Since this element is optional, it may not appear in the software identification tag. If this element is in the software identification tag, it may appear multiple times. The expectation is that every extended_information section included in the software identification tag is owned and managed by a single organization. Thus, a software creator may provide one extended_information element, a reseller may provide another extended_information element and a release manager may provide a 3rd extended_information element. Each of these will be independent entries and a tag modifier should generally not modify data in an extended_information section that they did not create or own.<br><br>This element may occur zero to unlimited times in the software identification tag. |
| **Example** | <extended_information><br><software_creator_activation_ref>xyzzy</software_creator_activation_ref><br></extended_information> |

## 8.6 Data type definitions

### 8.6.1 AliasDetailsComplexType

| Data Type | Complex Type | | |
|---|---|---|---|
| **Definition** | This type is used to define aliases that may have previously been associated with the software identified in the tag. Aliases are associated with software_creator_alias, software_licensor_alias and tag_creator_alias.<br><br>In general, aliases would be used if an organization changed names, or if a software product changed owners. Providing the alias details allows SAM practitioners with information they may need in order to associate a discovered software title with previously purchased versions of related software. | | |
| **Data Structure** | **XML Tag** | **Type** | **Definition** |
| | alias_name | XML character string<br><br>One entry | This element provides for the definition of the names of previous entities that may have been associated with the software defined by a specific tag. |
| | alias_regid | XML character string<br><br>One entry | This element provides for the definition of a specified Regid of previous entities (as specified in section 6.1.3.) If the entity is unknown, or is no longer in business, this value may be set to "unknown". |
| | <alias><br><alias_name>Macrovision</alias_name><br><alias_regid>regid.1998-02.com.macrovision</alias_regid><br></alias> | | |

### 8.6.2 EntityComplexType

| Data Type | Complex Type | | |
|---|---|---|---|
| **Definition** | This type is used to define the specific unique details for entities defined in software_creator, software_licensor or tag_creator. | | |
| **Data Structure** | **XML Tag** | **Type** | **Definition** |
| | name | XML character string<br><br>One entry | This element provides the name of the entity defined in the tag. This name should be consistent between software products and software releases. |
| | regid | XML character string<br><br>One entry | Regid of the entity. If the entity is unknown, or is no longer in business, this value may be set to "unknown". |
| | <tag_creator><br><name>Adobe</name><br><regid>regid.1986-12.com.adobe</regid><br></tag_creator> | | |

### 8.6.3 EntityDataComplexType

| Data Type | Complex Type | | |
|---|---|---|---|
| Definition | This type is used to define the additional aliases for entities associated with the software identified by a specific software identification tag. This includes any aliases (other owners) that may have previously been associated with the software identified in the tag as well as a regid to be specified for a specific entity. | | |
| Data Structure | XML Tag | Type | Definition |
| | alias | Complex type AliasDetailsComplexType<br><br>Zero to unlimited entries | This element provides for the definition of previous entities that may have been associated with the software defined by a specific tag. In general, aliases would be used if an organization changed names, or if a software product changed owners. Providing the alias details allows SAM practitioners with information they may need in order to associate a discovered software title with previously purchased versions of related software. |
| | The following example is appropriate for a Macrovision product that was purchased by and is now owned by Adobe®.<br><br><tag_creator_alias><br><alias><br><alias_name>Macrovision</alias_name><br><alias_regid>regid.1998-02.com.macrovision</alias_regid><br></alias><br></tag_creator_alias> | | |

### 8.6.4 GUIDType

| Data Type | GUIDtype | |
|---|---|---|
| Definition | This data type specifies a GUID and validates that the data value matches a GUID definition that should look something like the following:<br><br>00001101-0000-1000-8000-00805f9b34fb | |
| Data Structure | Type | String |
| | Restrictions | pattern value="[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}" |

### 8.6.5 ProductVersionComplexType

| Data Type | ComplexType | | |
|---|---|---|---|
| **Definition** | This data type is created to provide a standardized 4 digit numeric version. If the version number used for the product has less than 4 levels, lower levels should be set to zero '0' | | |
| **Data Structure** | **XML Tag** | **Type** | **Definition** |
| | major | Integer<br><br>One entry | Highest level of the version number. Typically, this is called the major version. |
| | minor | integer<br><br>One entry | 2nd level of the version number. Typically, this is call the minor version |
| | build | integer<br><br>One entry | 3rd level of the version number. Many organizations call this the build version. |
| | review | integer<br><br>One entry | 4th level of the version number. Many organizations call this the patch or review level. |

