Federal Office
for Information Security

Technical Guideline BSI TR-03183: Cyber Resilience Requirements for Manufacturers and Products

Part 2: Software Bill of Materials (SBOM)

Document history

Table 1: Document History

Version	***Date***	***Description***
1.0	2023-07-12	Version of BSI TR-03183-2 for first publication
1.1	2023-11-28	Translate to English; update requirements for creator, version and licence information
2.0.0	2024-09-20	Add some definitions and updates in “Terms used”, “Level of detail”, and “Explanations”; add sections “Dependency”, “Express specifications” and “Entry into force and transitional system”; add required data fields (actual filename, executable property, archive property, structured property); add additional field (concluded licences); move source code hash from additional field to optional field; add optional field (declared licences); update CycloneDX minimum version requirement from 1.4 to 1.5; change SPDX minimum version requirement from 2.3 to 2.2.1; alter hash value of executable component to hash value of deployable component; alter URI of the executable form of the component to URI of the deployable form of the component; extend the definition of components
2.1.0	2025-08-20	Add some definitions and updates in “Terms used”, “Licence information”, “Data fields”, and “Explanations”; restructure “Data fields”; update CycloneDX minimum version requirement from 1.5 to 1.6, update SPDX minimum version requirement from 2.2.1 to 3.0.1; introduce and define logical and identified components, clarify usage of BOM references; add section “Mapping of individual data fields”

Federal Office for Information Security
P.O. Box 20 03 63
53133 Bonn
E-Mail: TR03183@bsi.bund.de
Internet: https://bsi.bund.de/dok/TR-03183-en

Table of Contents

1 Introduction

This Technical Guideline describes the requirements for a “Software Bill of Materials (SBOM)”. An SBOM is a machine-processable document and corresponds to an electronic bill of materials / parts list. It inventories a code base and thus contains information on all components used in a software. This information can be presented in different breadth and depths - ranging from a basic structure to a fine-granular breakdown of software products and their components. Different formats exist for the representation and distribution of an SBOM.

An SBOM should be used by every software creator and provider in order to be able to transparently represent software complexity and to know which components (e.g. libraries) are used. This knowledge is essential for software management processes, especially for a continuous IT security process and the lifecycle management of software; it is therefore considered “best practice” for a secure software supply chain. An SBOM can be public or non-public and can be distributed in different ways. Typically, software creators use one or more third-party components. They create and manage the SBOMs of their own software; likewise, they take the consumer role of the SBOMs of the included components. The abundance of SBOM information and the possible differences in the structure of SBOMs mean a great deal of work for each creator. Only automation can effectively address that.

SBOM information can be used to check whether a product is potentially affected by a vulnerability by comparing its component list with the components listed in a vulnerability database. However, an SBOM does not contain any statement regarding vulnerabilities or their exploitability. SBOM data for a given software version is considered static, while vulnerability information is of dynamic characteristics. Whether and to which extent a vulnerability of an integrated component poses a risk to the product covered by the SBOM should not be detailed by the SBOM. The exchange of vulnerability information and the potential impact should be facilitated, by means of Security Advisories e.g. CSAF[1] (Common Security Advisory Framework) or VEX[2] (Vulnerability Exploitability Exchange).

In order to confirm whether a product is affected by a vulnerability or not, it is necessary to compare the SBOM of the product with vulnerability information, such as the CVE (Common Vulnerabilities and Exposures) or Security Advisories of the component manufacturers or providers. Furthermore, an analysis of the software itself is necessary to determine how its potentially affected subcomponents are used, and thus whether and how one's own software product is affected. This must be carried out as part of the vulnerability handling for the product. The result of this analysis is then provided to the users of the software as a Security Advisory or VEX for the product.

Originally, SBOMs were mainly used for license management. This Technical Guideline also specifies this use case in an interoperable manner, in order to provide a complete set of requirements for the common use cases of SBOMs.

In the Cyber Resilience Act (CRA)[3] the compilation of an SBOM is mandatory. The CRA is a market access regulation of the EU for products with digital elements, which obliges their providers to continuously operate a vulnerability handling process and to provide information on their products in a transparent and comprehensible form. In the US, an SBOM is already required for software acquired by the Federal government by the US Executive Order 14028 of May 2021[4] and the FDA (Food and Drug Administration) demands an SBOM to be submitted as part of the approval of new medical devices since March 2023[5].

[1] https://csaf.io/
[2] https://www.ntia.gov/files/ntia/publications/vex_one-page_summary.pdf
[3] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R2847
[4] https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/ Section 4
[5] https://www.congress.gov/117/bills/hr2617/BILLS-117hr2617enr.pdf Section 3305

2 Requirements language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in BCP 14[6] (RFC 2119[7], RFC 8174[8]) when, and only when, they appear in all capitals, as shown here.

[6] https://www.rfc-editor.org/info/bcp14
[7] https://www.rfc-editor.org/rfc/rfc2119
[8] https://www.rfc-editor.org/rfc/rfc8174

3 Basics

3.1 Definition of SBOM

A “Software Bill of Materials” (SBOM) is a machine-processable file containing supply chain relationships and details of the components used in a software product. It supports automated processing of information on these components. This covers both the so-called “primary component” and used (e.g. external/third-party) components.

An SBOM MUST contain certain minimum information (see section 5). It describes any relationship of the covered components as detailed in section 5. A component is “fully described” if its record in the SBOM contains all applicable data fields specified in section 5.2. An SBOM conformant to this Technical Guideline MAY be expanded and detailed as desired as long as this information does not introduce contradictions.

Note: This requirement allows for providing expanded information while checking conformance to this Technical Guideline can still be carried out deterministically. Expanded information not containing the data fields as specified in section 5.2 can always be removed from an SBOM without changing the status of conformance to this Technical Guideline. Therefore, tools can flag non-conforming information and SBOM creators can choose to ignore or fix those errors reported on expanded information. It is often difficult to decide in an automated manner which information is expanded information e.g. if context is missing; thus, such decisions oblige the SBOM creator.

A separate SBOM MUST be generated for each software version. An updated version of the SBOM for a given software version MUST be generated if and only if additional information on the included components is provided or errors in the SBOM data are corrected. Consequently, if any component is altered, a new software version MUST be assigned to this component; and in the case of a delivery item SBOM also to all depending parent components.

An SBOM MUST NOT contain vulnerability information, because SBOM data is static with respect to software that is not changing. Nonetheless, the formats for describing an SBOM may offer the ability to include vulnerability information. Still, a document containing both SBOM and vulnerability information does not conform to this Technical Guideline; see also Appendix, section 8.1.14.

3.2 Terms used

For explanations of the terms used and examples, see Appendix, section 8.1.

3.2.1 Component

A “component” as used in this Technical Guideline is a logical component (see section 3.2.2), a single executable file or an archive file, hence they MUST be represented as components in the SBOM. Additionally, any other file MAY be represented as a component. Note that an archive can be an executable file, too.

The primary component, i.e. the product itself and the root of the dependency tree from the perspective of this SBOM, is also a “component” as used in this Technical Guideline.

