Software that conforms to the SPDX specification at the extension profile compliance point shall be able to import and export serialized documents that conform with one of the SPDX serialization formats defined SPDX serialization formats, including the abstract Extension class serving as the base for all defined extension subclasses.

Conformance to the extension profile compliance point does not entail support for the Licencing, Security, Data Set, AI,

Build, or profiles of the SPDX but is expected to be used in combination with the other profiles to extend them.

This compliance point facilitates interchange of extended information that goes beyond the standard SPDX produced by tools supporting SPDX and is used between cooperating parties that understand the form of the extension and can produce and consume its non-standard content.

3 References

3.1 Normative References

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Apache Maven, Apache Software Foundation, https://maven.apache.org/

Bower API, https://bower.io/docs/api/#install

Common Platform Enumeration (CPE) – Specification, The MITRE Corporation, https://cpe.mitre.org/files/cpe-specification_2.2.pdf

NISTIR 7695, Common Platform Enumeration: Naming Specification Version 2.3, NIST, https://csrc.nist.gov/publications/detail/nistir/7695/final

npm-package.json, npm Inc., https://docs.npmjs.com/files/package.json

NuGet documentation, Microsoft, https://docs.microsoft.com/en-us/nuget/

POSIX.1-2017 The Open Group Base Specifications Issue 7, 2018 edition, IEEE/Open Group, https://pubs.opengroup.org/onlinepubs/9699919799/

purl (package URL), https://github.com/package-url/purl-spec

Resource Description Framework (RDF), 2014-02-25, W3C, http://www.w3.org/standards/techs/rdf

RFC-1321, The MD5 Message-Digest Algorithm, The Internet Society Network Working Group, https://tools.ietf.org/html/rfc1321

RFC-3174, US Secure Hash Algorithm 1 (SHA1), The Internet Society Network Working Group, https://tools.ietf.org/html/rfc3174

RFC-3986, Uniform Resource Identifier (URI): Generic Syntax, The Internet Society Network Working Group, https://tools.ietf.org/html/rfc3986

RFC-5234, Augmented BNF for Syntax Specifications: ABNF, The Internet Society Network Working Group, https://tools.ietf.org/html/rfc5234

RFC
Wor

Soft
iden

SPD

5 Model and serializations

5.1 Overview

This specification defines the data model of the SPDX standard, describing every piece of information about systems with software components. The data model is based on the Resource Description Framework (RDF) extensible knowledge representation data model, which provides a flexible and extensible way to represent and exchange information.

The data may be serialized in a variety of formats for storage and transmission.

5.2 RDF serialization

Since the data model is based on RDF, any SPDX data can be serialized in any of the multiple RDF serialization formats, including but not limited to:

- JSON-LD format as defined in JSON-LD 1.1;
- Turtle (Terse RDF Triple Language) format as defined in RDF 1.1 Turtle;
- N-Triples format as defined in RDF 1.1 N-Triples; and
- RDF/XML format as defined in RDF 1.1 XML Syntax.

The SPDX specification is accompanied by a JSON-LD context definition file that can be used to serialize SPDX in a much simpler and more human-readable JSON-LD format.

continued

5.3 Canonical serialization

Canonical serialization is a single, consistent, normalized, deterministic, and reproducible form.

Such a canonical form normalizes things like ordering and formatting.

The content of the canonical serialization is exactly the same as the JSON-LD serialization of RDF data (see 4.2), just represented in a consistent way.

Canonical serialization is in JSON format, as defined in RFC 8259 (IETF STD 90), with the following additional characteristics:

- No line breaks
- Key names MUST be wrapped in double quotes
- No whitespace outside of strings
- `true`, `false` and `null`: the literal names must be lowercase; no other literal names are allowed
- Integers: represented in base 10 using decimal digits. This designates an integer component that may be prefixed with an optional minus sign. Leading zeros are not allowed.
- Strings: UTF-8 representation without specific canonicalisation. A string begins and ends with quotation marks (%x22). Any Unicode characters may be placed within the quotation marks, except for the two characters that MUST be escaped by a reverse solidus: quotation mark, reverse solidus, and the control characters (U+0000 through U+001F).
- Arrays: An array structure is represented as square brackets surrounding zero or more items. Items are separated by commas.
- Objects: An object structure is represented as a pair of curly brackets surrounding zero or more name/value pairs (or members). A name is a string containing only ASCII characters (0x21-0x7F). The names within an object must be unique. A single colon comes after each name, separating the name from the value. A single comma separates a value from a following name. The name/value pairs are ordered by name.

5.4 Serialization information

A collection of elements may be serialized in multiple formats.

An SpdxDocument element represents a collection of elements across all serialization data formats within the model.

The actual serialized bytes is represented by an Artifact element within the model.

A Relationship of type serializedInArtifact links an SpdxDocument to one or more serialized forms of itself.

When serializing a physical SpdxDocument, any property of the logical element that can be natively represented within the chosen serialization format (e.g., `@context` prefixes in JSON-LD instead of the namespaceMap) may utilize these native mechanisms. All remaining properties shall be serialized within the SpdxDocument element itself.

A serialization must not contain more than one SpdxDocument.

A given instance of serialization must not define more than one SpdxDocument element.

5.5 Serialization in JSON-LD

5.5.1 JSON-LD context file

JSON-LD contexts allow JSON documents to use simple, human-readable, locally defined terms while ensuring data interoperability across different systems.

The SPDX global JSON-LD context file must be used universally for all SPDX documents in JSON-LD format that adhere to a specific SPDX version.

SPDX global JSON-LD context file is available at: https://spdx.org/rdf/3.0.1/spdx-context.jsonld

All SPDX documents in JSON-LD format must include a reference to the SPDX global context file at the top level. This reference is achieved using the following JSON construct:

```
"@context": "https://spdx.org/rdf/3.0.1/spdx-context.jsonld"
```

The SPDX context file defines aliases for specific JSON-LD properties to improve compatibility with the SPDX model. These aliases are:

- `spdxId`: An alias for the `@id` property.
- `type`: An alias for the `@type` property.

5.5.2 JSON-LD validation

An SPDX serialization in JSON-LD format is considered conformant to the SPDX specification if it adheres to the following two validation criteria:

- Structural validation: The JSON-LD document must structurally validate against the SPDX JSON Schema. This schema defines the expected structure of the JSON-LD document, including the required elements, data types, and permissible values.
- Semantic validation: The JSON-LD document must successfully validate against the SPDX OWL ontology. This ontology defines the expected relationships and constraints between SPDX elements. The SPDX OWL ontology also incorporates SHACL shape restrictions to further specify these constraints.

The SPDX JSON Schema is available at: https://spdx.org/schema/3.0.1/spdx-json-schema.json

The SPDX OWL ontology is available at: https://spdx.org/rdf/3.0.1/spdx-model.ttl