### 8.6.6 FootprintModuleComplexType

| Data Type | Complex Type | | |
|---|---|---|---|
| **Definition** | This type is used to define the files, registry entries and other data values associated with a specific software package installation. | | |
| **Data Structure** | **XML Tag** | **Type** | **Definition** |
| | referenced | XML URI type<br><br>zero to one entries | This element is similar to the external_description but can be used for the specific footprint module only. |
| | file | File footprint complex type:<br><br><file><br>< name><br>(<size>)*<br>(<md5>)*<br>(<version>)*<br>(<other name>)*<br></file><br><br>zero to unlimited entries | This element represents description of the file with its attributes: size, MD5, version, and any other needed. |

| | | | |
|---|---|---|---|
| | os_configuration_record | OS configuration entry complex type<br><os_configuration_record><br><record_type><br>(<path>)?<br>(<name>)?<br>(<internal_path>)?<br>(<entry>)+<br>(<name>)?<br>(<value>)?<br>(<type>)?<br></entry><br></os_configuration_record><br>zero to unlimited entries | This element provides other OS configuration information that a software manufacture may want to provide to indicate that their software is installed. Expected values for the record_type element are:<br>a) Registry<br>b) WMI<br>c) RPM<br>d) ODM<br>e) file-entry<br>Examples showing use of these items are shown below.<br>Other types may be defined over time see the ISO 19770 web page for updates - http://standards.iso.org/iso/19770/. |
| | other | Other footprint complex type<br><other type><br><param name><br></other><br>zero to unlimited entries | This element represents any footprint information that is required, but not listed above. It may contain any attributes. |

## <os_configuration_record> details and examples

**registry**

An os_configuration_record with record_type of registry uses the specified elements as a regular request to capture registry values from Microsoft® Windows® computers. Elements are used as follows:

| | |
|---|---|
| <record_type> | **registry** |
| <path> | full path to the registry value. This includes the fully specified root (i.e. HKEY_LOCAL_MACHINE) |
| <name> | NA |
| <internal_path> | NA |
| <entry> | |
| <name> | registry value required for comparison |
| <value> | comparison value |
| <type> | type of registry value |
| </entry> | |

Example:

```
<os_configuration_record>
      <record_type>registry</record_type>
      <path> HKEY_LOCAL_MACHINE\SOFTWARE\Adobe\Acrobat
Reader\7.0\AdobeViewer</path>
      <entry>
            <name>EULA</name>
             <value>1</value>
            <type>REG_DWORD</type>
      </entry>
</os_configuration_record>
```

**WMI**

An os_configuration_record with a record_type of WMI uses the specified elements as part of a WBEM Query Language (WQL) request. Elements are used as follows:

| | |
|---|---|
| <record_type> | **WMI** |
| <path> | namespace where the class is located |
| <name> | class of the object |
| <internal_path> | query statement used (i.e. where state='stopped' |
| <entry> | |
| <name> | attribute name |
| <value> | comparison value |
| <type> | type of class attribute value (optional) |
| </entry> | |

Example:

```
<os_configuration_record>
      <record_type>WMI</record_type>
      <path>Root\CIMV2</path>
      <name>Win32_Product</name>
      <internal_path>Name= ‘Adobe Acrobat 8 Professional’</internal_path>
      <entry>
                  <name>Version</name>
                  <value>8.1.2</value>
      </entry>
      <entry>
                  <name>Vendor</name>
                  <value>Adobe Systems</value>
                  <type>string</type>
      </entry>
</os_configuration_record>
```

**RPM**

An os_configuration_record with a record type of RPM uses the specified elements to read data from the RPM data store. Elements are used as follows:

| | |
|---|---|
| <record_type> | **RPM** |
| <path> | NA |
| <name> | name of the RPM package |
| <internal_path> | NA |
| <entry> | |
| <name> | attribute from the RPM package to use for comparison |
| <value> | comparison value |
| <type> | type of attribute value |
| </entry> | |