For components that comprise multiple components in a manner that the original components cannot be determined the original components SHOULD be listed with the appropriate information according to sections 5.2.2, 5.2.4 and 5.2.5. Information that is not available due to the way a component is assembled has to be omitted even if it is specified as a required data field, e.g. the hash or filename of the component.

For the collection of information about components the following principles MUST apply:

In the case of compiled code

The component that was used during the execution of the linker MUST be listed, even if alternative components or implementation could have been used.

If there is – for technical reasons – no linkage run the interpreted code section applies.

Both statements apply to statically and dynamically linked components / libraries.

In the case of interpreted code

The version number of the component MUST reflect the exact required version, if the component is part of the delivery item.

If the component is not part of the delivery item, the version number of the component SHOULD reflect the minimum required version as defined by the component creator. This SHOULD also take into account other factors than just the factual minimum technical requirement. This implies that the component creator SHOULD skip versions that are end-of-life or have known security vulnerabilities. The minimum required version MAY be determined by the minimum version, which was used for testing, e.g. because it was used during software development.

If multiple implementations of a single dependency exist then the "least common denominator" or the standard implementation SHOULD be used. For example:

- If a script can be run with all Bourne Shell compatible shells the component denoted in the SBOM should be the Bourne Shell.
- If a script can be run with any Python implementation the standard implementation must be used (Python) instead of an alternative implementation (Pyston).
- If a Java program can be run with any Java Runtime Environment (JRE) implementation the Open Source Software implementation (OpenJRE) must be used instead of a proprietary implementation.

3.2.2 Logical component

A logical component is an abstraction level that combines multiple components, e.g. to identify a product by name (as a means of associating a number of files logically to an application; see also Appendix, section 8.1.3).

To fully describe logical components, only the following data fields (see section 5.2) MUST be addressed as specified in section 5.1:

- Component creator
- Component name
- Component version
- Dependencies on other components
- Distribution licences
- Other unique identifiers
- Original licences
- Effective licence
- URL of the **security.txt**

3.2.3 External component

An "external component" is a component whose component creator differs from the component creator of the primary component.

The assignment of information to be listed in case of compiled and interpreted code is described as part of section 3.2.1.

3.2.4 Identified component (without referencing another BOM)

Depending on the level of detail of an SBOM not all components have to be fully described, some merely have to be identified, e.g. see Appendix, section 8.3.4.

For any component that MUST be identified and is not required to be fully described, the following data fields (see section 5.2) MUST be addressed as specified in section 5.1:

- Component creator
- Component name
- Component version
- Other unique identifiers

3.2.5 Component referenced by another BOM (referenced component)

For any component (regardless if fully described or merely identified) that is recorded in another BOM and is referred to by an SBOM, the following data fields (see section 5.2) MUST be extracted from the component description of the referenced component and added in the referencing SBOM in addition to the referred BOM:

- Component creator
- Component name
- Component version

Note: This information is required to identify the referenced component, if the reference cannot be resolved or used to access the referenced BOM.

If inconsistent component information exists in both, the referencing and the referenced BOM, the information of the referenced BOM prevails.

Information beyond the aforementioned data fields (creator, name, version) and the reference to the other BOM SHOULD NOT be recorded in the referencing SBOM to avoid inconsistencies, except for additional data fields required by an SBOM format specification to generate a valid SBOM in this format.

3.2.6 Executable file

An “executable file” as used in this Technical Guideline is any file which comprises code that is executed by a computer, either directly or by a runtime system.

3.2.7 Dependency

In this Technical Guideline a dependency is a directed requirement from one component to another component. This is regardless whether a component is a dependency (as a technical term) or contained in the component, e.g. statically linked or embedded.

3.2.8 Licence information

This Technical Guideline distinguishes between three categories of licence information:

- **Original licence(s)**
  Original licence(s) are all licences that have been assigned by the creator of the component.
- **Distribution licence(s)**
  Distribution licence(s) are all licences under which a component recorded in the current SBOM can be used by a licensee.
- **Effective licence**
  The effective licence is the licence under which the component is used by the licensee that is the creator of the current SBOM.

The terms “concluded licences” and “declared licences” are not defined and used consistently throughout different SBOM standards and implementations. To avoid inconsistencies and confusion and allow for an easy mapping this Technical Guideline uses different terms.

Note: Because the “effective licence” is set by the creator of the current SBOM, it has to be set anew when merging SBOMs. However, this process may result in the same “effective licence”.

3.2.9 Vendor/Supplier vs. creator

In this Technical Guideline, a distinction is made between “Vendor” and “Creator”, since “Manufacturer” may be interpreted as combining these two roles in the sense of “vendor of self-created software” (as the German term “Hersteller” is usually interpreted).

“Vendor” (German: “Anbieter”) describes the role of the entity that provides the software or component. Alternatively, but not necessarily with a commercial background the terms “Supplier” (German: “Lieferant”) are used.

“Creator” (German: “Ersteller”) describes the role of the entity that authored or created the software or component. Alternatively, the term “Author” (German: “Autor”) is used.

As the CRA is a market access regulation and this Technical Guideline specifies technical requirements, this Technical Guideline uses a different terminology than defined by the CRA. Therefore, this Technical Guideline does not mention the terms “Economic Operator”, “Distributor” or “Importer”, as the roles of these parties are unrelated to the technical requirements stated here. These technical requirements are independent of the role that is fulfilling them.

4 SBOM formats

A newly generated or updated SBOM MUST be in JSON- or XML-format and a valid SBOM according to one of the following specifications in one of the specified versions:

- CycloneDX[9], version 1.6 or higher
- System Package Data Exchange (SPDX)[10], version 3.0.1 or higher

Only officially released versions of these specifications MUST be used.

The transitional system of this Technical Guideline is specified in section 7.

A mapping of the information model in section 5.1 and section 3.2.5 to the data fields of the aforementioned SBOM format specifications is available in Appendix, section 8.2.

[9] https://cyclonedx.org/specification/overview/
[10] https://spdx.dev/specifications/

5 Content requirements

5.1 Level of detail

For an SBOM that is compliant with this Technical Guideline, recursive dependency resolution MUST be performed at least for each component included in the scope of delivery **on each path** downward (see Appendix, section 8.1.12) at least up to and including the first component that is outside the scope of delivery (see Appendix, section 8.1.11; for "delivery item SBOM", see Appendix, section 8.3.4).

Note: The first level of dependencies outside the scope of delivery must be identified (for "identified component", see section 3.2.4) in order to correlate these dependencies unambiguously. Without this identification, information is missing which is required to correctly chain two SBOMs, e.g. the SBOM of the delivery item and the SBOM of the environment in which it is executed.

If the primary component depends on multiple instances of a component which have different meta-information, all these instances MUST be listed separately with their individual meta-information.

SBOMs of used components MAY be referenced instead of being merged into the SBOM of the primary component, if and only if they are compliant with this Technical Guideline. The provider of the SBOM of the primary component is responsible for the availability of referenced SBOMs.

To determine the data fields that MUST be recorded in an SBOM the following order applies:

1. Referenced component (see section 3.2.5)
2. Identified component (see section 3.2.4)
3. Fully described component (see section 5.2)

The component in a referenced SBOM MUST either be a referenced component or provide all data fields that are required by the level of detail of the SBOM of the primary component.

