All properties (informative)

Property	Type	minCount	maxCount
comment	xsd:string	0	1
creationInfo	CreationInfo	1	1
dataLicense	/SimpleLicensing/AnyLicenseInfo	0	1
description	xsd:string	0	1
element	Element	1	*
extension	/Extension/Extension	0	*
externalIdentifier	ExternalIdentifier	0	*
externalRef	ExternalRef	0	*
import~~s~~	ExternalMap	0	*
name	xsd:string	0	1
namespaceMap	NamespaceMap	0	*
profileConformance	ProfileIdentifierType	0	*
rootElement	Element	1	*
spdxId	xsd:anyURI	1	1
summary	xsd:string	0	1
verifiedUsing	IntegrityMethod	0	*

7.1.24 Tool

Summary

An element of hardware and/or software utilized to carry out a particular function.

Description

A Tool is an element of hardware and/or software utilized to carry out a particular function.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/Tool`

Name:	Tool
Instantiability:	Concrete
SubclassOf:	Element

Superclasses

- /Core/Element

All properties (informative)

Property	Type	minCount	maxCount
comment	xsd:string	0	1
creationInfo	CreationInfo	1	1
description	xsd:string	0	1
extension	/Extension/Extension	0	*
externalIdentifier	ExternalIdentifier	0	*
externalRef	ExternalRef	0	*
name	xsd:string	0	1
spdxId	xsd:anyURI	1	1
summary	xsd:string	0	1
verifiedUsing	IntegrityMethod	0	*

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/identifierLocator`

Name:	identifierLocator
Nature:	DataProperty
Range:	xsd:anyURI

Referenced

- /Core/ExternalIdentifier

7.2.29 import~~s~~

Summary

Provides an ExternalMap of Element identifiers.

Description

Import~~s~~ provides an ExternalMap of an Element identifier~~s~~ that ~~are~~ is used within a document but defined external to that document.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/imports`~~s~~

Name:	import~~s~~
Nature:	ObjectProperty
Range:	ExternalMap

Referenced

- /Core/SpdxDocument

7.2.30 issuingAuthority

Summary

An entity that is authorized to issue identification credentials.

Description

An issuingAuthority is an entity that is authorized to issue identification credentials.

The entity may be a government, non-profit, educational institution, or commercial enterprise.

The string provides a unique identifier for the issuing authority.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/issuingAuthority`

Name:	issuingAuthority
Nature:	DataProperty
Range:	xsd:string

Referenced

- /Core/ExternalIdentifier

SpdxDocument

Description

This field provides information about the relationship between two Elements. For example, you can represent a relationship between two different Files, between a Package and a File, between two Packages, or between one ~~SPDXDocument~~ and another ~~SPDXDocument~~.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/relationshipType`

Name:	relationshipType
Nature:	ObjectProperty
Range:	RelationshipType

Referenced

- /Core/Relationship

7.2.42 releaseTime

Summary

Specifies the time an artifact was released.

Description

A releaseTime specifies the time an artifact was released.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/releaseTime`

Name:	releaseTime
Nature:	DataProperty
Range:	DateTime

Referenced

- /Core/Artifact

7.2.43 rootElement

Summary

This property is used to denote the root Element(s) of a tree of elements contained in a BOM.

Description

This property is used to denote the root Element(s) of a tree of elements contained in a BOM. The tree consists of other elements directly and indirectly related through properties or Relationships from the root.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/rootElement`

Name:	rootElement
Nature:	ObjectProperty
Range:	Element

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/RelationshipType`

Name:	RelationshipType

Entries

affects The `from` Vulnerability affects each `to` Element. The use of the `affects` type is constrained to `VexAffectedVulnAssessmentRelationship` classed relationships.

amendedBy The `from` Element is amended by each `to` Element.

ancestorOf The `from` Element is an ancestor of each `to` Element.

availableFrom The `from` Element is available from the additional supplier described by each `to` Element.

configures The `from` Element is a configuration applied to each `to` Element, during a LifecycleScopeType period.

contains The `from` Element contains each `to` Element.

coordinatedBy The `from` Vulnerability is coordinatedBy the `to` Agent(s) (vendor, researcher, or consumer agent).

copiedTo The `from` Element has been copied to each `to` Element.

delegatedTo The `from` Agent is delegating an action to the Agent of the `to` Relationship (which must be of type invokedBy), during a LifecycleScopeType (e.g. the `to` invokedBy Relationship is being done on behalf of `from`).

dependsOn The `from` Element depends on each `to` Element, during a LifecycleScopeType period.

descendantOf The `from` Element is a descendant of each `to` Element.