Example:

```
<os_configuration_record>
      <name>lvm2-2.02.28-1.fc8</name>
      <type>RPM</type>
      <entry>
            <name>Name</name>
            <value>lvm2</value>
      </entry>
      <entry>
            <name>Version</name>
            <value>2.02.28</value>
      </entry>
      <entry>
            <name>Signature</name>
            <value>DSA/SHA1, Thu 25 Oct 2007 06:10:53 AM CEST, Key ID
b44269d04f2a6fd2</value>
            <type>string</type>
```

```
        </entry>
        <entry>
            <name>Packager</name>
            <value>Fedora Project</value>
        </entry>
</os_configuration_record>
```

**ODM**

**An os_configuration_record with a record type of ODM uses the specified elements to read data from the AIX Object Data Manager. Elements are used as follows:**

```
        <record_type>   ODM
        <path>          NA
        <name>          type of ODM information to retrieve
        <internal_path> NA
        <entry>
            <name>      attribute from the LPP package to use for comparison
            <value>     comparison value
            <type>      type of attribute value
        </entry>
```

Example:

```
<os_configuration_record>
        <type>ODM</type>
        <name>lpp</name>
        <entry>
            <name>name</name>
            <value>bos.msg.en_US.docsearch.client.Dt</value>
            <type>string</type>
        </entry>
        <entry>
            <name>description</name>
            <value>Lite NetQuestion Local Web Server</value>
            <type>string</type>
        </entry>
        <entry>
            <name>ver</name>
            <value>5</value>
            <type>short</type>
        </entry>
</os_configuration_record>
```

**File_entry**

```
        <record_type>   file_entry
        <path>          path to file
        <name>          NA
        <internal_path> NA
        <entry>
            <name>      NA
            <value>     comparison value to find in file
            <type>      type of attribute value
        </entry>
```

Example:

```
<os_configuration_record>
        <type>file-entry</type>
        <path>/etc/inittab</path>
        <entry>
            <value>x:5:respawn:/etc/X11/prefdm -nodaemon</value>
        </entry>
</os_configuration_record>
```

# Annex A
(informative)

# Software identification tagging principles

## A.1 Introduction

The purpose of this Annex is to provide a conceptual overview of software identification tagging principles. It restates many of the normative requirements specified in the body of this part of ISO/IEC 19770, but is not intended to add to or modify those requirements.

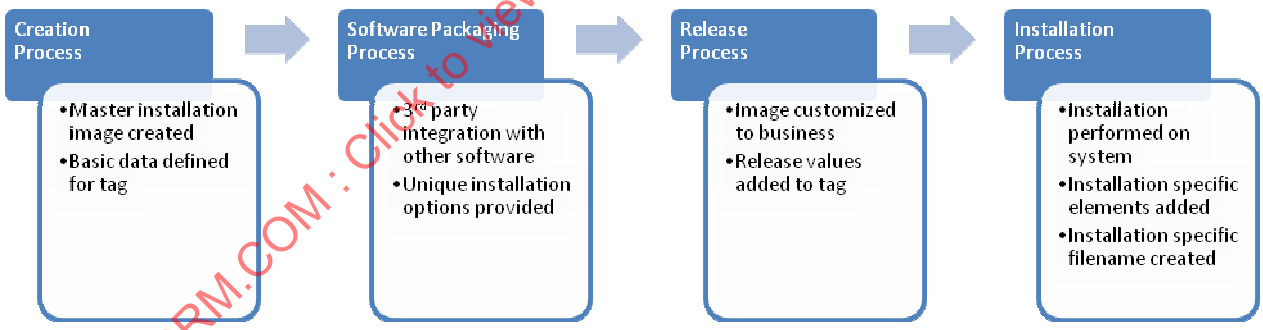
The fundamental objective of the software identification tag is to facilitate the identification and management of installed software. This must take into account the ways software is created, distributed, licensed and used.

## A.2 Life cycle of a software identification tag

### A.2.1 Overview

Data within the software identification tag will be created and/or modified at four major points through the software identification tag life cycle as shown below:

**Figure A.1 — Life cycle of a software identification tag**






Each phase of this process may be specified by different entities. In general, the creation process is specified by the software creator, the release process is specified by the release manager and the installation process is typically specified by the software licensor or the software creator, with some installation-specific information being specified by the software consumer organization.