An SBOM compliant with this Technical Guideline MUST contain the same information as available during the build process or equivalent information where the build process does not exist (for details related to Build SBOM, see Appendix, section 8.4.3).

Note: For easier identification by humans, the filename of the SBOM itself may reflect the name of its primary component.

5.2 Data fields

5.2.1 Required data fields for the SBOM itself

Each SBOM MUST contain at least the following information:

Table 2: Required data fields for the SBOM itself

Data field	***Description***
Creator of the SBOM	Email address of the entity that created the SBOM. If no email address is available this MUST be a "Uniform Resource Locator (URL)", e.g. the creator's home page or the project's web page.
Timestamp	Date and time of the SBOM data compilation according to the specification of the formats (see section 4). Note: It is recommended to only use timestamps in UTC ("Zulu" time).

Note: An SBOM format specification may require additional data fields to generate a valid SBOM in this format.

5.2.2 Required data fields for each component

For each component included in an SBOM, at least the following information MUST be provided:

Table 3: Required data fields for each component

Data field	***Description***
Component creator	Email address of the entity that created and, if applicable, maintains the respective component. If no email address is available this MUST be a "Uniform Resource Locator (URL)", e.g. the creator's home page or the project's web page. See also Appendix, section 8.1.8. Note: If the creator of a component still maintains the component but operates under a different name than the one at the time of integrating the component, the current name may be used.
Component name	Name assigned to the component by the component creator. If no name is assigned this MUST be the actual filename. Note: If the component name was changed between versions, the component name that was valid during the integration of the component must be used.
Component version	Identifier used by the creator to specify changes in the component to a previously created version. The following points apply to determine a version in this order: 1. Existing identifiers MUST NOT be changed for this purpose. 2. Identifiers according to Semantic Versioning[11] or alternatively Calendar Versioning[12] SHOULD be used if one determines the versioning scheme; this is often the component creator. 3. If no version is assigned this MUST be the modification date of the file expressed as date-time according to RFC 3339[13] section 5.6. To determine the creation time the file metadata MUST be consulted. See also Appendix, section 8.1.9.
Filename of the component	The actual filename of the component (i.e. not its file system path); see also section 3.2.1
Dependencies on other components	Enumeration of all components on which this component is directly dependent, according to requirements in section 5.1, or which this component contains according to requirements in section 3.2.1. Furthermore, the completeness of this enumeration MUST be clearly indicated.
Distribution licences	Distribution licence(s) of the component under which it can be used by a licensee. For specifics see sections 6.1 and 8.1.13.

[11] https://semver.org/
[12] https://calver.org/
[13] https://www.rfc-editor.org/rfc/rfc3339

Data field	*Description*
Hash value of the deployable component	Cryptographically secure checksum (hash value) of the deployed/deployable component (i.e. as a file on a mass storage device) as SHA-512; see also section 3.2.1
Executable property	Describes whether the component is executable; possible values are “executable” and “non-executable”; see also Appendix, section 8.1.4
Archive property	Describes whether the component is an archive; possible values are “archive” and “no archive”; see also Appendix, section 8.1.5
Structured property	Describes whether the component is a structured file, i.e. metadata of the contents is still present (see section 3.2.1); possible values are “structured” and “unstructured”; see also Appendix, section 8.1.6 If a component contains both structured and unstructured parts the value “structured” MUST be used.

Note: An SBOM format specification may require additional data fields to generate a valid SBOM in this format.

5.2.3 Additional data fields for the SBOM itself

Each SBOM MUST additionally include the following information, if it exists and fulfils the requirements of an SBOM format specification for the specific data field:

Table 4: Additional data fields for the SBOM itself

Data field	*Description*
SBOM-URI	“Uniform Resource Identifier (URI)” of this SBOM

5.2.4 Additional data fields for each component

For each component included in an SBOM, the following information MUST additionally be provided, if it exists and fulfils the requirements of an SBOM format specification for the specific data field:

Table 5: Additional data fields for each component

Data field	*Description*
Source code URI	“Uniform Resource Identifier (URI)” of the source code of the component, e.g. the URL of the utilised source code version in its repository, or if a version cannot be specified the utilised source code repository itself.
URI of the deployable form of the component	“Uniform Resource Identifier (URI)”, which points directly to the deployable (e.g. downloadable) form of the component.
Other unique identifiers	Other identifiers that can be used to identify the component or to look it up in relevant databases, such as Common Platform Enumeration (CPE) or Package URL (purl).
Original licences	The licence(s) that have been assigned by the creator of the component. For specifics see sections 6.1 and 8.1.13.

5.2.5 Optional data fields for each component

Each SBOM MAY additionally include the following information, if it exists and fulfils the requirements of an SBOM format specification for the specific data field:

Table 6: Optional data fields for each component

Data field	*Description*
Effective licence	The licence under which the component is used by the licensee that is the creator of the current SBOM. For specifics see sections 6.1 and 8.1.13.
Hash value of the source code of the component	Cryptographically secure checksum (hash value) of the component source code. A specific algorithm how to create a hash value of multiple source files or a source code tree, and which hash algorithm is utilised for that has not yet been determined.
URL of the **security.txt**[14]	Contains the "Uniform Resource Locator (URL)" of the component creator's **security.txt**.

Note: This list is not meant to be exhaustive.

[14] https://www.rfc-editor.org/rfc/rfc9116

6 Express specifications

This section explicitly specifies aspects of section 5 in detail.

6.1 Licence identifiers and expressions

Principles of licence identification:

1. Licences MUST be referred to (in the sense of "named") by their appropriate SPDX licence identifier or licence expression based on such an identifier. This reference is made with so-called "identifiers for licences and licence expressions"[15].
   While the SPDX and CycloneDX specifications allow for including licence text of components in SBOMs, this MUST NOT be used as a substitute for a licence identifier.
2. Indication of licence identifiers not defined by SPDX
   If an appropriate licence identifier can neither be found in the list maintained by SPDX nor constructed by an SPDX licence expression based on a defined licence identifier, the licence database "Scancode LicenseDB AboutCode"[16] MUST be consulted next. Identifiers from this database use the prefix **LicenseRef-scancode-[...]** in their SPDX licence identifier to indicate their origin.
3. Licences which cannot be assigned to an extant identifier or licence expression based on an extant identifier
   If an appropriate licence identifier cannot be discovered in any of the established identifier lists, the prefix **LicenseRef-<licence_inventorising_entity>-[...]** MUST be used according to "Annex B. SPDX license expressions"[17] to assign a unique (for each LicenseRef namespace) licence identifier.

