Design Document

Design Document: SBOM Conformance Checker 2025

Document history

Version	Date	Description
0.1	5 Nov 2024	Scope and expected command options
1.0	26 Nov 2024	Implementation proposal for FSCT v3 support
1.0.1	11 Feb 2025	Update implementation status / Prepare for GSoC 2025 application
1.1	11 July 2025	Add SPDX 3.0 mapping to NTIA and FSCT v3 Minimum Expected Elements

Scope

- Add support for FSCT v3 at data maturity level “Minimum Expected” [DONE]
    - https://github.com/spdx/ntia-conformance-checker/pull/224
- Maintain support for NTIA Minimum Requirements/FSCT v2 [DONE]
- Maintain support for SBOMs in SPDX 2.2, 2.2.1 and 2.3
- Add support of SPDX 3.0 [GSoc 2025]
    - https://github.com/spdx/ntia-conformance-checker/issues/248
- Add support for EU AI Act through fields in SPDX AI and Dataset Profiles [GSoC 2025]

Expanded Scope

- Add support for TR-03183-2 v2.0.0
- Add support for FSCT v3 at data maturity levels “Recommended Practice” and “Aspirational Goal”

*Note that **sbomqs** already support FSCT v3 and TR-03183-2 for SPDX 2.2 and 2.3 documents. Therefore, it's important to clarify the specific need for this work, whether it aims to address gaps in existing support or provide additional features.*

Current command-line options

```
Usage: ntia-checker [OPTIONS]
```

```
Options:
  --file TEXT             The file to be parsed
  --output [print|json]  Output format  [default: print]
  -v, --verbose           Use verbose printing
  --output_path TEXT     Filepath for optionally storing output.
  -h, --help              Show this message and exit.
```

Proposed command-line options

```
Usage: ntia-checker [OPTIONS]

Options:
  --standard [fsct2|fsct3-min|fsct3-rec|fsct3-asp|bsi2.0.0|ntia]
                          Requirements to check against
  --file TEXT             The file to be parsed
  --output [print|json]  Output format  [default: print]
  -v, --verbose           Use verbose printing
  --output_path TEXT     Filepath for optionally storing output.
  -h, --help              Show this message and exit.
```

Current design

- Most of the work is done in ntia_conformance_checker/sbom_checker.py with ntia_conformance_checker/main.py works as an entrypoint/dispatcher from the command line.
- Compliance checker is done within SbomChecker class in sbom_checker.py
- When instantiating an instance from the class, the __init__ method will start the compliance check immediately.
- The True/False value of self.ntia_minimum_elements_compliant relies on the True/False testing of these value/return value:
    - self.doc_author
    - self.doc_timestamp
    - validate_full_spdx_document(self.doc) (from spdx_tools.spdx.validation.document_validator)
    - self.check_dependency_relationships()
    - self.get_components_without_names()
    - self.get_components_without_versions()
    - self.get_components_without_suppliers()
    - self.get_components_without_identifiers()
- All of the check logic sits in self.check_ntia_minimum_elements_compliance(), which uses information collected during instantiation.

New design proposals

- A new compliance checker can be added by either 1) adding a new method similar to self.check_ntia_minimum_elements_compliance() or 2) adding a new class similar to SbomChecker.
    - First approach: SbomCheck.check_ntia_minimum_elements_compliance(), SbomCheck.check_fsct3_minimum_elements_compliance(), etc.
    - Second approach: NTIAChecker.check_compliance(), FSCT3Checker.check_compliance(), FSCT3Checker.check_compliance(level="aspirational"), EUAIAChecker.check_compliance(), etc. (and turn SbomChecker to a superclass)
    - The first approach may not work for the input from different structures, unless we modify the self.parse_file() method to handle multiple formats.
    - The second approach can be designed in such a way that one class will handle one format.
    - The second approach may be more modular and allow us to add/remove checkers more cleanly.
- It may be possible to extract few parsing and printing methods out of SbomChecker class. Make them shared static methods that can be used by other checkers. Or, alternatively, make them abstract/interface and let subclasses implement them, to ensure that essential methods will be implemented and use shared names.
- It may also be possible to decouple further the list of requirements from the code, by having the list of requirements as a CSV, a YAML or other similar machine-readable file, and let the SbomChecker (or its subclass) read from it. The file will basically be a mapping between a requirement in a standard and a field in SBOM. This may require additional implementation that is not clear yet if it is worth the effort.