### A.2.2 Creation process

When software is created as a gold master, it will often include a software identification tag that has a number of elements pre-defined. These elements are generally owned by the software creator or the software licensor. These elements include (elements in bold are required elements):

1. abstract
2. component_of
3. complex_of

4. data_source
5. dependency
6. elements_owner
7. **entitlement_required_indicator**
8. license_linkage
9. package_footprint
10. product_category
11. product_identifier
12. **product_title**
13. **product_version**
14. release_date
15. sku
16. **software_creator**
17. software_creator_alias
18. **software_licensor**
19. software_licensor_alias
20. **software_id**
21. supported_languages
22. **tag_creator**
23. tag_creator_alias
24. tag_creator_copyright
25. upgrade_for
26. usage_identifier
27. validation

In general, these elements should be relatively static for a particular software product and should be provided in the installation master copy.

Software creators may choose to create tags only at installation time. This process is acceptable as long as the data in the tag that is actually installed matches the same type of data that would be provided as part of a product that provides a pre-installation version of the tag. Procedures should be defined that allow the software consumer to provide their own values for release elements in the tag as well as the potential to provide software packagers with the ability to alter specified tag elements.

### A.2.3 Software packaging process

Some software will go through a packaging process that may be done by a 3rd party. This may be done in the case of an OEM product that is integrated into a complete software solution, or by licensed bundlers who may combine multiple products into a suite.

In these cases, the elements that may be updated will vary based on the agreed terms between the software creator, software licensor and the software packager. It may entail the software packager creating their own software identification tag to replace the existing tag, or alternatively to create a package tag which will refer to the existing tag and software as a component. Another alternative is that it may entail the modification and/or

addition of software identification tag elements. If the tag is simply incorporating the existing product without modifying the software_creator or software_licensor owned elements, the elements that may be expected to be added or modified include (none of which are mandatory elements):

1. component_of
2. complex_of
3. data_source
4. dependency
5. elements_owner
6. license_linkage
7. packager

By providing additional information in the software identification tag elements above, software packagers can specify information that can assist a SAM practitioner with identifying software that is linked to their package rather than something that is a stand-alone package.

### A.2.4 Release process

Once a software master version is delivered to an organization, it is often provided with customized installation details and tested prior to the software being distributed and finally installed. During this phase, the release manager is the owner of certain elements in the tag and will often modify specified elements in a software identification tag. The elements relevant to the release process include (none of which are required items)

1. elements_owner
2. packager
3. release_id
4. release_package
5. release_rollout
6. release_verification

Larger software consumer organizations may also choose to add extended information to the software identification tag that provides additional data that can be used for support, SAM procedures, or other processes.

### A.2.5 Installation process

This process is typically defined by the software creator or software licensor and may be modified as appropriate by the release manager within an organization.

When a software product is installed on a computing device, the software identification tag will receive its final filename. As mentioned in section 6.1.7, the use of a unique_sequence_id that includes machine identifying information is highly preferred since it provides general uniqueness to the filename across the enterprise, but more importantly, it provides the ability to identify which system was used to install a software package on a removable device.

At the time of installation, a number of data values will also be updated. These values typically include (none of which are required items):

1. installation_details
2. serial_number

3. validation

## A.3 Unique definition of software_id

A unique software_id corresponds to a unique product at the binary level for distribution/update purposes. Uniqueness is guaranteed by a combination of a unique tag_creator_regid name and a tag_creator maintained unique_id.

The software_id for a specific version of a specific software package should remain consistent for every distribution of that software package. Other details in the software identification tag may change to indicate differences in channel distribution, or even the fact that a software product is included in a bundled version of a 3rd party's software suite.

There may be instances where the actual software_id installed with a software bundle may not be known until installation time. This would be the case in products where multiple configurations of the software are available through a single installer and the installer is only made aware of which version of a software package will reside on a computing device at install time. In these cases, each configuration option that may include different licensing entitlements should be provided with its own unique software_id.

## A.4 Filename specification

### A.4.1 Overview

Filenames will have two different forms – one when the filename is used prior to the software package being installed (distribution filename), the other when the software package is installed on a computing device (installation filename).

### A.4.2 Distribution filename

The distribution filename should follow the rules specified in section 6.1.6. This filename includes the unique software identity and provides information about the tag_creator. The distribution filename is specified by the software_creator or the tag_creator, and is likely to be exactly the same for every distribution copy of the software created.

### A.4.3 Installation filename

When a software identification tag is installed on a computing device, the filename must be unique for that particular device. This is a requirement because multiple software identification tags may be installed in the same directory, and every filename must be unique. This is accomplished using the filename specification as defined in section 6.1.7.