For licence identification the following principles of licence similarity MUST be applied:

- Licences are similar if they can be mapped to another licence according to "Annex C. License List matching guidelines"[18].
- Placeholders and templates in licence texts
  If placeholders and templates exist in licence texts, a replacement of these placeholders and templates by individualised content MUST NOT be understood as a licence modification, but MUST be assigned to the same licence identifier.
- SPDX operators and licence concatenations
  Licence expressions of multiple licensing, licence choices and licence exceptions MUST be mapped with SPDX operators. The licence operators link licence identifiers. Permitted operators MUST be selected according to "Annex B. SPDX license expressions".
- Exception clauses for licences
  If an exception clause is attached to a licence text, it MUST be attached to a licence identifier with **WITH** according to the allowed SPDX operators. The names of the exception clause MUST be described with identifiers analogous to the requirements on licence identification[19].
- Text modifications
  If a wording in a licence text is slightly modified compared to a licence text of a licence with a known licence identifier, this licence identifier SHOULD be used for the modified licence if the modification is not substantial. Examples are the addition or removal of liability-clauses or remarks to existing trademarks.

[15] https://spdx.org/licenses/
[16] https://scancode-licensedb.aboutcode.org/
[17] https://spdx.github.io/spdx-spec/v3.0.1/annexes/spdx-license-expressions/
[18] https://spdx.github.io/spdx-spec/v3.0.1/annexes/license-matching-guidelines-and-templates/
[19] https://spdx.org/licenses/exceptions-index.html

7 Transitional system

The version with the highest version number of this Technical Guideline available on BSI's website is called the "most recent version". To be compliant to this Technical Guideline, the most recent version MUST be used for the generation of SBOMs. Any previous version MUST NOT be applied, except for the immediately preceding version which MAY be applied to generate SBOMs compliant to this Technical Guideline up to six months after a new "most recent version" has been issued according to the document history in table 1.

Unless otherwise defined, an SBOM version compliant to this Technical Guideline at the delivery date of this SBOM version remains complaint to the Technical Guideline, even if new versions of this Technical Guideline have been published afterwards. Consequently, consumers of SBOMs SHOULD be able to interpret SBOM versions that were compliant to this Technical Guideline at the time being delivered.

8 Appendix

This section provides additional, explanatory information.

While the sections 3 to 5 set normative requirements for contents, extent and format of a Software Bill of Materials (SBOM), this section is intended to provide helpful information on these requirements. This includes reasons for certain requirements, further explanation of some of the terms used and examples.

8.1 Explanations

This section provides more information on certain terms or definitions in this Technical Guideline.

8.1.1 Machine-processable

SBOMs are defined as machine-processable files in this Technical Guideline. This implies that machines can create, read, modify, process, analyse and evaluate the content and act based on the data. The content itself is well-defined and structured. The term “machine-readable” can be interpreted in multiple ways and is therefore not used.

8.1.2 Component

In this Technical Guideline a “component” is typically a single executable file[20] or an archive file; however, also logical components can be used as an abstraction level, which is recommended for linking to other SBOMs.

Notes:

- An archive can be an executable file, too.
- The use of the term “file” is necessary in order to unambiguously define which objects this Technical Guideline applies to.

8.1.3 Logical component

A logical component can be an operating system, application, framework, container, etc., when it is required to address a component on an abstract level, e.g. a dependency on an installed operation system (logical component) in contrast to its installable image (archive file).

Another example is that an SBOM for a PHP application inside a container would list without the abstraction level just the container itself and each executable PHP file. The information about the original PHP application as product is lost. The logical component provides a place to represent that information and also allows to reference an extant SBOM of the PHP application.

8.1.4 Executable property

Examples for executable files include compiled binaries (“executables”), interpreted code (e.g. Python, Shell, Java) and shared libraries. Not covered by the term are, e.g. configuration files, graphic files, documentation.

Note: This property is important in order to identify files, which may contain malicious code.

8.1.5 Archive property

An archive is a combination of multiple components.

[20] see also https://en.wikipedia.org/wiki/Executable

Notes:

- Compression of an archive does not change this property.
- This property is important in order to identify files, which may need to be dissected.

8.1.6 Structured property

Structured archives contain metadata so that the original components can still be determined afterwards. Examples include containers, packages, ISO images, and archives as .zip, .tar, .tar.gz, .7z.

Unstructured archives do not contain such metadata, i.e. information about their structure is not embedded in the file. Examples for unstructured archives include firmware images or other archives that are not decomposable into their original components.

Notes:

- Binaries containing statically linked libraries are unstructured, executable files, with the exception of self-extracting archives which are structured archives while they are executable files and potentially contain a statically linked decompression algorithm.
- This property is important in order to identify files, which can be dissected.

8.1.7 Examples for executable, archive and structured property

Table 7: Examples for executable, archive and structured property

Example	*Executable*	*Archive*	*Structured*	*Comment*
Configuration file	–	–	– / √	Not required to be listed in an SBOM.
rpm package	–	√	√	
Python script	√	–	–	
Binary with integrated SBOM	√	–	√	
Firmware image	√	– / √	–	
Self-extracting archive	√	√	√	

8.1.8 Component creator

If the source file of a component is cloned, or if the maintainer of a component renames the project, or if the contents of the components source code (if applicable) is changed, the maintainer steps into the role of the component creator.

8.1.9 Component version

In general, it is recommended to use a well-recognised versioning scheme. The aim is to have a monotonically strictly increasing sequence of version numbers. While the creator of the component can choose the versioning scheme completely arbitrarily, entities further downstream in the supply chain should not change it.

8.1.10 Relationship between URI, IRI and Punycode

Note: If only an Internationalized Resource Identifier (IRI) is available, it has to be transformed into a URI. For their fully qualified domain name (FQDN) part, Punycode[21] has to be used.

8.1.11 Scope of delivery

The scope of delivery of a software product comprises all parts of software, originated by the supplier or a third party, that are delivered with the software product. Not included by this term are parts of software that have to be acquired or obtained separately.

8.1.12 Path to component

A path to a component is defined as tuple of edges that connect nodes which lead to this specific component. It starts at the primary component and ends at this specific component. Note, that multiple paths to a single component exist if this component is depended upon by multiple, disjunct components. A path connects two nodes and may lead across other nodes; hence a path clearly denotes a specific chain of nodes. Consequently, if a component has *n* direct dependencies there are at least *n* paths across this node, each of them across a different direct dependency of this component. For visualisations see section 8.3.

8.1.13 Licence information

The required data fields for components include licence information. On the one hand handling licence information is a common use case for SBOMs. On the other hand, licence information can be crucial in handling vulnerabilities, e.g. if the licence of a component does not allow the user to modify the code to mitigate a vulnerability.

The original licence of an upstream component may differ from its distribution licence assigned by a creator of a downstream component utilising the upstream component. An example for this is if the primary licensee is forced by the component creator to choose from different sets of licences which are mutually exclusive. A classic example is Qt where the primary licensee has to decide between GPL and a proprietary licence; only the made choice can be handed further down the supply chain. Hence the distribution licences can differ from the original licences.

The statement in section 5.1 about multiple component instances with different meta-information also applies if only the licence information differs.