describes The `from` Element describes each `to` Element. To denote the root(s) of a tree of elements in a collection, the rootElement property should be used.

doesNotAffect The `from` Vulnerability has no impact on each `to` Element. The use of the `doesNotAffect` is constrained to `VexNotAffectedVulnAssessmentRelationship` classed relationships.

expandsTo The `from` archive expands out as an artifact described by each `to` Element.

exploitCreatedBy The `from` Vulnerability has had an exploit created against it by each `to` Agent.

fixedBy Designates a `from` Vulnerability has been fixed by the `to` Agent(s).

fixedIn A `from` Vulnerability has been fixed in each ~~of the~~ `to` Element~~(s)~~. The use of the `fixedIn` type is constrained to `VexFixedVulnAssessmentRelationship` classed relationships.

foundBy Designates a `from` Vulnerability was originally discovered by the `to` Agent(s).

generates The `from` Element generates each `to` Element.

hasAddedFile Every `to` Element is a file added to the `from` Element (`from` hasAddedFile `to`).

hasAssessmentFor Relates a `from` Vulnerability and each `to` Element~~(s)~~ with a security assessment. To be used with `VulnAssessmentRelationship` types.

hasAssociatedVulnerability Used to associate a `from` Artifact with each `to` Vulnerability.

hasConcludedLicense The `from` Software ~~A~~rtifact is concluded by the SPDX data creator to be governed by each `to` license.

hasDataFile The `from` Element treats each `to` Element as a data file. A data file is an artifact that stores data required or optional for the `from` Element's functionality. A data file can be a database file, an index file, a log file, an AI model file, a calibration data file, a temporary file, a backup file, and more. For AI training dataset, test dataset, test artifact, configuration data, build input data, and build output data, please consider using the more specific relationship types: `trainedOn`, `testedOn`, `hasTest`, `configures`, `hasInputs`, and `hasOutputs`, respectively. This relationship does not imply dependency.

hasDeclaredLicense The `from` Software Artifact was discovered to actually contain each `to` license, for example as detected by use of automated tooling.

hasDeletedFile Every `to` Element is a file deleted from the `from` Element (`from` hasDeletedFile `to`).

hasDependencyManifest The `from` Element has manifest files that contain dependency information in each `to` Element.

hasDistributionArtifact The `from` Element is distributed as an artifact in each Element `to` (e.g. an RPM or archive file).

hasDocumentation The `from` Element is documented by each `to` Element.

hasDynamicLink The `from` Element dynamically links in each `to` Element, during a LifecycleScopeType period.

hasEvidence Every `to` Element is considered as evidence for the `from` Element (`from` hasEvidence `to`).

hasExample Every `to` Element is an example for the `from` Element (`from` hasExample `to`).

hasHost The `from` Build was run on the `to` Element during a LifecycleScopeType period (e.g. the host that the build runs on).

hasInput~~s~~ The `from` Build has each `to` Element~~s~~ as an input, during a LifecycleScopeType period.

hasMetadata Every `to` Element is metadata about the `from` Element (`from` hasMetadata `to`).

hasOptionalComponent Every `to` Element is an optional component of the `from` Element (`from` hasOptional-Component `to`).

hasOptionalDependency The `from` Element optionally depends on each `to` Element, during a Lifecycle-ScopeType period.

hasOutput~~s~~ The `from` Build element generates each `to` Element as an output, during a LifecycleScopeType period.

hasPrerequisite The `from` Element has a prerequisite on each `to` Element, during a LifecycleScopeType period.

hasProvidedDependency The `from` Element has a dependency on each `to` Element, dependency is not in the distributed artifact, but assumed to be provided, during a LifecycleScopeType period.

hasRequirement The `from` Element has a requirement on each `to` Element, during a LifecycleScopeType period.

hasSpecification Every `to` Element is a specification for the `from` Element (`from` hasSpecification `to`), during a LifecycleScopeType period.

hasStaticLink The `from` Element statically links in each `to` Element, during a LifecycleScopeType period.

hasTest Every `to` Element is a test artifact for the `from` Element (`from` hasTest `to`), during a LifecycleScopeType period.

hasTestCase Every `to` Element is a test case for the `from` Element (`from` hasTestCase `to`).

hasVariant Every `to` Element is a variant the `from` Element (`from` hasVariant `to`).

invokedBy The `from` Element was invoked by the `to` Agent, during a LifecycleScopeType period (for example, a Build element that describes a build step).

modifiedBy The `from` Element is modified by each `to` Element.

other Every `to` Element is related to the `from` Element where the relationship type is not described by any of the SPDX relationhip types (this relationship is directionless).

packagedBy Every `to` Element is a packaged instance of the `from` Element (`from` packagedBy `to`).

patchedBy Every `to` Element is a patch for the `from` Element (`from` patchedBy `to`).

publishedBy Designates a `from` Vulnerability was made available for public use or reference by each `to` Agent.

reportedBy Designates a `from` Vulnerability was first reported to a project, vendor, or tracking database for formal identification by each `to` Agent.

republishedBy Designates a `from` Vulnerability's details were tracked, aggregated, and/or enriched to improve context (i.e. NVD) by each `to` Agent.

serializedInArtifact The `from` ~~SPDXDocument~~ SpdxDocument can be found in a serialized form in each `to` Artifact.

testedOn The `from` Element has been tested on the `to` Element(s).

trainedOn The `from` Element has been trained on the `to` Element(s).

underInvestigationFor The `from` Vulnerability impact is being investigated for each `to` Element. The use of the `underInvestigationFor` type is constrained to `VexUnderInvestigationVulnAssessmentRelationship` classed relationships.

usesTool The `from` Element uses each `to` Element as a tool, during a LifecycleScopeType period.

7.3.10 SupportType

Summary

Indicates the type of support that is associated with an artifact.

Description

SupportType is an enumeration of the various types of support commonly found for artifacts in the software supply chain. Specific details of what that support entails are provided by agreements between the producer and consumer of the artifact.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Core/SupportType`

Name:	SupportType

Entries

deployed in addition to being supported by the supplier, the software is known to have been deployed and is in use. For a software as a service provider, this implies the software is now available as a service.

development the artifact is in active development and is not considered ready for formal support from the supplier.

endOfSupport there is a defined end of support for the artifact from the supplier. This may also be referred to as end of life. There is a validUntilDate that can be used to signal when support ends for the artifact.

Metadata

`https://spdx.org/rdf/3.0.1/terms/AI/SafetyRiskAssessmentType`

Name:	SafetyRiskAssessmentType

Entries

high The second-highest level of risk posed by an AI system.

low Low/no risk is posed by an AI system.

medium The third-highest level of risk posed by an AI system.

serious The highest level of risk posed by an AI system.

15 Build

Summary

The Build Profile defines the set of information required to describe an instance of a Software Build.

Description

A Software Build is defined here as the act of converting software inputs into software artifacts using software build tools. Inputs can include source code, config files, artifacts that are build environments, and build tools. Outputs can include intermediate artifacts to other build inputs or the final artifacts.

The Build profile provides a subclass of Element called Build.

It also provides a minimum set of required Relationship Types from the Core profile:

- hasInput~~s~~: Describes the relationship from the Build element to its inputs.
- hasOutput~~s~~: Describes the relationship from the Build element to its outputs.
- invokedBy: Describes the relationship from the Build element to the Agent that invoked it.

In addition, the following Relationship Types may be used to describe a Build.

- hasHost: Describes the relationship from the Build element to the build stage or host.
- configures: Describes the relationship from a configuration to the Build element.
- ancestorOf: Describes a relationship from a Build element to Build eelements that describe its child builds.
- decendentOf: Describes a relationship from a child Build element to its parent.
- usesTool: Describes a relationship from a Build element to a build tool.

All relationships in the Build Profile are scoped to the “build” LifecycleScopeType period.

The `hasInputs` relationship can be applied to a config file or a build tool if the nature of these inputs are not known at the creation of an SPDX document.

Metadata

`https://spdx.org/rdf/3.0.1/terms/Build`

Name:	Build

Annex D

SPDX License List ~~Matching Guidelines~~ matching guidelines and T~~emplates~~ templates (Normative)

8 SPDX ~~license list~~ License List matching guidelines

The SPDX License List Matching Guidelines provide guidelines to be used for the purposes of matching licenses and license exceptions against those included on the SPDX License List[1]. There is no intent here to make a judgment or interpretation, but merely to ensure that when one SPDX user identifies a license as "BSD-3-Clause," for example, it is indeed the same license as what someone else identifies as "BSD-3-Clause" and the same license as what is listed on the SPDX License List. As noted here, some of the matching guidelines are implemented in the XML files of the SPDX License List repository.

9 How these guidelines are applied

9.1 Purpose

To ensure consistent results by different SPDX document creators when matching license information that will be included in the License Information in File field. SPDX document creators or tools may match on the license or exception text itself, the official license header, or the SPDX License List short identifier.

9.2 Guideline: official license headers

The matching guidelines apply to license and exception text, as well as official license headers. Official license headers are defined by the SPDX License List as specific text specified within the license itself to be put in the header of files. (see explanation of SPDX License List fields[2] for more info).