Additionally, it is highly recommended that software identification tag installation routines follow the recommendation in section 6.1.7 to include machine unique information in the installation filename. Doing this provides the ability for SAM practitioners (and others) to identify which machine was used to install software on removable, or shared (network-based) media.

The unique machine information may also include details specific to a particular virtual machine. Though not specifically identified in section 6.1.7, the more unique information that can be provided in a machine ID, the more likely it will be that software identification tags can be associated with the specific device that was used to install the software.

## A.5 Tag installation locations

There are two locations where a software identification tag will be installed on a computing device. One is a common system location (see sections 6.1 and A.6.3); the other is the top level of the installation directory for the software package.

## A.6 Principles of operation without a registration authority

### A.6.1 General

This part of ISO/IEC 19770 has been written to avoid the necessity for a registration authority. A registration authority could maintain central lists of many of the types of information covered by this part of ISO/IEC 19770, such as:

a) All platforms, their respective owners, and where software identification tags should be stored on each

b) Unique software creator names and identifiers

c) Unique software identifiers

### A.6.2 Unique references to identified creators and licensors

The following principles have been incorporated into this part of ISO/IEC 19770 to facilitate operation without a registration authority:

a) The elements tag_creator, software_creator and software_licensor must all utilize a specific registration ID (regid) that is guaranteed to be unique to the organization and that can be used to identify the organization. This regid is created based on the definition developed for the iSCSI standard as specified in the IETF RFC 3720.

b) The regid incorporates the creator's domain name (as specified in IETF RFC 1034, section 3.5 and IETF RFC 1123, section 2.1). Using the regid (and by extension components of the domain for the organization) allows this part of ISO/IEC 19770 to provide for a unique ID that does not require an independent registration authority and provides additional information to track back to the original tag creator.

c) The tag creator has the responsibility for ensuring a unique_id for everything created for the tag creator's regid.

d) There are provisions for on-line reference information, such as for package footprint information. When used as an on-line reference, these require a URI, which uniquely identifies the tag-creator.

This approach allows tag creators to exist and operate independently of software creators, to allow the creation of software identification tags for the software of software creators which may have gone out of business, or which are not creating tags for their software. It also allows the ready creation of software identification tags for software which was created before this part of ISO/IEC 19770 was developed.

### A.6.3 Platform storage specifications

The only information which is not contained within the tag itself, or by an embedded reference, is the specification of where a given platform (e.g. Windows®, UNIX®, and Linux™) will store its software identification tags. There are a comparatively limited number of platforms, but the requirement for information in this area is dealt with in the following ways:

a) Each platform provider has the right to specify where this information will be held on its platform. The platform provider will be able to communicate this information in any way it chooses, e.g. via its website. It

is the recommendation of this part of ISO/IEC 19770 that platform providers publish this information, at least, in a subdirectory named '19770' under their main domain name address.

b) Common system directories that should be used for tag storage on various platforms have been specified in section 6.1.

c) Technical Reports may be published on http://standards.iso.org/iso/19770/-2/ with consolidated information about known platforms and conventions for data values being used commonly by industry in software identification tags.

Furthermore, each software creator should have a unique software creator identity ('software_creator'), which will facilitate the consolidation of information by software creator even if produced by different tag creators. However, the lack of such unique identifiers for specific software creators will not prevent the successful implementation of this part of ISO/IEC 19770.

# Annex B
(informative)

# Software provider use cases and guidance

## B.1 Introduction

The software provider will implement the ISO 19770-2 standard for several reasons:

a) Ease of identification — Software consumers will find it easier to identify software and collect inventory; while allowing increased implementation of SAM practices. Software audits and software auditors (internal or otherwise), will have a better understanding of what is installed across the organization.

b) Accuracy of identification — Current methods of software identification typically rely on software recognition signatures based on application components discovered on machines. These signatures are often not at the same resolution level as software entitlements. Additionally, many product titles have no obvious correlation to the application components that are actually installed. These issues make identification results hard to reconcile with software entitlements.

c) Control over software identification — To ensure consistency, software creators are able to specify exactly what can and cannot be changed in the software identification tag (tag).

When electronic software entitlements become standardized, the software creator and provider who already use tags will be able to implement software entitlements that provide automated or nearly automated reconciliations to existing tags. As a software creator is implementing tags for their product lines, they will need to consider details of how software entitlements may influence how much information is provided in the tag.