In general, the licence fields for both, “declared licenses” and “concluded licenses” should be filled according to the utilised format specification (i.e. a specific version of CycloneDX or SPDX). Note that these format specifications provide slightly deviating definitions for “declared license(s)” and “concluded license(s)”, though both specifications focus on the manner in which this information was obtained to differentiate their license fields. BSI TR-03183-2 rather focusses on the licence situation from the perspective of the SBOM creator up to the original source (i.e. the “tip” of the supply chain) and from the SBOM creator downwards in the supply chain (“downstream licence(s)”), because this is the relevant licence information for a recipient of a software (component) to determine their degree of freedom to use, analyse, alter, distribute it, and under which licence(s) a software (component) can be obtained from its original source.

Consequently, this Technical Guideline deliberately specifies different terms and definitions for licence information than both format specifications to avoid overloading the terms “declared license” and “concluded license” even more with a third definition. The licence information specified by BSI TR-03183-2 must be mapped to the corresponding format specification as follows, but mind that one should also avoid violating the definitions of these fields by the utilised format specification version.

[21] https://www.rfc-editor.org/rfc/rfc3492

8.1.14 Vulnerability information

The SBOM definition in this Technical Guideline states that vulnerability information is not contained in an SBOM. Information on vulnerabilities of a certain version of a software changes over time while the crucial information of an SBOM (e.g. on dependencies) is static. If vulnerability information is included in an SBOM, this static data is unnecessarily propagated along the software supply chain in unaltered form each time the vulnerability information is updated. Consequently, it is required not to include vulnerability information in an SBOM, even though an SBOM format specification supports that. The recommended format for distributing vulnerability information is CSAF (including also VEX as a profile).

8.1.15 Digital signature

Ideally, SBOMs should be digitally signed so recipients can verify their authenticity.

8.1.16 References to another BOM

If the creator of an SBOM uses references to an external BOM to reference SBOM information, this creator is responsible for the availability of the referenced SBOM information, e.g. by providing an own copy of this information. The goal is to provide the SBOM information as if the SBOM creator provided the complete information themselves.

8.2 Mapping of the individual data fields

The following tables contain mapping of the required information of the data fields specified in section 5 to the formats SPDX and CycloneDX. This mapping bears some challenges because the structures of the two formats differ. For example, SPDX offers an implicit differentiation between files and packages, whereas CycloneDX includes a component type. Both formats lack some of the values required by this Technical Guideline. To resolve this issue the respective data fields are represented by BSI's taxonomy[22] as key-value pairs.

Notes:

- Data fields required by the format specification (SPDX or CycloneDX) or other regulations shall remain. Additionally, data fields beyond these requirements and the requirements of this Technical Guideline may be included at the SBOM creator's convenience; this also covers additional instances of data fields, e.g. hash values.
- The "*OR*" and "*XOR*" statements in the following tables are directly derived either from "or" or "(n)either / (n)or" statements in the normative text in section 5.2 or from the corresponding SBOM format specification. For example, in CycloneDX data fields of the primary component are described at a different location in the tag hierarchy than other components.
- The "..." (ellipsis character) in the following tables is a placeholder for the values of the data fields which have to be inserted by the SBOM creator.
- The "*example_URI_xx*" in the following tables is a placeholder for "spdxId"s to indicate the connection between the SPDX elements. These "spdxId"s have to be inserted by the SBOM creator.

[22] https://github.com/BSI-Bund/tr-03183-cyclonedx-property-taxonomy

Table 8: Mapping of required data fields for the SBOM itself

Data field	***SPDX v3.0.1 (JSON)***	***CycloneDX v1.6 (JSON)***

Creator of the SBOM	{
	"type": "CreationInfo",
	"createdBy": ["*example_URI_01*"]
	},
	{
	"type": "Person",
	XOR
	"Organization",
	"spdxId": "*example_URI_01*",
	"externalIdentifers": [{
	"type": "ExternalIdentifier",
	"externalIdentifierType": "email",
	"identifier": "...@..."
	XOR
	"type": "ExternalIdentifier",
	"externalIdentifierType": "urlScheme",
	"identifier": "*https://...*"
	}]
	}
		{
		"metadata": {
		"manufacturer": [{
		"url": "..."
		XOR
		"contact" [{
		"email": "..."
		}]
		}]
		}
		}

Timestamp	{
	"type": "CreationInfo",
	"created": "..."
	}
		{
		"metadata": {
		"timestamp": "..."
		}
		}

Table 9: Mapping of required data fields for each component

Data field	***SPDX v3.0.1 (JSON)***	***CycloneDX v1.6 (JSON)***

Component creator	{
	"type": "software_Package",
	"originatedBy": ["*example_URI_01*"]
	},
	{
	"type": "Person",
	XOR
	"Organization",
	"spdxId": "*example_URI_01*",
	"externalIdentifers": [{
	"type": "ExternalIdentifier",
	"externalIdentifierType": "email",
	"identifier": "...@..."
	XOR
	"type": "ExternalIdentifier",
	"externalIdentifierType": "other",
	"identifier": "*https://...*"
	}]
	}
		{
		"metadata" {
		"component": {
		"manufacturer": [{
		"url": "..."
		XOR
		"contact" [{
		"email": "..."
		}]
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"manufacturer": [{
		"url": "..."
		XOR
		"contact" [{
		"email": "..."
		}]
		}]
		}]
		}

Component name	{
	"type": "software_Package",
	"name": "..."
	}
		{
		"metadata" {
		"component": {
		"name": "...",
		"group": "..."
		}
		}
		}
		XOR
		{
		"components": [{
		"name": "...",
		"group": "..."
		}]
		}
		Note: Please mind that "group" is not a required data field neither by this TR nor the CycloneDX specification; though in general it appears to be reasonable to distinguish between equally named components built by different projects.

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Component version	{
	"type": "software_Package",
	"software_packageVersion": "..."
	}
		{
		"metadata" {
		"component": {
		"version": "..."
		}
		}
		}
		XOR
		{
		"components": [{
		"version": "..."
		}]
		}

Filename of the component	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_File",
	"spdxId": "*example_URI_02*",
	"name": "..."
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasDistributionArtifact",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	}
		{
		"metadata" {
		"component": {
		"properties": [{
		"name": "bsi:component:filename",
		"value": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"properties": [{
		"name": "bsi:component:filename",
		"value": "..."
		}]
		}]
		}

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Dependencies on other components	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "contains",
	XOR
	"dependsOn",
	"to": [...]
	"completeness": "complete"
	XOR
	"incomplete"
	XOR
	"noAssertion"
	}
		{
		"dependencies": [{
		"ref": ...,
		"dependsOn": [...]
		}]
		},
		{
		"metadata" {
		"component": {
		"components": [{...}]
		}
		}
		},
		{
		"components": {
		"components": [{...}]
		}
		},
		{
		"compositions": [{
		"ref": ...,
		"aggregate": "complete",
		XOR
		"incomplete",
		XOR
		"unknown",
		"assemblies": [...],
		OR
		"dependencies": [...]
		}]
		}
		
		Note (see CycloneDX v1.6 specification)[23]: Components without any own "dependencies" MUST be declared as empty elements within the dependency graph.
		
		Note: "compositions" are only used to describe the completeness of "dependencies" and "assemblies".

[23] https://cyclonedx.org/docs/1.6/json/#dependencies

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Distribution licences	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasConcludedLicense",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	},
	{
	"type": "simpleLicensing_LicenseExpression",
	"spdxId": "*example_URI_02*",
	"licenseExpression": "..."
	}
		{
		"metadata" {
		"component": {
		"licenses": [{
		"expression": "...",
		"acknowledgement": "concluded"
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"licenses": [{
		"expression": "...",
		"acknowledgement": "concluded"
		}]
		}]
		}

Hash value of the deployable component	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_File",
	"spdxId": "*example_URI_02*",
	"verifiedUsing": [{
	"type": "Hash",
	"algorithm": "sha512",
	"hashValue": "..."
	}]
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasDistributionArtifact",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	}
		{
		"metadata" {
		"component": {
		"externalReferences": [{
		"type": "distribution",
		"hashes": [{
		"alg": "SHA-512",
		"content": "..."
		}]
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"externalReferences": [{
		"type": "distribution",
		"hashes": [{
		"alg": "SHA-512",
		"content": "..."
		}]
		}]
		}]
		}

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Executable property	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_File",
	"spdxId": "*example_URI_02*",
	"software_additionalPurpose": ["..."],
	"comment": "software_additionalPurpose field is used to indicate the properties of BSI TR-03183-2"
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasDistributionArtifact",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	}
	
	Note: Add “executable” to the “software_additionalPurpose” list if the component is executable; otherwise, do not add it.
		{
		"metadata" {
		"component": {
		"properties": [{
		"name": "bsi:component:executable",
		"value": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"properties": [{
		"name": "bsi:component:executable",
		"value": "..."
		}]
		}]
		}

Archive property	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_File",
	"spdxId": "*example_URI_02*",
	"software_additionalPurpose": ["..."],
	"comment": "software_additionalPurpose field is used to indicate the properties of BSI TR-03183-2"
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasDistributionArtifact",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	}
	
	Note: Add “archive” to the “software_additionalPurpose” list if the component is an archive; otherwise, do not add it.
		{
		"metadata" {
		"component": {
		"properties": [{
		"name": "bsi:component:archive",
		"value": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"properties": [{
		"name": "bsi:component:archive",
		"value": "..."
		}]
		}]
		}

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Structured property	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_File",
	"spdxId": "*example_URI_02*",
	"software_additionalPurpose": ["..."],
	"comment": "software_additionalPurpose field is used to indicate the properties of BSI TR-03183-2"
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasDistributionArtifact",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	}
	
	Note: Add “container” to the “software_additionalPurpose” list if the component is a structured file; otherwise, do not add it.
	
	Note: Add “firmware” to the “software_additionalPurpose” list if the component is not a structured file; otherwise, do not add it.
		{
		"metadata" {
		"component": {
		"properties": [{
		"name": "bsi:component:structured",
		"value": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"properties": [{
		"name": "bsi:component:structured",
		"value": "..."
		}]
		}]
		}

Table 10: Mapping of additional data fields for the SBOM itself

Data field	***SPDX v3.0.1 (JSON)***	***CycloneDX v1.6 (JSON)***

SBOM-URI	{
	"type": "SpdxDocument",
	"rootElement": "..."
	},
	{
	"type": "software_Sbom",
	"spdxId": "...",
	"rootElement": "*example_URI_01*"
	},
	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	}
	
	Note: The "software_Package" with the "spdxId" "*example_URI_01*" refers to the primary element of the SBOM.
		{
		"serialNumber": "..."
		}

Table 11: Mapping of additional data fields for each component

Data field	***SPDX v3.0.1 (JSON)***	***CycloneDX v1.6 (JSON)***

Source code URI	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_SoftwareArtifact",
	"spdxId": "*example_URI_02*",
	"software_primaryPurpose": "source",
	"externalRef": [{
	"type": "ExternalRef",
	"externalRefType": "SourceArtifact",
	"locator": "..."
	}]
	},
	{
	"type": "Relationship",
	"from": "*example_URI_02*",
	"relationshipType": "generates",
	"to": ["*example_URI_01*"],
	"completeness": "complete"
	}
		{
		"metadata" {
		"component": {
		"externalReferences": [{
		"type": "source-distribution",
		"url": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"externalReferences": [{
		"type": "source-distribution",
		"url": "..."
		}]
		}]
		}

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

URI of the deployable form of the component	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_File",
	"spdxId": "*example_URI_02*",
	"binaryArtifact": "..."
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasDistributionArtifact",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	}
		{
		"metadata" {
		"component": {
		"externalReferences": [{
		"type": "distribution",
		"url": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"externalReferences": [{
		"type": "distribution",
		"url": "..."
		}]
		}]
		}

Other unique identifiers	{
	"type": "software_Package"
	"externalIdentifiers": [{
	"externalIdentifierType": "cpe22",
	OR
	"cpe23",
	OR
	"swid",
	OR
	"packageURL",
	"identifier": "..."
	}]
	}
		{
		"metadata" {
		"component": {
		"cpe": "*cpe:/...*"
		OR
		"swid": {
		"tagId": "..."
		}
		OR
		"purl": "..."
		}
		}
		}
		XOR
		{
		"components": [{
		"cpe": "*cpe:/...*"
		OR
		"swid": {
		"tagId": "..."
		}
		OR
		"purl": "..."
		}]
		}

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Original licences	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "hasDeclaredLicense",
	"to": ["*example_URI_02*"],
	"completeness": "complete"
	},
	{
	"type": "simpleLicensing_LicenseExpression",
	"spdxId": "*example_URI_02*",
	"licenseExpression": "..."
	}
		{
		"metadata" {
		"component": {
		"licenses": [{
		"expression": "...",
		"acknowledgement": "declared"
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"licenses": [{
		"expression": "...",
		"acknowledgement": "declared"
		}]
		}]
		}

Table 12: Mapping of optional data fields for each component

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Effective licence	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "Relationship",
	"from": "*example_URI_01*",
	"relationshipType": "other",
	"to": ["*example_URI_02*"],
	"comment": "hasEffectiveLicense",
	"completeness": "complete"
	},
	{
	"type": "simpleLicensing_LicenseExpression",
	"spdxId": "*example_URI_02*",
	"licenseExpression": "..."
	}
		{
		"metadata" {
		"component": {
		"properties": [{
		"name":
		"bsi:component:effectiveLicense",
		"value": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"properties": [{
		"name": "bsi:component:effectiveLicense",
		"value": "..."
		}]
		}]
		}

Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

Hash value of the source code of the component	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_SoftwareArtifact",
	"spdxId": "*example_URI_02*",
	"software_primaryPurpose": "source",
	"verifiedUsing": [{
	"type": "Hash",
	"algorithm": "sha512",
	"hashValue": "..."
	}]
	},
	{
	"type": "Relationship",
	"from": "*example_URI_02*",
	"relationshipType": "generates",
	"to": ["*example_URI_01*"],
	"completeness": "complete"
	}
		{
		"metadata" {
		"component": {
		"externalReferences": [{
		"type": "source-distribution",
		"hashes": [{
		"alg": "SHA-512",
		"content": "..."
		}]
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"externalReferences": [{
		"type": "source-distribution",
		"hashes": [{
		"alg": "SHA-512",
		"content": "..."
		}]
		}]
		}]
		}

URL of the **security.txt**	{
	"type": "software_Package",
	"externalRef": [{
	"type": "ExternalRef",
	"externalRefType": "securityOther",
	"locator": "..."
	}]
	}
		{
		"metadata" {
		"component": {
		"externalReferences": [{
		"type": "rfc-9116",
		"url": "..."
		}]
		}
		}
		}
		XOR
		{
		"components": [{
		"externalReferences": [{
		"type": "rfc-9116",
		"url": "..."
		}]
		}]
		}

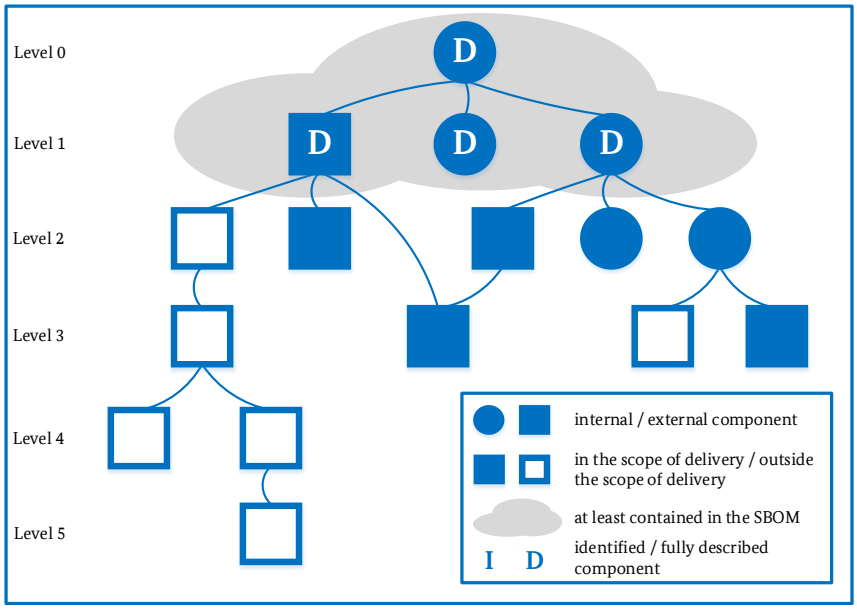
Data field	*SPDX v3.0.1 (JSON)*	*CycloneDX v1.6 (JSON)*

References to another BOM	{
	"type": "software_Package",
	"spdxId": "*example_URI_01*"
	},
	{
	"type": "software_Sbom",
	"spdxId": "*example_URI_02*",
	"externalRef": [{
	"type": "ExternalRef",
	"externalRefType": "buildMeta",
	"locator": "..."
	}]
	},
	{
	"type": "Relationship",
	"from": "*example_URI_02*",
	"relationshipType": "describes",
	"to": ["*example_URI_01*"],
	"completeness": "complete"
	}
		{
		"components": [{
		"externalReferences": [{
		"type": "bom",
		"url": "..."
		}]
		}]
		}

8.3 Level of detail of an SBOM

Structurally, an SBOM can be created in different levels of detail, e.g. according to the following classification.

8.3.1 Top-level SBOM



In addition to the full description of the primary component, the SBOM contains the full description of all components, which the primary component directly depends on.

Figure 1: Top-level SBOM

8.3.2 *n*-level SBOM

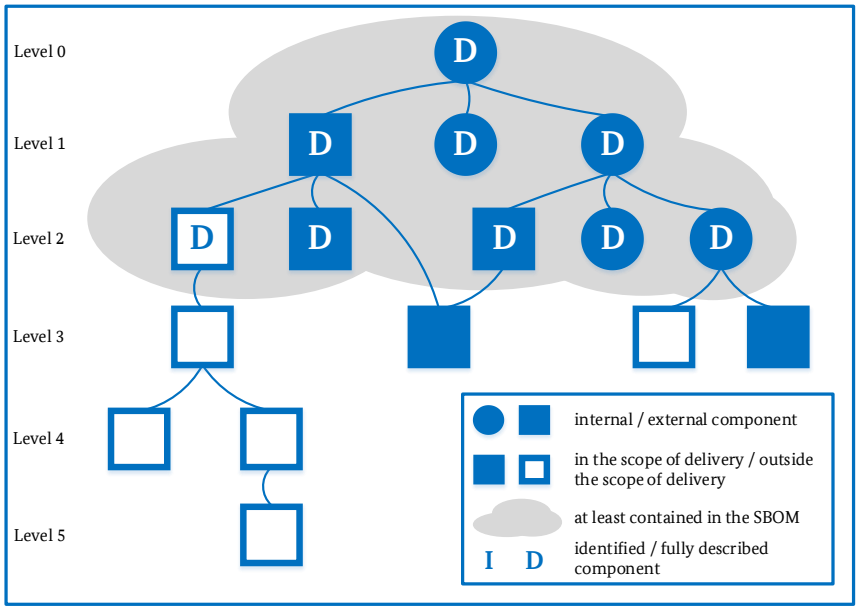


Figure 2: n-level SBOM for n=2

In addition to the full description of the primary component, the SBOM contains the full description of all components, which are directly or transitively depended upon via *n* levels by the primary component. This means that the recursive resolution of the transitive dependencies is limited to *n* steps in depth. If the path from the primary component is shorter than *n* levels, all components on this path have to be resolved and, consequently, fully described.

A top-level SBOM is a 1-level SBOM.

8.3.3 Transitive SBOM

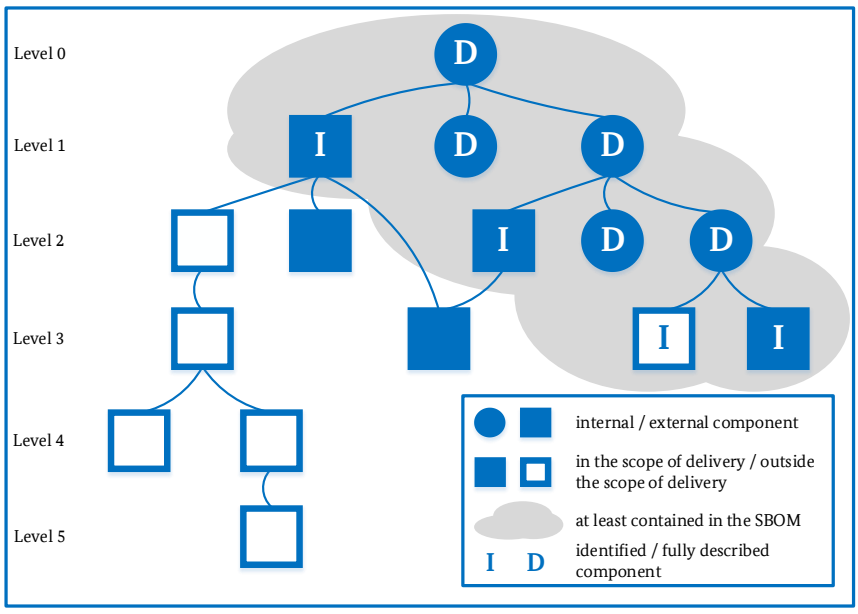


Figure 3: Transitive SBOM

In addition to the full description of the primary component, the SBOM contains information on at least all components, which are directly or transitively depended upon by the primary component. The full description and recursive resolution of components and their dependencies is performed **on each path** at least up to the first external component (i.e. third-party component). The first external component must at least be identified in the SBOM in order to correlate this dependency unambiguously; consequently, its dependencies in turn do not have to be resolved.

Compared to an n-level SBOM including a first external component the transitive SBOM may contain less information on this external component, because a transitive SBOM does not need to fully describe external components.

8.3.4 Delivery item SBOM

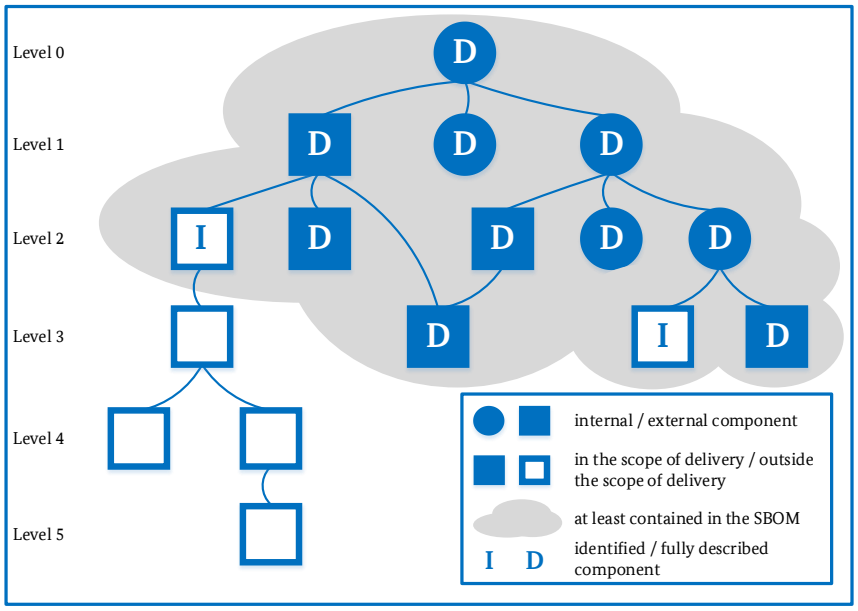


Figure 4: Delivery item SBOM

In addition to the full description of the primary component, the SBOM contains the full description of at least all components, which belong to the scope of delivery and are directly or transitively depended upon by the primary component. The full description and recursive resolution of components and their dependencies is performed **on each path** at least up to the first component, which is outside the scope of delivery. The first component outside the scope of delivery must at least be identified in the SBOM in order to correlate this dependency unambiguously; consequently, its dependencies in turn do not have to be resolved.

8.3.5 Complete SBOM

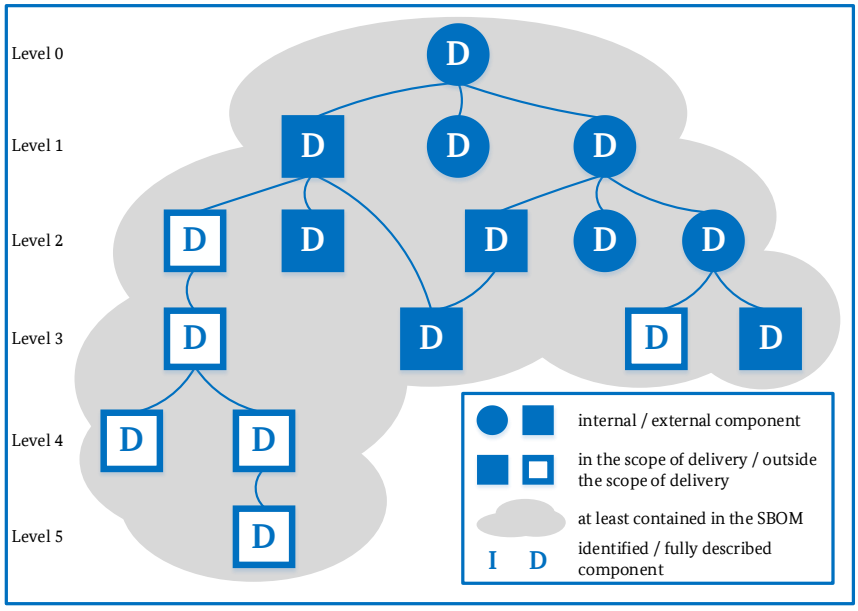


Figure 5: Complete SBOM

In addition to the full description of the primary component, the SBOM contains the full description of all components, which are directly or transitively depended upon by the primary component. The full description and recursive resolution of the components and their dependencies is carried out completely.

8.4 SBOM classification

Depending on how or when an SBOM is created as part of the development, delivery, installation and execution process of a component, the data differs, which is available in that specific situation. Consequently, the information that can be compiled in the SBOM also differs. A common differentiation is to distinguish between the following SBOM classes.

8.4.1 Design SBOM

The SBOM is created based on the planned set of included components of a new software artefact. The components do not have to exist yet.

8.4.2 Source SBOM

The SBOM is created from the development environment, the source files and the dependencies it uses.

8.4.3 Build SBOM

The SBOM is created as part of the build process based on e.g. source files, dependency information, already created components, volatile build process data and other SBOMs.

Notes:

- In order to enable capturing executable, binary components that already exist (e.g. precompiled code), creating a Build SBOM focuses on the linker run for translated code, not the compiler run.
- In order to let hash values unambiguously identify components, reproducible builds have to be employed.
- In the case of interpreted code, only the source code exists; hence each executable file has to be listed as a component. The interpreter has to be specified as a dependency, as far as reasonably possible.

8.4.4 Analysed SBOM

The SBOM is created after the build process by analysing artefacts such as executables, packages, containers and virtual machine images. This type is also referred to as “3rd party SBOM”.

8.4.5 Deployed SBOM

The SBOM provides an inventory of the software on a system. This can be a compilation of other SBOMs, taking into account configuration options and examination of execution behaviour in a (possibly simulated) deployment environment.

8.4.6 Runtime SBOM

The SBOM is created using the system executing the software to capture running (i.e. executing) components as well as their external calls and dynamically loaded components at runtime only (i.e. in memory). This type may also be referred to as “Dynamic SBOM”.

8.5 Further information

8.5.1 Information from the NTIA

The “National Telecommunications and Information Administration (NTIA)” of the “United States Department of Commerce” offers a great deal of further information on the subject of SBOM at https://ntia.gov/sbom.

8.5.2 Information from CISA

The Cybersecurity and Infrastructure Security Agency (CISA) of the United States Department of Homeland Security also offers further information on SBOM at https://cisa.gov/sbom.

8.5.3 Information from CycloneDX

CycloneDX also offers guides and resources for beginners at https://cyclonedx.org/guides/.

8.5.4 Information from SPDX

SPDX also offers further information on producing and/or consuming SPDX documents and SBOMs at https://spdx.dev/learn/overview/.