The following XML tag is used to implement this guideline: `<standardLicenseHeader>`

10 Substantive text

10.1 Purpose

To ensure that when matching licenses and exceptions to the SPDX License List, there is an appropriate balance between matching against the substantive text and disregarding parts of the text that do not alter the substantive text or legal meaning. Further guidelines of what can be disregarded or considered replaceable for purposes of matching are listed below here and in the subsequent specific guidelines. A conservative approach is taken in regard to rules relating to disregarded or replaceable text.

[1] https://spdx.org/licenses/
[2] https://github.com/spdx/license-list-XML/blob/master/DOCS/license-fields.md

18.2 Guideline

Ignore copyright notices. A copyright notice consists of the following elements, for example: “2012 Copyright, John Doe. All rights reserved.” or “(c) 2012 John Doe.”

The following XML tag is used to implement this guideline: `<copyrightText>`

For example: `<copyrightText>Copyright 2022 The Linux Foundation</copyrightText>`

19 License name or title

19.1 Purpose

To avoid a license mismatch merely because the name or title of the license is different than how the license is usually referred to or different than the SPDX full name. This also avoids a mismatch if the title or name of the license is simply not included.

19.2 Guideline

Ignore the license name or title for matching purposes, so long as what ignored is the title only and there is no additional substantive text added here.

The following XML tag is used to implement this guideline: `<titleText>`

For example: `<titleText>Attribution Assurance License</titleText>`

20 Extraneous text at the end of a license

20.1 Purpose

To avoid a license mismatch merely because extraneous text that appears at the end of the terms of a license is different or missing. This also avoids a mismatch if the extraneous text merely serves as a license notice example and includes a specific copyright holder’s name.

20.2 Guideline

Ignore any text that occurs after the obvious end of the license and does not include substantive text of the license, for example: text that occurs after a statement such as, “END OF TERMS AND CONDITIONS,” or an exhibit or appendix that includes an example or instructions on to how to apply the license to your code. Do not apply this guideline or ignore text that is comprised of additional license terms (e.g., permitted additional terms under GPL-3.0, section 7).

To implement this guideline, use the `<optional>` XML element tag as described in Guideline: omittable text.

21 HTTP ~~Protocol~~ protocol

21.1 Purpose

To avoid a license mismatch due to a difference in a hyperlink protocol (e.g. ~~http vs. https~~ HTTP vs. HTTPS).

21.2 Guideline

~~HTTP~~ http:// and ~~HTTPS~~ https:// should be considered equivalent.

XML files do not require specific markup to implement this guideline.

22 SPDX License ~~list~~ List

22.1 Template access

The license XML can be accessed in the license-list-data repository under the license-list-XML directory. Although the license list XML files can also be found in the license-list-XML[4] repository, users are encouraged to use the published versions in the license-list-data[5] repository. The license-list-data repository is tagged by release. Only tagged released versions of the license list are considered stable.

22.2 License List XML format

A full schema for the License List XML can be found at SPDX License List XML Schema[6].

22.3 Legacy Text Template format

Prior to the XML format, a text template was used to express variable and optional text in licenses. This text template is still supported, however, users are encouraged to use the more expressive XML format.

A legacy template is composed of text with zero or more rules embedded in it.

A rule is a variable section of a license wrapped between double angle brackets <<>> and is composed of 4 fields. Each field is separated with a semi-colon `;`. Rules cannot be embedded within other rules. Rule fields begin with a case sensitive tag followed by an equal sign =.

Rule fields:

- **type:** indicates whether the text is replaceable or omittable as per Substantive text guidelines.
  - Indicated by `<<var; . . . >>` or
  - Indicated by `<<beginOptional; . . .>>` and `<<endOptional>>` respectively.
  - This field is the first field and is required.
- **name:** name of the field in the template.
  - This field is unique within each license template.
  - This field is required.
- **original:** the original text of the rule.
  - This field is required for a rule type: `<<var; . . . >>`
- **match:** a POSIX extended regular expression (ERE).
  - This field is required for a rule type: `<<var; . . . >>`

The POSIX ERE[7] in the match field has the following restrictions and extensions:

- Semicolons are escaped with `\;`
- POSIX Bracket Extensions are not allowed

For example: `<<var;name=organizationClause3;original=the copyright holder;match=.+>>`

[4]https://github.com/spdx/license-list-XML
[5]https://github.com/spdx/license-list-data
[6]https://github.com/spdx/license-list-XML/blob/master/schema/ListedLicense.xsd
[7]http://pubs.opengroup.org/onlinepubs/9699919799/