From a software product definition and development perspective, it is beneficial to define the tag as early in the project as possible to ensure that the software portion is focused on the right tools and technology to meet the requirements specified for the target languages and platforms.

The use of standardized tags benefits both software consumers and software creators alike. Software consumers gain increased efficiencies through a simplified discovery process and a more effective overall process. Software creators benefit from software consumers having sound asset management practices ensuring software will be installed and used in accordance with the software license agreements.

The following use cases provide different perspectives on how the tag is created and updated through the lifecycle of a software product.



## B.2 Roles involved in the software identification tag creation/management

Numerous individuals occupying different roles will be responsible for ensuring a software identification tag is created and maintained properly, including (but not limited to):

a) Product manager — this person defines the product being developed/enhanced and specifies many aspects of the product that are referenced in the tag. Referring to the tag as part of the product definition, a more specific product document can be created.

b) Development manager — this person determines the technology used to develop and deliver software. Having the tag specifications provided in advance allows them to have a clearer understanding of the end-user environment for the product.

c) Software licensing/product release specialist — this person ensures end-users, IT specialists and auditors know exactly which piece of software they are using. If a suite or bundle consists of multiple products, this group needs to understand how that may impact licensing or product release activities (including catalog updates). This person is responsible for making sure that a software item, its license and any associated catalog information is correct and properly cross-referenced.

## B.3 Product manager role

The product manager is responsible for defining the requirements for a software product. Part of these requirements will be to specify the elements a software identification tag (tag) needs to include and often the values that must be specified for certain tag elements. The more details a product manager can specify about the various tags that may be associated with a product, the more details product development and software release specialists will have available to understand product requirements.

The following elements will often be important to the product manager and should be well understood.

a) Mandatory elements

As the product manager generates the product definition, the definition template should include a section that specifies required portions of the tag. Initially the product manager will work through the mandatory elements in the tag:

1) Entitlement required — this tag determines whether or not the discovered software should be accounted for during the reconciliation process. The product manager can clearly state when an entitlement is needed (when a software license sold to software consumers), or not (when software is installed in trial mode, or is provided for free).

2) Product title — this element corresponds to the official marketing name of the product as defined by the software creator. Typically this is the name which the SAM practitioner and IT specialists will call the product. The title itself may not have a direct impact in the reconciliation process.

3) Product version — this important element allows the definition of both a text-based marketing version (commonly used by software creators to simplify the naming of a particular version for promotional purposes), and the formal numeric version for the software. The numeric version may include up to four elements which provide the complete version information: major version number, minor version number, build, and review. A software vendor may license a software product by its major or minor version only. In such cases, the product manager should make sure that the software entitlement information (purchase order, invoice, software license certificate or equivalent) can be clearly linked to the specified version number to ensure there is no confusion during the reconciliation process.

4) Software creator identity — the purpose of this element is to provide a unique and consistent identification of the vendor producing the software. While some software companies may have slightly different regional names, it is important to augment the name with an identifier that will remain constant across all countries, regions and languages – this is specified in the regid element. This value should be the same for all products and releases from the software creator.

5) Software unique identifier — the product manager working with the rest of the development organization will define a unique identifier for each product version. The unique identifier will enable proper comparison at reconciliation time.

b) Optional elements

The product manager will define other elements that need to be specified. A best practices approach to the definition of a software identification tag should include allowing the software creator to know which optional elements are best specified as the software is shipped as well as understand which software elements are likely to be modified as the software progresses through the sales channel and eventual installation.

The ISO 19770-2 standard includes a rich set of optional elements that augments the content of the tag itself and increases efficiencies in the identification and reconciliation processes. This section highlights the use of a few key optional tags defined in the standard.

1) Component association and components list — these two optional elements enable product managers to tag a software program as a component of a licensing product entity such as suite or bundle. Likewise, the components list element provides the ability to list the remaining components associated with the same licensing entity (i.e., the suite). Inclusion of one or both optional elements in the tag greatly improves the proper identification of seemingly independent software installations and at the same time ensures the reconciliation process will account for the proper software license of suite versus point products and vice-versa.

2) License and channel information — this additional element further refines the identification of the software installed on a given machine, increasing the efficiency of the SAM practitioner and the effectiveness of the overall license reconciliation process. For example, knowing the source of an installation would make it easy for SAM practitioners to separate software purchased directly versus titles included as OEM with the purchase of new personal computers (PCs).