  Example of what requirement YAML file can look like:

```
name: NTIA Minimum Elements for an SBOM
id: ntia
checker-class: SbomChecker

  requirements:
  - name: Author of SBOM Data
      description: The name of the entity that creates the SBOM [..]
      checker: doc_author

  - namefield: Timestamp
      description: Record of the date and time of the SBOM data [..]
      checker: doc_timestamp

  - field: Component Name
      description: Designation assigned to a unit of software [..]
      checker: has_name_in_all_components()
```

```
  name: NTIA Minimum Elements for an SBOM
  id: fsct3
  checker-class: SbomChecker

    requirements:
    - name: Author Name
        description: The Author Name is intended to be the name [..]
        checker: doc_author
        level: minimum

    - name: Timestamp
        description: The Timestamp is the date and time that the [..]
        checker: doc_timestamp
        level: minimum

    - name: Type
        description: The Type Attribute provides context for how [..]
        checker: doc_type
        level: aspirational
```

Example of approach #1

```
class SbomChecker:
    def __init__:
        if standard_1:
            do_standard_1_thing
        elif standard_2:
            do_standard_2_thing
        ...

    def parse_file:
        if format_1:
            do_format_1_thing
        elif format_2:
            do_format_2_thing
        ...

    def check_doc_version:
        if format_1:
            ...
        elif format_2:
            ...

    def check_standard_1_compliance:
```

```
        ...

    def check_standard_2_compliance:
        ...

    def print_table_output:
        if standard_1:
            do_standard_1_thing
        elif standard_2:
            do_standard_2_thing
        ...
```

Example of approach #2

```
class Standard1Checker:
    def __init__:
        do_standard_1_thing

    def parse_file:
        if format_1:
            do_format_1_thing
        elif format_2:
            do_format_2_thing
        ...

    def check_doc_version:
        if format_1:
            ...
        elif format_2:
            ...

    def check_compliance:
        ...

    def print_table_output:
        do_standard_1_thing


class Standard2Checker:
    def __init__:
        do_standard_2_thing

    def parse_file:
        if format_1:
            do_format_1_thing
```

```
        elif format_2:
            do_format_2_thing
        ...

    def check_doc_version:
        if format_1:
            ...
        elif format_2:
            ...

    def check_compliance:
        ...

    def print_table_output:
        do_standard_2_thing
```

If we like to drop support of Standard 2 (code highlighted with green), for example, with Option #2, largely we will remove the entire class Standard2Checker and the remaining work will be inside the dispatcher class/dispatcher command-line parser that will reject the call of the standard and inform the user.

For Option #1, as lines of codes of different standards interleaved, it may take a little more effort to go around. It's not difficult, but not as clean when compare with Option #2 which each standard tends to have continuous lines of code.

Methods like parse_file() and potentially check_doc_version() can be refactored and relocated away from the checker class for both Options #1 and #2. But it will benefit Option #2 more because Option #2 is more likely to have redundant code in these methods.

Data field mapping

NTIA Minimum Elements <--> SPDX 2.3 <--> SPDX 3.0

SPDX 2.3 mapping is from SPDX 2.3 Section K.2.2 "Mapping NTIA Minimum Elements to SPDX Fields".

The SPDX Specification contains fields able to address each of the NTIA minimum required data fields.

NTIA SBOM Minimum Field	**Satisfying SPDX 2.3 field**	**Satisfying SPDX 3.0 field**
Author Name	(6.8) Creator	SpdxDocument.creationInfo.createdBy.name

NTIA SBOM Minimum Field	Satisfying SPDX 2.3 field	Satisfying SPDX 3.0 field
Supplier Name	(7.5) Package Supplier	Package.suppliedBy.name
Component Name	(7.1) Package Name	Package.name
Version String	(7.3) Package Version	Package.packageVersion
Component Hash	(7.10) Package Checksum	Package.verifiedUsing.hashValue (verifiedUsing should be of type Hash)
Unique Identifier	(7.2) Package SPDX Identifier (6.5) SPDX Document Namespace	Package.spdxId
Relationship	(11.1) Relationship: CONTAINS, DESCRIBES The document must DESCRIBES at least one package.	SpdxDocument.rootElement is Sbom Sbom must contain at least one Package
Timestamp	(6.9) Created	SpdxDocument.creationInfo.created

FSCT v3 Minimum Expected Elements <--> SPDX 2.3 <--> SPDX 3.0

A set of Baseline Attributes is defined in Section 2.2 of Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM) Third Edition. There are three maturity levels (Minimum Expected, Recommended Practice, and Aspirational Goal) for content provided in Attribute entries. The table below shows Attributes with “Minimum Expected” maturity level.

FSCT v3 Baseline Attributes, Minimum Expected	Satisfying SPDX 2.3 field	Equivalent NTIA SBOM Minimum Field	Satisfying SPDX 3.0 field
(2.2.1) SBOM Meta-Information			
(2.2.1.1) Author Name	(6.8) Creator	Author Name	SpdxDocument.creationInfo.createdBy.name

An SBOM must list the entity that prompted the creation of the SBOM.			
(2.2.1.2) Timestamp the Timestamp should be consistent across time zones and locales and use a common international format, such as ISO 8601	(6.9) Created	Timestamp	SpdxDocument.creationInfo.created
(2.2.1.4) Primary Component (or Root of Dependencies) The Primary Component, or root of Dependencies, is the subject of the SBOM or the foundational Component being described in the SBOM.	SPDX document		SpdxDocument.rootElement.rootElement - SpdxDocument.rootElement should be Sbom. - The rootElement of that Sbom should be Package. - That Package is considered as Primary Component
(2.2.2) Component Attributes			(for each Component, not only the Primary Component)
(2.2.2.1) Component Name the Component name should declare the commonly used public name for the Component	(7.1) Package Name	Component Name	Package.name
(2.2.2.2) Version declare the version string as provided by the Supplier	(7.3) Package Version	Version String	Package.packageVersion
(2.2.2.3) Supplier Name the Supplier Name should be declared for all Components	(7.5) Package Supplier	Supplier Name	Package.suppliedBy.name

(2.2.2.4) Unique Identifier at least one unique identifier should be declared for each Component listed in the SBOM. A globally unique identifier is preferred.	(7.2) Package SPDX Identifier (6.5) SPDX Document Namespace	Unique Identifier	Package.spdxId
(2.2.2.5) Cryptographic Hash Provide a hash for any Component listed in the SBOM for which the hash was provided or sufficient information is available to generate the hash. If sufficient information is not available, indicate as unknown.	(7.10) Package Checksum	Component Hash	Package.verifiedUsing.hashValue (verifiedUsing should be of type Hash)
(2.2.2.6) Relationship Relationship and relationship completeness declared for the Primary Component and direct Dependencies	(11.1) Relationship: CONTAINS, DESCRIBES The document must DESCRIBES at least one package.	Relationship	SpdxDocument.rootElement is Sbom Sbom must contain a Package as its rootElement
(2.2.2.7) License Provide license information for the Primary Component	(7.13) Concluded License (7.15) Declared License		SpdxDocument.rootElement.rootElement (the rootElement of the root Sbom under the SpdxDocument) must have `hasDeclaredLicense` and `hasConcludedLicense` relationships. SPDX 3.0 specific conformance: - If the `hasConcludedLicense` for a Software Artifact (including Package) is not the same as its `hasDeclaredLicense`, a

			written explanation SHOULD be provided in the `hasConcludedLicense` relationship comment field.
(2.2.2.8) Copyright Notice Provide copyright notice for the Primary Component	(7.17) Copyright Text		SpdxDocument.rootElement.rootElement.copyrightText or SimpleLicensingText.licenseText or License.licenseText (from `hasDeclaredLicense` or `hasConcludedLicense` relationship)

References

FSCT v1 — Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM) https://www.ntia.gov/sites/default/files/publications/framingsbom_20191112_0.pdf

FSCT v2 — Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM) Second Edition https://www.ntia.gov/files/ntia/publications/ntia_sbom_framing_2nd_edition_20211021.pdf

FSCT v3 — Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM) Third Edition https://www.cisa.gov/resources-tools/resources/framing-software-component-transparency-2024

Issue #214 — Update to new edition of minimum elements (2024) https://github.com/spdx/ntia-conformance-checker/issues/214 A proposal to update the NTIA Conformance Checker tool to support checking of Framing Software Component Transparency (FSCT) v3 Baseline Attributes (Section 2.2).

NTIA	The Minimum Elements For a Software Bill of Materials (SBOM): Pursuant to Executive Order 14028 on Improving the Nation's Cybersecurity https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom
ntia-conformance-checker	NTIA Conformance Checker https://github.com/spdx/ntia-conformance-checker
sbomqs	sbomqs: Quality metrics for SBOMs https://github.com/interlynk-io/sbomqs
spdx-python-model	Python library to read SPDX 3 JSON and produce SPDX 3 objects https://github.com/spdx/spdx-python-model
spdx-tools	Python library to parse, validate and create SPDX documents https://github.com/spdx/tools-python
TR-03183-2 v2.0.0	Technical Guideline TR-03183: Cyber Resilience Requirements for Manufacturers and Products - Part 2: Software Bill of Materials (SBOM) Version 2.0.0 https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03183/TR-03183_node.html