3) Package footprint — this element allows the software creator to specify the files, registry entries, and other information that can be used to identify a software package. The element provides for multiple entries per file so that patches and minor releases can be handled efficiently with a single referenced file. The goal is to provide information that a discovery agent can use to validate that a tag is correctly identifying installed software. An additional benefit allows a discovery tool to eliminate all "known" files from the discovered list. Providing an authoritative list of files, registry entries, WMI entries, or other platform specific information for a particular product helps discovery tools and SAM practitioners to filter information from the list of all discovered items. By filtering out "known" information, practitioners will have a useful perspective of unknown or new software that may be installed in their environment.

4) Product identifier — this element is meant to be an identifier that follows a specific product from release to release. This should not be used as a marketing term for the product (such as the product title), but should be an identifier that is consistent from release to release. For example, if an organization may create and sell a product called "Acme Widgets 2007 Pro" and the next release of that product came out with the name "Acme Widgets 2008 Expert", using a product title, a practitioner would not know whether these two products were part of a specified maintenance agreement. However, if both products had the Product Identifier of "fc3cc419-b5a1-9f16-ed203e537c40", then a practitioner could determine that both products were part of the same maintenance agreement and then determine compliance. This element allows an organization to specify which upgrades are allowed through maintenance agreements without identifying a specific title.

5) Serial number — including the serial number as part of the identification tag is directly proportional to the importance of the serial number as a software entitlement element for the software license purchased by software consumers. Software publishers that require software consumer and/or purchase-specific serial numbers in order to install and use the software should strongly consider including the serial number element in the tag for direct correlation to the purchase order / software entitlement during the reconciliation process. Serial numbers are known at the time of installation, therefore making it possible to establish a direct link between the installation and the licensing software entitlement.

    i) Special consideration regarding serial numbers in the tag: because serial numbers often represent the enablement of specific software features, product managers must consider whether to include the serial number in the clear as part of the identification tag, or otherwise include a one-way hash form of the software consumer-specific serial number to minimize exposure and possible leakage of valid serial numbers over the Internet. If a software vendor chooses to include an obfuscated/hashed version of the serial number in the tag, the same obfuscated/hashed version should be included in the purchase order, software entitlement or software license certificate for use during the manual or automated reconciliation process.

6) SKU — The SKU may be an important element for those companies that do not utilize serial numbers as part of the activation of software. with the SKU is typically needed to associate installed software with a software entitlement.

7) Supported languages — The supported languages element allows software creators to specify the specific language(s) installed on a machine. This information is important for software vendors who sell language-specific software licenses as the software entitlement/purchase order would specify the language-specific products sold to the software consumer.

8) Software creator alias — This optional element reflects the name of the creator prior to an acquisition; this is particularly important in the case where a specific upgrade is allowed from a previous software creator's version to the current creator's version. This should be an included element in the tag when releasing new versions of software products after an acquisition.

9) Upgrade for — This element is designed to simplify automated reconciliations to ensure that upgrades are reconciled properly. This allows a product to identify itself as an upgrade for a specific product or products. With this information, SAM practitioners or auditors will be able to reconcile existing installations of the upgrade product with known entitlements of the older products. Software creators expect to enforce upgrade licenses, and this element allows association of current titles with the original versions.

The use of digital signatures ensures the integrity of elements remains unchanged after the tag has been created. Cost will need to be considered for the extra protection versus the implementation and testing requirements in order to make a decision whether or not to sign specific elements within the tag.

## B.4 Development manager

The development manager will work directly with the product manager to ensure the product specification is clear and complete. The development manager will need the same level of detail as the product manager.

However, when it comes to design and implementation, the development manager will need more detail. The following list summarizes some implementation details that need to taken into consideration and decided upon prior to proceeding with implementation:

a) **Software identification tag file creation:** When and how will the software identification tag be created?

There are multiple approaches to consider. The specific circumstances of each software creator, development cycles, operational and manufacturing process will determine the approach that will work best. Possible options include:

1) Pre-generated tag file(s) — included with the installation disc, selected and copied to the target machine at installation time.

2) Generated at installation time — software identification tag file is created as part of the installation process. This allows a tag to include specific installation options, such as which version was actually installed.

3) Partially pre-generated and updated on the fly — this option allows the software identification tag to be provided upon initial installation of the software and updated as necessary. Updates may include changes to activation status or other elements as the product is run and/or registered.

b) Software identification tag file for multi-product licensing entities (e.g. suites) — How will the tag file be created and maintained?

Development managers need to consider where and how the tag file(s) for the suite components will be created and maintained. Options for the creation of software identification tags may include providing a software identification tag for the suite and separate individual software identification tags for each

individual product, or creating one software identification tag that details the entire suite. Additionally, if a product is validating its tag, will it validate for an individual application and for the suite, or for all applications that are part of the suite.

c) Lifecycle management of software identification tag files — How does the software identification tag file evolve with changes to the installed software?

Development managers need to anticipate a number of lifecycle scenarios that will impact software identification tags; scenarios may include:

1) Patches and updates — if the patch or update changes the product version, the tag file will need to reflect the latest version. If the product version element was originally signed, then the signature as well should be updated.

2) Missing software identification tag file — tag files may be accidentally deleted by end-users or corrupted; in such cases, the software should implement self-healing mechanisms to regenerate the tag.

3) Trial-licenses — when trial-licenses expire, updates to the tag should reflect the trial has expired (i.e. update the activation status element).

4) Evidence of tampering — if the file includes signed tag elements, the software should perform periodic signature checks to detect potential tampering. If tag tampering is detected, the software should implement a self-healing mechanism to regenerate the tag. The development manager and the product manager should determine business policies concerning tampering.

d) Additional implementation considerations

1) How can the tag be validated for correctness during the QA cycle?

2) Centralized function versus product-specific implementation: mid to large software creators should strongly consider isolating product teams from the details and implementation overhead through developing and maintaining a centralized software identification tagging function.

3) Common commercial off the shelf (COTS) software will benefit from providing software provider specified package footprint information (see package footprint element - 8.4.10). This footprint information can be hosted on the software creator's site to enable ease of management. Providing package footprint information will significantly aid SAM tools and SAM practitioners with automatic filtering out of "known" files, registry and WMI entries and other items found during discovery, and allow development of an exception-based SAM management practice.

4) Package footprint, if used, will typically refer to a URI location – generally hosted on the software creator's domain. When providing footprint information, the development manager should institute a policy that keeps all file information up-to-date as patches and minor releases are released to the market. The package footprint element allows more than one entry for each file, so referring to a specific URI location allows tags from multiple versions of a product to be included in a single package footprint.

The development team needs to ensure that the software identification tag is incorporated into the design and lifecycle process for software development and maintenance.

# Annex C
# (informative)

# Tool provider use cases and guidance

## C.1 Discovery tool providers

### C.1.1 Introduction

Discovery tools should be able to read data from software identification tags when available. The roles in a software consumer organization that would use a discovery tool might include an audit manager tasked with reconciling software entitlements to software identification tags or a SAM owner tasked with collecting and analyzing information.

Use cases for tool vendor products and their consumption can be divided into primary and secondary scenarios.

### C.1.2 Primary use cases

Discovery tools should be able to do the following:

a) *Implement consistent and uniform values in software identification tag data.*

NOTE If the tag_creator that created the software identification tag employs multiple "software licensor identity", "product identifier", "serial number" or "stock keeping unit" elements, the discovery tool should be able to refer to a software recognition table (provided by each tag_creator) to facilitate the reconciliation of different values (8.3.5, 8.4.14, 8.4.20, 8.4.21). This task could require extended elements to be furnished by the tag_creator.

EXAMPLE If software company *A* were acquired by software company *B,* then *B* should provide reconciliation information for software formerly released and tagged by *A.* Or, if *A* released a software package only later to release a newer version of the same with an altered "software creator name," then *A* should provide information in the product_id and software_creator alias elements that allows the relationship between the two products to be automatically identified.

b) *Reconcile data from software identification tags with that from corresponding software entitlements.*

NOTE This reconciliation could be facilitated by an agent on the platform or an administrative console reconciling from multiple agents. Reconciliation processes need both software identification tag and software entitlement data to determine software license compliance.

EXAMPLE The tool should be able to identify differences between installs of different but related products (by dependency, complex, platform or product version), such as differences between Microsoft® Excel®, Microsoft® Excel® Viewer (unaccompanied), Microsoft® Office Standard 2000, Microsoft® Office Standard 2003 and Microsoft® Office XP Professional.

c) *Read all mandatory elements during a discovery including all optional elements, if available.* Such capability would utilize standardization in location and format of software identification tag data.

### C.1.3 Secondary use cases

Discovery tools should be able to do the following: