**UN/CEFACT**

United Nations

Centre for Trade Facilitation and Electronic Business (UN/CEFACT)

Methodology and TECHNOLOGY PROGRAMME Development Area Specifications Domain

JSON Schema Naming and Design Rules Technical Specification

**SOURCE:** API TechSpec Project Team

**ACTION:** Ready for publication

**DATE:** 13 September 2022

**STATUS: v1.0**

**Disclaimer (Updated UN/CEFACT Intellectual Property Rights Policy – ECE/TRADE/C/CEFACT/ 2010/20/Rev.2)**

ECE draws attention to the possibility that the practice or implementation of its outputs (which include but are not limited to Recommendations, norms, standards, guidelines and technical specifications) may involve the use of a claimed intellectual property right.

Each output is based on the contributions of participants in the UN/CEFACT process, who have agreed to waive enforcement of their intellectual property rights pursuant to the UN/CEFACT IPR Policy (document ECE/TRADE/C/CEFACT/2010/20/Rev.2 available at <http://www.unece.org/cefact/cf_docs.html> or from the ECE secretariat). ECE takes no position concerning the evidence, validity or applicability of any claimed intellectual property right or any other right that might be claimed by any third parties related to the implementation of its outputs. ECE makes no representation that it has made any investigation or effort to evaluate any such rights.

Implementers of UN/CEFACT outputs are cautioned that any third-party intellectual property rights claims related to their use of a UN/CEFACT output will be their responsibility and are urged to ensure that their use of UN/CEFACT outputs does not infringe on an intellectual property right of a third party.

ECE does not accept any liability for any possible infringement of a claimed intellectual property right or any other right that might be claimed to relate to the implementation of any of its outputs.

## Abstract

This JSON Schema Naming and Design Rules technical specification defines an architecture and a set of rules necessary to define, describe and use JSON to consistently express business information exchanges namely via APIs. It is based on the JSON Schema team’s specification and the UN/CEFACT Core Components Technical Specification. This specification will be used by UN/CEFACT to define JSON Schema and JSON Schema documents, which will be published as UN/CEFACT standards. It will also be used by other organisations who are interested in maximizing inter- and intra-industry interoperability.

この JSON Schema Naming and Design Rules 技術仕様は、JSON を定義、記述、および使用してビジネス情報交換を API 経由で一貫して表現するために必要なアーキテクチャと一連のルールを定義します。 これは、JSON スキーマ チームの仕様と UN/CEFACT コア コンポーネント技術仕様に基づいています。 この仕様は UN/CEFACT によって使用され、UN/CEFACT 標準として公開される JSON スキーマおよび JSON スキーマ ドキュメントを定義します。 また、業界間および業界内の相互運用性を最大化することに関心のある他の組織によっても使用されます。

[Abstract 2](#_bookmark0)

* 1. [Document History 5](#_bookmark1)
  2. [Change Log 5](#_bookmark2)
  3. [JSON Schema Naming and Design Rules Project Team 7](#_bookmark3)
  4. [Acknowledgements 7](#_bookmark4)
  5. [Contact information 7](#_bookmark5)
  6. [Notation 7](#_bookmark6)
  7. [Audience 8](#_bookmark8)

1. [INTRODUCTION 9](#_bookmark9)
   1. [Objectives 9](#_bookmark10)
   2. [Requirements 9](#_bookmark11)
   3. [Dependencies 9](#_bookmark12)
   4. [Caveats and Assumptions 9](#_bookmark13)
   5. [Guiding Principles 10](#_bookmark14)
   6. [Conformance 10](#_bookmark15)
2. [JSON SCHEMA ARCHITECTURE 12](#_bookmark16)
   1. [Basic architecture 12](#_bookmark17)
      1. [JSON serialization in a RESTful context 12](#_bookmark18)
      2. [Overall JSON Schema Structure 12](#_bookmark19)
   2. [Versioning and "$id" 13](#_bookmark24)
   3. [General naming rules moving from CCTS to JSON 15](#_bookmark29)
   4. [JSON SCHEMA LANDSCAPE 17](#_bookmark33)
   5. [Data types 18](#_bookmark34)
      1. [Primitive Data Types 18](#_bookmark35)
      2. [Approved Core Component Types 19](#_bookmark39)
      3. [Unqualified Data Types 19](#_bookmark40)
      4. [Qualified Data Types for Date and Time 26](#_bookmark43)
      5. [Other Qualified Data Types 30](#_bookmark45)
   6. [Restriction and Extension 35](#_bookmark48)
      1. [Restriction 35](#_bookmark49)
      2. [Extension 38](#_bookmark50)
      3. [Publication and reusing contextualization 38](#_bookmark51)
   7. [ABIE and BBIE representation in JSON Schema 41](#_bookmark52)
      1. [General handling of ABIEs and BBIEs 41](#_bookmark53)
      2. [ASBIE representation in JSON Schema supporting document based and](#_bookmark55) [resource-based information 42](#_bookmark55)
   8. [Fostering implementation 44](#_bookmark56)
      1. [Compatibility with JSON schema draft before version 2020-12 44](#_bookmark57)
      2. [Hints for tool developers and designers when specifying real-life guidelines 45](#_bookmark58)
      3. [Referencing the Github Repository in an OpenAPI specification 48](#_bookmark59)
3. [APPENDIX A: EXAMPLES 49](#_bookmark60)
4. [APPENDIX B: NAMING AND DESIGN RULES LIST 50](#_bookmark61)
5. [APPENDIX C: GLOSSARY 56](#_bookmark62)

### Document History

### Change Log

**Table 1 – Document history**

|  |  |  |
| --- | --- | --- |
| **Phase** | ***Status*** | ***Date Last Modified*** |
| Draft development | First draft | 17 Dec 2021 |
| Publication | Publication | 13 Sep 2022 |

The change log is designed to alert users about significant changes that occurred during the development of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Date of Change** | **Version** | **Paragraph Changed** | **Summary of Changes** |
| 24 Jan 2022 | 0.2 | 3 | Adding rules for basic data types |
| 25 Jan 2022 | 0.3 | 3 |  |
| 08 Feb 2022 | 0.4 | 3.6 | Extensions, Restrictions, ABIEs, QDTs |
| 17 Feb 2022 | 0.5 | 5 | Adding rules list into appendix B |
| 22 Feb 2022 | 0.5 | 3.2, 3.4, 3.5 | JSON schema versioning Date Time qDT  Identification Schemes part of qDT Note on quantity unit of Rec20+21 JSON schema structure |
| 14 Mar 2022 | 0.6 | 3.3 R 13 | Handling of hard spaces |
|  |  | 3.5.4 | Adjusted to modifications in next chapter |
|  |  | 3.5.5 | Modified code and identifier list export |
|  |  | 3.6.1 | Added example for lower layer |
|  |  |  | restriction |
|  |  | 3.6.3 New R36, | New chapter about contextualisation |
|  |  | higher rules |  |
|  |  | renumbered |  |
|  |  | 3.7 R 37 | Deprecated ABIEs |
| 21 Mar 2022 | 0.7 | R9 | Handling of $id |
|  |  | R28 | Placement of code list files |
|  |  | 3.6.3 | Explanation of Export methods |
| 30 Mar 2022 | 0.8 | R 12ff.  Table 8  R 39 | Adding new R 12 to R 14 for the origin of JSON schema names.  Adjusted export options  New R 39 for UN/CEFACT publication |
| 05 Sep 2022 | 0.9 | 1.4  3.1.1  3.1.2  R 40  3.8  4  5  New: | Consideration of comments from the public review and minor corrections |

|  |  |  |  |
| --- | --- | --- | --- |
| **Date of Change** | **Version** | **Paragraph Changed** | **Summary of Changes** |
|  |  | R 4  R 46, R47 |  |
| 13 Sep 2022 | 1.0 | Table 6  3.8.3  R7, R10 | Compatibility for base64 encoding Referencing Github  Naming of schemas Minor changes |

**Table 2 - Document change log**

### JSON Schema Naming and Design Rules Project Team

We would like to recognize the following for their significant participation in the development of this Unites Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) JSON Schema Naming and Design Rules technical specification.

|  |  |
| --- | --- |
| **ATG2 Chair** |  |
| Marek Laskowski |  |
| **Project Lead** |  |
| Jörg Walther |  |
| **Lead editors** |  |
| Andreas Pelekies | Gerhard Heemskerk |

### Acknowledgements

This version of UN/CEFACT JSON Schema Naming and Design Rules Technical Specification has been created to foster convergence among Standards Development Organisations (SDOs). It has been developed in close coordination with these organisations:

* + - DCSA
    - GS1
    - Odette

### Contact information

ATG2 – Marek Laskowski, [Marek.laskowski@gmail.com](mailto:Marek.laskowski@gmail.com) NDR Project Lead – Jörg Walther, [jwalther@odette.org](mailto:jwalther@odette.org) Editor – Andreas Pelekies, [Andreas@pelekies.de](mailto:Andreas@pelekies.de)

Editor – Gerhard Heemskerk, [Gerhard.heemskerk@kpnmail.nl](mailto:Gerhard.heemskerk@kpnmail.nl)

### Notation

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,

SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this specification, are to be interpreted as described in Internet Engineering Task Force (IETF) Request For Comments (RFC) 2119[1](#_bookmark7).

1 Key words for use in RFCs to Indicate Requirement Levels - Internet Engineering Task Force, Request For Comments 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt?number=2119>

A representation of a definition or a rule. Examples are informative.

Example

Explanatory information. Notes are informative.

[Note]

Identification of a rule that requires conformance. Rules are normative. In order to ensure continuity across versions of the specification, rule numbers “n” are randomly generated. The number of a rule that is deleted will not be re-issued. Rules that are added will be assigned a previously unused random number.

[R n|c]

The second number “c” after the pipe symbol identifies the conformance

|

category of the given rule as defined in section [2.6](#_bookmark15) [Conformance](#_bookmark15).

**Courier** All words appearing in **bolded courier font** are values, objects or keywords. Representation of non-printable characters like whitespace are surrounded by double-quotes, e.g. " ".

**<<var>>** All placeholders are surrounded by double less-than and greater-than characters. The meaning of the placeholder is described in the text.

### Audience

The audience for this UN/CEFACT JSON Schema Naming and Design Rules Technical Specification is:

* + - Members of the UN/CEFACT Applied Technologies Groups who are responsible for development and maintenance of UN/CEFACT JSON Schema.
    - The wider membership of the other UN/CEFACT Groups who participate in the process of creating and maintaining UN/CEFACT JSON Schema definitions.
    - Designers of tools who need to specify the conversion of user input into JSON Schema definitions adhering to the rules defined in this document.
    - Designers of JSON Schema definitions outside of the UN/CEFACT Forum community. These include designers from other organisations that have found these rules suitable for their own organisations.

# Introduction

### Objectives

This JSON Schema NDR technical specification document forms part of a suite of documents that aim to support modern web developers to make use of UN/CEFACT semantics.

この JSON スキーマ NDR 技術仕様ドキュメントは、最新の Web 開発者が UN/CEFACT セマンティクスを利用できるようにサポートすることを目的としたドキュメント スイートの一部です。

It can be applied on any layer of the UN/CEFACT Reference Data Models to create conformant JSON artefacts in accordance with the UN/CEFACT Core Components Technical Specification Version 2.01. This includes comprehensive RDMs like Buy-Ship- Pay, or Accounting as well as their contextualization like the Supply-Chain-Reference- Data-Model (SCRDM), Multi-Modal-Transport-Reference-Data-Model (MMT-RDM) down to single message implementation like the Road Consignment Note (eCMR) or the certificate of origin (COO)

. UN/CEFACT リファレンス データ モデルの任意のレイヤーに適用して、UN/CEFACT コア コンポーネント技術仕様バージョン 2.01 に従って適合する JSON アーティファクトを作成できます。 これには、Buy-Ship-Pay や Accounting などの包括的な RDM と、Supply-Chain-Reference-Data-Model (SCRDM)、Multi-Modal-Transport-Reference-Data-Model (MMT-RDM) などのコンテキスト化が含まれます。 Road Consignment Note (eCMR) や Certificate of Origin (COO) などの単一メッセージ実装へも適用できます。

### Requirements

Users of this specification should have an understanding of basic data modelling concepts, basic business information exchange concepts and basic JSON concepts.

この仕様のユーザーは、基本的なデータ モデリングの概念、基本的なビジネス情報交換の概念、および基本的な JSON の概念を理解している必要があります。

### Dependencies

This document depends on

* UN/CEFACT Core Components Technical Specification Version 2.01.
* API TechSpec Open API design rules.

### Caveats and Assumptions

Schemas created as a result of employing this specification should be made publicly available as schema documents in a universally free, accessible, and searchable library. UN/CEFACT will make its contents freely available to any government, individual or organisation who wishes access.

この仕様を採用した結果として作成されたスキーマは、普遍的に無料でアクセス可能で検索可能なライブラリ内のスキーマ ドキュメントとして公開する必要があります。 UN/CEFACT は、アクセスを希望する政府、個人、または組織がその内容を自由に利用できるようにします。

Although this specification defines schema components as expressions of Reference Data Models, non-CCTS developers can also use it for other logical data models and information exchanges.

この仕様では、スキーマ コンポーネントを参照データ モデルの式として定義していますが、CCTS 以外の開発者は、他の論理データ モデルや情報交換にも使用できます。

This specification does not address transformations via scripts or any other means. It does not address any other representation of CCTS artefacts – such as XML, JSON-LD, OWL, and XMI.

この仕様は、スクリプトやその他の手段による変換には対応していません。 XML、JSON-LD、OWL、XMI など、CCTS アーティファクトの他の表現には対応していません。

### Guiding Principles

* JSON Schema Creation

UN/CEFACT JSON Schema design rules will support JSON Schema creation through handcrafting as well as automatic generation.

UN/CEFACT JSON スキーマ設計ルールは、自動生成だけでなく、手作業による JSON スキーマの作成もサポートします。

* Tool Use and Support

The design of UN/CEFACT JSON Schema will not make any assumptions about sophisticated tools for creation, management, storage, or presentation being available.

UN/CEFACT JSON Schema の設計では、作成、管理、保存、またはプレゼンテーションのための特定のツールが利用可能であることを想定していません。

* Technical Specifications

UN/CEFACT JSON Schema Naming and Design Rules will be based on technical specifications holding the equivalent of JSON Schema recommendation status.

UN/CEFACT JSON Schema Naming and Design Rules は、JSON スキーマの推奨ステータスと同等の技術仕様に基づいています。

* JSON Schema Specification

UN/CEFACT JSON Schema Naming and Design Rules will be fully conformant with the JSON Schema recommendation.

UN/CEFACT JSON Schema Naming and Design Rules は、JSON Schema 勧告に完全に準拠します。

* Interoperability

The number of ways to express the same information in a UN/CEFACT JSON Schema and UN/CEFACT JSON instance document is to be kept as close to one as possible.

UN/CEFACT JSON スキーマと UN/CEFACT JSON インスタンス ドキュメントで同じ情報を表現する方法の数は、できるだけ 1 つに近づける必要があります。

* Maintenance

The design of UN/CEFACT JSON Schema must facilitate maintenance.

UN/CEFACT JSON スキーマの設計は、保守を容易にする必要があります。

* Context Sensitivity

The design of UN/CEFACT JSON Schema must ensure that context-sensitive document types are not precluded.

UN/CEFACT JSON スキーマの設計では、特定コンテキストに基づくドキュメント タイプが排除されないようにする必要があります。

* Ease of implementation

UN/CEFACT JSON Schema should be intuitive and reasonably clear in the context for which they are designed. They should allow an intuitive implementation in REST APIs, a.k.a. RESTful API, as well as other interchange appliances.

UN/CEFACT JSON スキーマは、設計されたコンテキストにおいて直感的で合理的に明確でなければなりません。 それらは、REST API、別名 RESTful API、および他の交換アプライアンスでの直感的な実装を可能にする必要があります。

### Conformance

Designers of JSON Schema in governments, private sector, and other standards organisations external to the UN/CEFACT community have found this specification suitable for adoption. To maximize reuse and interoperability across this wide user community, the rules in this specification have been categorized to allow these other organisations to create conformant JSON Schema while allowing for discretion or extensibility in areas that have minimal impact on overall interoperability.

政府、民間部門、および UN/CEFACT コミュニティ以外のその他の標準化団体の JSON スキーマの設計者は、この仕様が採用に適していることを発見しました。 この幅広いユーザー コミュニティ全体で再利用と相互運用性を最大化するために、この仕様のルールは、これらの他の組織が適合する JSON スキーマを作成できるように分類されています。

Accordingly, applications will be considered to be in full conformance with this technical specification if they comply with the content of normative sections, rules and definitions.

|  |
| --- |
| [R 1|1] |
| Conformance SHALL be determined through adherence to the content of the normative sections and rules. Furthermore, each rule is categorized to indicate the intended audience for the rule by the following: |

|  |  |
| --- | --- |
| **Category** | **Description** |
| **1** | Rules, which must not be violated. Else, conformance and interoperability are  lost. |
| **2** | Rules, which may be modified, while still conformant to the NDR structure. |

**Table 3 - Conformance categories**

# JSON Schema Architecture

### Basic architecture

The CCTS defines naming and design rules for a hierarchical data model that supports a document centric modelling approach as well as a resource based modelling approach. In order to support the document centric modelling approach and to be backwards compatible it is designed in a hierarchy. REST APIs on the other hand are resource based only. This means that when moving from CCTS to REST APIs using JSON Schema both options are to be considered. In addition, the JSON syntax has its own naming and design rules that differs from the naming and design rules from the CCTS. This section elaborates on how to move from CCTS to JSON Schema.

CCTS は、ドキュメント中心のモデリング アプローチとリソース ベースのモデリング アプローチをサポートする階層データ モデルの命名規則と設計規則を定義します。 ドキュメント中心のモデリング アプローチをサポートし、下位互換性を維持するために、階層構造で設計されています。 一方、REST API はリソース ベースのみです。 これは、JSON スキーマを使用して CCTS から REST API に移行する場合、両方のオプションを考慮する必要があることを意味します。 さらに、JSON 構文には独自の命名規則と設計規則があり、CCTS の命名規則と設計規則とは異なります。 このセクションでは、CCTS から JSON スキーマに移行する方法について詳しく説明します。

#### JSON serialization in a RESTful context

In order to use the JSON schema artefacts in REST API specifications, the question arises at which level a hierarchical structure is split into a resource-based structure. The UN/CEFACT project API Town Plan has already dealt with this fundamental problem. It formulated that the decision cannot be made centrally in advance. Rather, it depends on the implementation needs in the respective concrete project or the concrete domain.

REST API 仕様で JSON スキーマ アーティファクトを使用するには、どのレベルで階層構造をリソースベースの構造に分割するかという問題が生じます。 UN/CEFACT プロジェクトの API タウン プランは、この根本的な問題にすでに対処しています。 事前に中央で決定を下すことはできないと定式化しました。 むしろ、それぞれの具体的なプロジェクトまたは具体的なドメインでの実装のニーズに依存します。

For this reason, a form of serialization is chosen within the JSON Schema NDR that allows both options for each decision point: The retention of the document-centric hierarchy or the separation according to resources. All ASBIE[2](#_bookmark20) connections are affected by this. The corresponding data type is modelled in the chapter [3.7](#_bookmark52).

このため、JSON スキーマ NDR 内でシリアル化の形式が選択され、各決定点で両方のオプション (ドキュメント中心の階層の保持またはリソースによる分離) が可能になります。 これにより、すべての ASBIE 接続が影響を受けます。 対応するデータ型は、3.7 章でモデル化されています。

#### Overall JSON Schema Structure

|  |
| --- |
| [R 2|1] |
| In the scope of this specification, a JSON schema is a file that complies with a JSON schema definition as defined at https://json-schema.org. It may include subschemas defined in the **$defs** section. A JSON schema fragment means both the overall JSON schema as  well as each of its included subschemas. |

The current version of the JSON Schema draft at the time of publication of this document is 2020-12. It was created in particular due to user requirements in the development of APIs. It corresponds to the OpenAPI 3.1.x version. With this in mind, the latest version of the JSON schema Draft is used in this NDR.

このドキュメントの公開時点での JSON スキーマ ドラフトの現在のバージョンは 2020-12 です。 これは特に、API の開発におけるユーザーの要件のために作成されました。 OpenAPI 3.1.x バージョンに対応しています。 これを念頭に置いて、JSON スキーマ ドラフトの最新バージョンがこの NDR で使用されます。

|  |
| --- |
| [R 3|1]  Each JSON schema SHALL be declared to be a “JSON Draft 2020-12 schema[3](#_bookmark25)” with the appropriate **$schema** string property defined as **https://json-**  **schema.org/draft/2020-12/schema**. |

|  |
| --- |
| [R 4|2] |
| In section [3.8.1](#_bookmark57) a set of rules is defined that allows achieving compatibility with many tools, which do not yet support JSON schema version 2020-12. This set of rules MAY be applied in a publication or the resulting schemas may be published as a second set of JSON schemas marked as "deprecated compatibility set". |

|  |
| --- |
| [R 5|1] |
| Each JSON schema SHALL contain a "**title"** annotation. It SHALL be an overall description title. |

|  |
| --- |
| [R 6|1] |
| Each JSON schema SHALL contain a "**description"** annotation. It contains an overall description for that file as well as copyright information. |

|  |
| --- |
| [R 7|1] |
| Each declared Document and Library ABIE definitions and their BBIE and ASBIE members SHALL contain a "**title"** annotation and a "**description"** annotation. The "**title"** annotation SHALL be the CCTS Dictionary Entry name for the BIE. If there exists a contextualised business name, it SHALL be used instead.  "**description"** annotation shall be the CCTS definition value. Members of enums SHALL NOT contain the "**title"** or the "**description"** annotation. |

|  |
| --- |
| [R 8|1] |
| The **"unevaluatedProperties"** property of each JSON schema fragment SHALL be set to **false**, excluding subschemas for primitive data types, unqualified data types and qualified data types. For subschemas, specifying primitive data types, unqualified data types or qualified data types the **"unevaluatedProperties"** property SHALL be stated as  defined in this document. |

### Versioning and "$id"

Fostering interoperable and highly automated data exchange means enabling machines to process the information in the correct syntactical structure and the correct semantic

3 https://json-schema.org/specification-links.html

4 Basic Business Information Entity

meaning. As requirements change on a regular base, the created standards need to adapt to the new requirements. Therefore, it is necessary to define the given version of the technical artefacts in a machine-readable way.

相互運用可能で高度に自動化されたデータ交換を促進するということは、マシンが正しい構文構造と正しい意味論で情報を処理できるようにすることを意味します。 要件は定期的に変更されるため、作成された標準は新しい要件に適応する必要があります。 したがって、特定のバージョンのテクニカル アーティファクトを機械可読な方法で定義する必要があります。

It is a clear goal to keep the JSON schema artefact structure as compatible as possible with older and future versions.

JSON スキーマ アーティファクト構造を、古いバージョンや将来のバージョンと可能な限り互換性を維持することは、明確な目標です。

|  |
| --- |
| [R 9|1] |
| The JSON schema file names SHALL NOT contain a version information. Differences in versions are only indicated by $id and the folder structure in which the JSON schema artefacts are located. |

|  |
| --- |
| [R 10|1] |
| Each JSON schema being published by user groups or standardisation organisations SHALL contain an identifier for the schema in the appropriate **$id** URI property. JSON schema exports that are only used in a closed environment (e.g. for testing) do NOT NEED to contain the **$id** property.  The URI SHALL follow the following format:  **"$id": "<basepath>/<variant>/<domain>/<version>[/<RDM>]/<BIE>"**  with **<basepath>** identifying the originator. For UNECE artefacts that is  "https://github.com/uncefact/spec-JSONschema"  **<variant>** representing the JSON schema draft version and the export variant. e.g. "JSONschema2020-12/library"  **<domain>** like "BuyShipPay"  **<version>** in the UNECE publication format e.g. "D22A"  **<BIE>** with one   * distinct name for each message assembly ABIE[5](#_bookmark28) (e.g. Cross Industry Invoice) without a file extension * name for all BBIE components: "BasicComponents" * distinct name for every RDM set of library ABIE components:   e.g. "BSP-RDMComponents" or "SC-RDMComponents"  - distinct name for each extension collection  **<RDM>** For the snapshot variant additional structuring is allowed. The JSON schema file name SHALL be built with the following format:  **<originator>-<abbreviation>.json**  with   * **<originator>** identifying the originator. For UNECE artefacts, it SHALL be **UNECE**. * **<abbreviation>** identifying the RDM set of Library ABIE components. If a contextualised business name exists for a message structure, it SHALL be used instead. If a .json-File with this name already exists, the message model name SHALL be added, separated by another hyphen. |

5 Aggregated Business Information Entity

|  |
| --- |
| [Examples]  "$id": "https://github.com/uncefact/spec-JSONschema/JSONschema2020-12/ library/BuyShipPay/D22A/BasicComponents"  UNECE-BasicComponents.json  "$id": "https://github.com/uncefact/spec-JSONschema/JSONschema2020-12/ library/BuyShipPay/D22A/CrossIndustryInvoice"  UNECE-CrossIndustryInvoice.json  "$id": "https://github.com/uncefact/spec-JSONschema/JSONschema2020-12/ library/BuyShipPay/D22A/CrossIndustryInvoice-Variant"[6](#_bookmark30)  UNECE-CrossIndustryInvoice-Variant.json |

|  |
| --- |
| [R 11|1] |
| The BasicComponents JSON schema file SHALL contain all subschemas for primitive data types, unqualified data types as well as qualified data types. |

### General naming rules moving from CCTS to JSON

The dictionary entry names follow specific naming rules defined in the CCTS containing

" **"**

**.**

special characters like full stops naming entities.

and white spaces

that are not allowed in JSON for

The basic rules listed below apply when transferring CCTS names in JSON schema.

JSON スキーマで CCTS 名を転送する場合は、以下に示す基本的な規則が適用され

ます。

|  |
| --- |
| [R 12|1] |
| A property is a name/value pair inside a JSON object. The property name is the key or name part of the property. The property value is the value part of the property. |
| [Example]  {  "propertyName": "propertyValue"  } |

|  |
| --- |
| [R 13|1] |
| JSON property names SHALL be derived from Dictionary Entry Names (DEN).  In e.g. in a BBIE or ASBIE the DEN contains the DEN of the surrounding ABIE, it SHALL be removed. In case a BBIE or ASBIE contains consecutive identical words, the duplication SHALL be removed. If by applying the NDR rules words in the DEN are duplicated, the duplication SHALL be removed. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [R 14|1] | | | | |
| Any special characters such full stops | | | **.** | , non-breaking spaces (ASCII code 160) and |
| underscores **\_** SHALL be removed from the underlying Dictionary Entry Name. If a digit (0-9) was before and another digit after the white space, the white space SHALL be | | | | |
| replaced by a hyphen | - | . | | |
| [Example]  "This. is\_ a. class. name" is represented as "thisIsAClassName"  "ISO 4217 3 A" is represented as "ISO4217-3A" | | | | |

|  |
| --- |
| [R 15|1] |
| JSON property names SHALL be lower camel-cased ASCII strings and JSON schema compliant: The character after a white space shall be a capital letter. Capital letters in the DEN SHALL NOT be kept. |
| [Example]  "Specified. IBAN. Identifier" is represented as "specifiedIbanId"  "AAA Archive\_ Document. Specified. AAA Archive\_ Archive Parameter" is represented as "specifiedAaaArchiveParameter" |

|  |
| --- |
| [R 16|1] |
| The abbreviations and acronyms SHALL be used as defined in [Table 4](#_bookmark32).  [[R 15|1]](#_bookmark31) SHALL be taken into account. |

|  |  |
| --- | --- |
| CCTS Appearance | JSON Representation |
| "Uniform Resource. Identifier"  or  "URI\_  Identification. Identifier" | "Uri"  with  "type": "string"  "format": "uri"  The rule for abbreviating "Identifier" is not applied in this case. It SHALL NOT be abbreviated as "UriId". |
| "Identification Scheme" | "Scheme" |
| "Details" | "Type" |
| "Identifier" | "Id" |
| "Indicator" | SHALL be omitted. "isOrHas" is added as a prefix. |
| "Identification. Identifier" | "Id" |
| "Text" | SHALL be omitted |
| "Specified\_" | SHALL be omitted |
| "AAA " at the beginning  "TT\_"  "Transport\_" "Supply Chain\_"  "CI\_" | SHALL be omitted, if the resulting name of the ABIE is unique, else it SHALL be kept |
| "Formatted\_" | SHALL be omitted |

|  |  |
| --- | --- |
| CCTS Appearance | JSON Representation |
| "Trade\_ Party" at the end | SHALL be omitted |

**Table 4 – JSON Representation for abbreviations and acronyms**

|  |
| --- |
| [R 17|1] |
| The Object Class Term **"Identification Scheme"** SHALL be represented as  **"Scheme"**. [[R 15|1]](#_bookmark31) SHALL be taken into account. |

### JSON schema landscape

BasicComponents

* Unqualified Data Types
* Qualified Data Types
* OpenAPI Data Types

Code Lists and Identification Lists

Agri-RDMComponents

MMT-RDMComponents

BSP-RDMComponents

* ABIEs
* BBIEs

SC-RDMComponents

* ABIEs
* BBIEs

CrossIndustryInvoice eCMR

…

* ABIEs

ExtensionComponents

…

* ABIEs
* QDTs

Figure 1 – JSON schema landscape

### Data types

The CCTS defines a hierarchical relationship of basic data types. From primitive data types (PDT), Approved Core Component Types (CCT) and finally unqualified data types (UDT) are formed.[7](#_bookmark36)

CCTS は、基本データ型の階層関係を定義します。 プリミティブ データ型 (PDT) から、承認済みコア コンポーネント型 (CCT)、そして最後に非修飾データ型 (UDT) が形成されます。

#### Primitive Data Types

The decimal data type, which is used in particular to represent amounts (in a specific currency), as well as measured values, requires special treatment. JSON does not support such a decimal data type. It only supports the data type "number", which is technically implemented as a float or double precision data type.

特に (特定の通貨での) 金額や測定値を表すために使用される 10 進データ型には、特別な処理が必要です。 JSON は、このような 10 進データ型をサポートしていません。 これは、浮動小数点または倍精度データ型として技術的に実装されているデータ型「数値」のみをサポートします。

There are many discussions[8](#_bookmark37), but also practical experiences (e.g. based on the application of validation rules from the implementation of EN16931), which show the difficulties of using float data types instead of a decimal data type.

多くの議論がありますが、10 進データ型の代わりに float データ型を使用することの難しさを示す実際の経験 (例えば、EN16931 の実装からの検証規則の適用に基づく) もあります。

In summary, it can be stated that the use of a float data type inevitably leads to rounding differences and imprecise representations of the transmitted values. Since the implementation of the UNECE reference data models involves the exchange of business data, precise transmission of values is the top priority. With this in mind, the decimal data type is represented as a string representation in JSON schema. This can be implemented cleanly and without loss in the various implementation languages, even if direct arithmetic use is not possible at JSON level.

要約すると、float データ型を使用すると、必然的に丸め誤差が発生し、送信された値の表現が不正確になると言えます。 UNECE 参照データ モデルの実装にはビジネス データの交換が伴うため、正確な値の伝達が最優先事項です。 これを念頭に置いて、10 進数データ型は JSON スキーマの文字列表現として表されます。 これは、JSON レベルで直接演算を使用できない場合でも、さまざまな実装言語で無駄なくきれいに実装できます。

Examples for the implementation of the decimal type are:

|  |  |
| --- | --- |
| **Language** | **Implementation** |
| C# | decimal |
| Go | decimal |
| Java | java.math.BigDecimal |
| JavaScript | decimal.js |
| Python | decimal.Decimal |

**Table 5 – Implementation of the decimal type in different languages**

Primitive data types (PDT) SHALL be represented in JSON schema, as stated in [Table 6](#_bookmark38). They SHALL be placed under **$defs/pdt/**.

[R 18|1]

7 See CCTS Section 8.1

8 See e.g. https://github.com/zalando/jackson-datatype-money/blob/main/MONEY.md

|  |  |
| --- | --- |
| **CCTS**  **Primitive data type** | **JSON Representation** |
| Binary | Representation when used with OpenAPI 3.0.x:  "binaryType":  {  "title": "Binary",  "description": "",  "type": "string",  "format": "byte"  }  Representation when used with OpenAPI 3.1.x (full JSON Schema support, default publication variant):  "binaryType":  {  "title": "Binary",  "description": "",  "type": "string", "contentEncoding": "base64"  } |
| Boolean | "type": "boolean" |
| Decimal | "decimalType":  {  "title": "Decimal",  "description": "",  "type": "string",  "pattern": "^([+-]?(0?|[1-9][0-9]\*)(\\.?\\d+))$"  } |
| Integer | "type": "integer" |
| String | "type": "string" |

**Table 6 – JSON representation of CCTS Primitive data types**

#### Approved Core Component Types

The Approved Core Component Types have no direct representation in JSON schema. Instead, UDTs are mapped directly into JSON schema.

承認済みコア コンポーネント タイプは、JSON スキーマで直接表現されていません。 代わりに、UDT は JSON スキーマに直接マップされます。

#### Unqualified Data Types

UDTs form the basis for all further data structures of the CCTS. They consist of the actual value (**content**), as well as usually optional supplementary components[9](#_bookmark41). During contextualisation, some of these supplementary components are often omitted.

UDT は、CCTS のその他すべてのデータ構造の基礎を形成します。 これらは、実際の値 (コンテンツ) と、通常はオプションの補足コンポーネントで構成されます。 コンテキスト化の間、これらの補足コンポーネントのいくつかはしばしば省略されます。

This in fact multiplies the number of UDTs in the actual implementation and complicates it technically. For this reason, contextualisation of UDTs is not mapped into JSON schema. Instead, the complete UDTs in the higher data types are always used.

これにより、実際の実装では UDT の数が倍増し、技術的に複雑になります。 このため、UDT のコンテキスト化は JSON スキーマにマップされません。 代わりに、より高いデータ型の完全な UDT が常に使用されます。

|  |
| --- |
| [R 19|1] |
| Unqualified data types SHALL be represented in subschemas. **"Type"** as part of the Dictionary Entry Name SHALL be retained. |

|  |
| --- |
| [R 20|1] |
| The CCTS content property SHALL be represented in a subschema with the name  **"content"**. Its data type SHALL use the underlying PDT. The content-property SHALL  be required. |

|  |
| --- |
| [R 21|1] |
| Property names of supplementary components SHALL NOT repeat the JSON subschemas property name. |

|  |
| --- |
| [R 22|1] |
| Supplementary components may reference to code lists and/or identification schemes. In this case, the JSON property SHALL reference the appropriate code list or identification scheme as defined in section [3.5.5](#_bookmark45) [Other Qualified Data Types.](#_bookmark45) |

|  |
| --- |
| [R 23|1] |
| Unqualified data types SHALL be represented in subschemas as shown in [Table 7.](#_bookmark42) The **title** and **description** properties are not shown in the following table. Instead, they are indicated with the placeholder **<title and description>** as those can change over time. They SHALL be published in alignment with rules [[R 5|1],](#_bookmark21) [[R 6|1]](#_bookmark22), and [[R 7|1].](#_bookmark23)  They SHALL be placed under **$defs/udt**. |

|  |  |
| --- | --- |
| **CCTS**  **Unqualified data type** | **JSON Representation** |
| * Amount. Type * Amount. Content * Amount Currency. Identifier * Amount Currency. Code List Version. Identifier | **"amountType": {**  **<<title and description>> "type": "object", "properties": {**  **"content": {**  **<<title and description>>**  **"$ref": "#/$defs/pdt/decimalType"**  **},**  **"currencyId": {**  **<<title and description>> "$ref": "ISO\_4217-**  **3A.json#/$defs/codeList/iso4217-3AType"**  **},**  **"currencyCodeListVersionId": {**  **<<title and description>> "type": "string"**  **}**  **},**  **"required": [ "content" ], "unevaluatedProperties": false**  **}** |

|  |  |
| --- | --- |
|  |  |
| * Binary Object. Type | "binaryObjectType": {  <<title and description>> "type": "object", "properties": {  "content": {  <<title and description>> "$ref": "#/$defs/pdt/binaryType"  },  "format": {  <<title and description>> "type": "string"  },  "mimeCode": {  <<title and description>> "type": "string"  },  "encodingCode": {  <<title and description>> "$ref":  "UNECE\_CharacterSetEncoding.json#/$defs/ codeList/characterSetEncodingType"  },  "characterSetCode": {  <<title and description>>  "$ref": "UNECE\_CharacterSets.json#/$defs/ codeList/characterSetsType"  },  "uri": {  <<title and description>> "type": "string",  "format": "uri"  },  "filename": {  <<title and description>> "type": "string"  }  },  "required": [ "content" ] , "unevaluatedProperties": false  } |
| * Binary Object. Content |
| * Binary Object. Format. |
| Text |
| * Binary Object. Mime. |
| Code |
| * Binary Object. |
| Encoding. Code |
| * Binary Object. |
| Character Set. Code |
| * Binary Object. Uniform |
| Resource. Identifier |
| * Binary Object. |
| Filename. Text |
|  |  |
| * Code. Type | "codeType": {  <<title and description>> "type": "object", "properties": {  "content": {  <<title and description>> "type": "string"  },  "listId": {  <<title and description>> "type": "string"  },  "listAgencyId": {  <<title and description>> "$ref": "UNECE\_UNTDID-  3055.json#/$defs/codeList/untdid3055Type"  }, |
| * Code. Content |
| * Code List. Identifier |
| * Code List. Agency. |
| Identifier |
| * Code List. Agency |
| Name. Text |
| * Code List. Name. Text |
| * Code List. Version. |
| Identifier |
| * Code. Name. Text |
| * Language. Identifier |
| * Code List. Uniform |
| Resource. Identifier |

|  |  |
| --- | --- |
| Code List Scheme. Uniform Resource. Identifier | "listAgencyName": {  <<title and description>> "type": "string"  },  "listName": {  <<title and description>> "type": "string"  },  "listVersionId": {  <<title and description>> "type": "string"  },  "name": {  <<title and description>> "type": "string"  },  "languageId": {  <<title and description>> "$ref": "UNECE\_UNTDID-  3453.json#/$defs/codeList/untdid3453Type"  },  "listUri": {  <<title and description>> "type": "string",  "format": "uri"  },  "listSchemaUri": {  <<title and description>> "type": "string",  "format": "uri"  }  },  "required": [ "content" ] , "unevaluatedProperties": false  } |
|  |  |
| * Date Time. Type | "dateTimeType": {  <<title and description>> "type": "string", "format": "date-time"  } |
|  |  |
| * Date. Type | "graphicType": {  <<title and description>>  "$ref": "#/$defs/udt/binaryObjectType"  } |
|  |  |
| * Graphic. Type | "graphicType": {  <<title and description>>  "$ref": "#/$defs/udt/binaryObjectType"  } |
|  |  |
| * Identifier. Type * Identifier. Content * Identification Scheme. Identifier * Identification Scheme. Name. Text | "identifierType": {  <<title and description>> "type": "object", "properties": {  "content": {  <<title and description>> "type": "string"  }, |

|  |  |
| --- | --- |
| * Identification Scheme Agency. Identifier * Identification Scheme. Agency Name. Text * Identification Scheme. Version. Identifier * Identification Scheme Data. Uniform Resource. Identifier * Identification Scheme. Uniform Resource. Identifier | "schemeId": {  <<title and description>> "type": "string"  },  "schemeName": {  <<title and description>> "type": "string"  },  "schemeAgencyId": {  <<title and description>> "$ref": "UNECE\_UNTDID-  3055.json#/$defs/codeList/untdid3055Type"  },  "schemeAgencyName": {  <<title and description>> "type": "string"  },  "schemeVersionId": {  <<title and description>> "type": "string"  },  "schemeDataUri": {  <<title and description>> "type": "string",  "format": "uri"  },  "schemeUri": {  <<title and description>> "type": "string",  "format": "uri"  }  },  "required": [ "content" ], "unevaluatedProperties": false  } |
|  |  |
| * Indicator. Type | "indicatorType": {  <<title and description>> "type": "boolean"  } |
|  |  |
| * Measure. Type * Measure. Content * Measure Unit. Code * Measure Unit. Code List Version. Identifier | "measureType": {  <<title and description>> "type": "object", "properties": {  "content": {  <<title and description>> "$ref": "#/$defs/pdt/decimalType"  },  "unitCode": {  <<title and description>> "$ref": "UNECE\_UNTDID-  6411.json#/$defs/codeList/untdid6411Type"  },  "unitCodeListVersionId": {  <<title and description>> "type": "string"  }  },  "required": ["content" ], |

|  |  |
| --- | --- |
|  | "unevaluatedProperties": false  } |
|  |  |
| * Name. Type * Text. Content * Language. Identifier * Language. Locale. Identifier | "nameType": {  <<title and description>> "$ref": "#/$defs/udt/textType"  } |
|  |  |
| * Numeric. Type * Numeric. Content * Numeric. Format. Text | "numericType": {  <<title and description>> "type": "object", "properties": {  "content": {  <<title and description>> "$ref": "#/$defs/pdt/decimalType"  },  "format": {  <<title and description>> "type": "string"  }  },  "required": [ "content" ] , "unevaluatedProperties": false  } |
|  |  |
| * Percent. Type | "percentType": {  <<title and description>> "$ref": "#/$defs/udt/numericType"  } |
|  |  |
| * Picture. Type | "pictureType": {  <<title and description>>  "$ref": "#/$defs/udt/binaryObjectType"  } |
|  |  |
| * Quantity. Type * Quantity. Content * Quantity Unit. Code * Quantity Unit. Code List. Identifier * Quantity Unit. Code List Agency. Identifier * Quantity Unit. Code List Agency Name. Text | "quantityType": {  <<title and description>> "type": "object", "properties": {  "content": {  <<title and description>> "$ref": "#/$defs/pdt/decimalType"  },  "unitCode": {  <<title and description>> "$ref": "UNECE\_REC-  20+21.json#/$defs/codeList/rec20+21Type"  },  "unitCodeListId": {  <<title and description>> "type": "string"  },  "unitCodeListAgencyId": {  <<title and description>> "$ref": "UNECE\_UNTDID-  3055.json#/$defs/codeList/untdid3055Type"  }, |

|  |  |
| --- | --- |
|  | "unitCodeListAgencyName": {  <<title and description>> "type": "string"  }  },  "required": [ "content" ], "unevaluatedProperties": false  } |
| [Note]  Rec 20 supports a combination with Rec 21 by adding a prefix to the  Rec 21 code values. In the usage of this JSON subschema, the combined code list can be restricted as needed. |
|  |  |
| * Rate. Type | "rateType": {  <<title and description>> "$ref": "#/$defs/udt/numericType"  } |
|  |  |
| * Sound. Type | "soundType": {  <<title and description>>  "$ref": "#/$defs/udt/binaryObjectType"  } |
|  |  |
| * Text. Type * Text. Content * Language. Identifier * Language. Locale. Identifier | "textType": {  <<title and description>> "type": "object", "properties": {  "content": {  <<title and description>> "type": "string"  },  "languageId": {  <<title and description>> "$ref": "ISO\_6391-1-  2A.json#/$defs/codeList/iso6391-1-2AType"  },  "languageLocaleId": {  <<title and description>> "type": "string"  }  },  "required": [ "content" ], "unevaluatedProperties": false  } |
|  |  |
| * Time. Type | "timeType": {  <<title and description>> "type": "string",  "format": "time"  } |
|  |  |
| * Value. Type | "valueType": {  <<title and description>> "$ref": "#/$defs/udt/numericType"  } |
|  |  |

|  |  |
| --- | --- |
| * Video. Type | "videoType": {  <<title and description>>  "$ref": "#/$defs/udt/binaryObjectType"  } |

**Table 7 – JSON representation of Unqualified data types**

#### Qualified Data Types for Date and Time

The CCTS supports the wide subset of the different date and time formats of ISO 8601. However, this flexibility is only needed and used to a limited extent in practical applications. Often, date, time and combined information can be reduced to their simple representation form, which is directly supported by JSON schema.

CCTS は、ISO 8601 のさまざまな日付と時刻の形式の幅広いサブセットをサポートしています。ただし、この柔軟性は、実際のアプリケーションでは限られた範囲でのみ必要とされ、使用されます。 多くの場合、日付、時刻、および結合された情報は、JSON スキーマで直接サポートされている単純な表現形式に縮小できます。

There exist a few exceptions, so that in the CCTS some specialised QDTs have been defined. The modelling of these QDTs goes back to the early EDIFACT definitions and no longer seems up-to-date for application in OpenAPI using JSON schema. Nevertheless, this notation is still used in a wide community. Against this background, the following simplification of these QDTs is used:

いくつかの例外が存在するため、CCTS ではいくつかの特殊な QDT が定義されています。 これらの QDT のモデル化は初期の EDIFACT 定義にまでさかのぼり、JSON スキーマを使用した OpenAPI でのアプリケーションにはもはや最新のものではないようです。 それにもかかわらず、この表記法は依然として広いコミュニティで使用されています。 この背景に対して、これらの QDT の次の単純化が使用されます。

|  |
| --- |
| [R 24|1] |
| The **"Date Mandatory\_ Date Time. Type"** SHALL be replaced by the  **formattedDateTimeType**. |

|  |
| --- |
| [R 25|1] |
| The **"Time Only\_ Formatted\_ Date Time. Type"** SHALL be replaced by the  **formattedDateTimeType**. |

The implementation of the Formatted Date Time Type shall take into account the direct mappability of certain date and time information directly into JSON schema. To allow an intuitive implementation, the code list UNTDID 2379 is replaced by a JSON specific variant for this purpose.

Formatted Date Time Type の実装では、特定の日付と時刻の情報を JSON スキーマに直接マッピングできることを考慮に入れる必要があります。 直感的な実装を可能にするために、コード リスト UNTDID 2379 は、この目的のために JSON 固有のバリアントに置き換えられます。

|  |
| --- |
| [R 26|1] |
| The **"Formatted\_ Date Time. Type"** SHALL be represented as follows.  **"formattedDateTimeType": {**  **<<title and description>> "oneOf": [**  **{ "type": "string", "format": "date-time" },**  **{ "type": "string", "format": "time" },**  **{ "type": "string", "format": "date" },**  **{ "type": "string", "format": "duration" },**  **{ "type": "object", "properties": {**  **"content": { "type": "string" }, "format": { "$ref": "UNECE\_UNTDID2379-**  **JSON.json#/$defs/codeList/untdid2379JsonType" }**  **},**  **"required": ["content", "format"]**  **}**  **]**  **}** |

|  |
| --- |
| **[Example]**  JSON schema definition:  **{ "properties": {**  **"myDateTime": { "$ref": "#/$defs/formattedDateTimeType"}**  **}**  **}**  JSON instance:  Hint: The presence of "content" indicates that it is a UNECE specific format not directly supported by JSON schema.  **{**  **"myDateTime": {"content": "2022-W02", "format": "CCYY-Www"}, "myDateTime": {"content": "1T10:00/1T12:00", "format":**  **"NThh:mm/NThh:mm"},**  **"myDateTime": "2022-02-11", "myDateTime": "2022-02-11T12:23:58Z",**  **"myDateTime": "12:23:58Z", "myDateTime": "P10W"**  **}** |

|  |
| --- |
| [R 27|1] |
| Based on the code list **"UNTDID 2379"** an additional code list **"UNTDID 2379 json"** SHALL be specified. All format definitions that are already represented in their meaning by existing JSON date and time formats SHALL be omitted. This code list SHALL be  maintained in accordance with UNTDID 2379. All other formats SHALL be represented as follows. |
| **"untdid2379JsonType": {**  **"title": "Date and Time format codes for JSON representation.", "description": "This code list is based on UNTDID 2379. It is adjusted**  **to take JSON date and time representation into account.\n # The following abbreviations are used\n**   * **'C' – Century\n** * **'Y' – Year\n** * **'M' – Month\n** * **'D' – Day\n** * **'h' – Hour\n** * **'m' – Minute\n** * **'s' – Second\n** * **'w' – Week\n** * **'T' – Time zone offset separator (+/-/Z) \n**   **\n**   * **'A' – 10 day period within a month of a year\n** * **'B' – 1: First half month; 2: second half month\n** * **'E' – Week of a month\n** * **'G' – Working days\n** * **'H' – Half month\n** * **'I' – 1-9: Shift in a day\n** * **'K' – 1-5: First to fifth week in a month\n** * **'M' – Trimester: A period of three months\n** * **'N' – 1-7: Numeric representation of the day (Monday = 1, Sunday = 7)\n** * **'P' – A period of 4 months\n** * **'Q' – Quarter\n** * **'RR' – 00-99: Time period\n** * **'S' – Semester\n**   **\*\n** |

|  |
| --- |
| * **Hyphens and additional character in a format string are kept. According to ISO 8601 a slash is used to separate time spans.\n**   **# Codes from UNTDID 2379 and their representation in JSON\n**   * **'2' – is represented as 'date' format\n** * **'3' – is represented as 'date' format\n** * **'4' – is represented as 'date' format\n** * **'5' – is represented as 'date-time' format\n** * **'6' – is represented as 'CCYY-MM-B'\n** * **'7' – is represented as 'CCYY-MM-K'\n** * **'8' – is represented as 'CCYY-MM-DD-I'\n** * **'9' – is represented as 'CCYY-MM-DD-RR'\n** * **'10' – is represented as 'date-time' format\n** * **'101' – is represented as 'date' format\n** * **'102' – is represented as 'date' format\n** * **'103' – is represented as 'YY-Www-N'; 01 is first week of January; 1 is always Monday\n** * **'104' – is represented as 'MM-WEE/MM-WEE'\n** * **'105' – is represented as 'YY-DDD'; January the first = Day 001; Always start numbering the days of the year from January 1st through December 31st \n** * **'106' – is represented as '-MM-DD'\n** * **'107' – is represented as 'DDD'\n** * **'108' – is represented as 'WW'\n** * **'109' – is represented as '-MM-'\n** * **'110' – is represented as '--DD'\n** * **'201' – is represented as 'date-time' format\n** * **'202' – is represented as 'date-time' format\n** * **'203' – is represented as 'date-time' format\n** * **'204' – is represented as 'date-time' format\n** * **'205' – is represented as 'date-time' format\n** * **'206' – is represented as 'date-time' format\n** * **'207' – is represented as 'date-time' format\n** * **'208' – is represented as 'date-time' format\n** * **'209' – is represented as 'date-time' format\n** * **'210' – is represented as 'hh:mm:ssZhh:mm/hh:mm:ssZhh:mm'\n** * **'301' – is represented as 'date-time' format\n** * **'302' – is represented as 'date-time' format\n** * **'303' – is represented as 'date-time' format\n** * **'304' – is represented as 'date-time' format\n** * **'305' – is represented as '-MM-DDThh:mm' format\n** * **'306' – is represented as '--DDThh:mm' format\n** * **'307' – is represented as 'date-time' format\n** * **'308' – is represented as 'CCYY-MM-DDThh:mmZhh:mm/CCYY-MM- DDThh:mmZhh:mm' \n** * **'401' – is represented as 'time' format\n** * **'402' – is represented as 'time' format\n** * **'404' – is represented as 'time' format\n** * **'405' – is represented as 'duration' format\n** * **'406' – is represented as 'Zhh:mm'\n** * **'501' – is represented as 'hh:mm/hh:mm' \n** * **'502' – is represented as 'hh:mm:ss/hh:mm:ss' \n** * **'503' – is represented as 'hh:mm:ssZhh:mm/hh:mm:ssZhh:mm' \n** * **'600' – is represented as 'CC'\n** * **'601' – is represented as 'YY' \n** * **'602' – is represented as 'CCYY' \n** * **'603' – is represented as 'YY-S' \n** * **'604' – is represented as 'CCYY-S' \n** * **'608' – is represented as 'CCYY-Q' \n** * **'609' – is represented as 'YY-MM' \n** * **'610' – is represented as 'CCYY-MM' \n** * **'613' – is represented as 'YY-MM-A' \n** |

|  |
| --- |
| * **'614' – is represented as 'YY-MM-A' \n** * **'615' – is represented as 'YY-Www \n** * **'616' – is represented as 'CCYY-Www' \n** * **'701' – is represented as 'YY/YY' \n** * **'702' – is represented as 'CCYY/CCYY' \n** * **'703' – is represented as 'YY-S/YY-S' \n** * **'704' – is represented as 'CCYY-S/CCYY-S' \n** * **'705' – is represented as 'YY-P/YY-P' \n** * **'706' – is represented as 'CCYY-P/CCYY-P' \n** * **'707' – is represented as 'YY-Q/YY-Q' \n** * **'708' – is represented as 'CCYY-Q/CCYY-Q' \n** * **'709' – is represented as 'YY-MM/YY-MM' \n** * **'710' – is represented as 'CCYY-MM/CCYY-MM' \n** * **'713' – is represented as 'YY-MM-DDThh:mm/YY-MM-DDThh:mm' \n** * **'715' – is represented as 'YY-Www/YY-Www' \n** * **'716' – is represented as 'CCYY-Www/CCYY-Www' \n** * **'717' – is represented as 'YY-MM-DD/YY-MM-DD' \n** * **'718' – is represented as 'CCYY-MM-DD/CCYY-MM-DD' \n** * **'719' – is represented as 'CCYY-MM-DDThh:mm/CCYY-MM-DDThh:mm' \n** * **'720' – is represented as 'NThh:mm/NThh:mm' \n** * **'801' – is represented as 'duration' format \n** * **'802' – is represented as 'duration' format \n** * **'803' – is represented as 'duration' format \n** * **'804' – is represented as 'duration' format \n** * **'805' – is represented as 'duration' format \n** * **'806' – is represented as 'duration' format \n** * **'807' – is represented as 'duration' format \n** * **'808' – is represented as 'S' \n** * **'809' – is represented as 'P' \n** * **'810' – is represented as 'M' \n** * **'811' – is represented as 'H' \n** * **'812' – is represented as 'A' \n** * **'813' – is represented as 'N' \n** * **'814' – is represented as 'G' \n ",**   **"oneOf": [**  **{ "const": "CCYY-MM-B" },**  **{ "const": "CCYY-MM-K" },**  **{ "const": "CCYY-MM-DD-I" },**  **{ "const": "CCYY-MM-DD-RR" },**  **{ "const": "YY-Www-N" },**  **{ "const": "MMWEE/MMWEE" },**  **{ "const": "YY-DDD" },**  **{ "const": "-MM-DD" },**  **{ "const": "DDD" },**  **{ "const": "-WW" },**  **{ "const": "-MM-" },**  **{ "const": "--DD" },**  **{ "const": "hh:mm:ssZhh:mm/hh:mm:ssZhh:mm" },**  **{ "const": "-MM-DDThh:mm" },**  **{ "const": "--DDThh:mm" },**  **{ "const": "CCYY-MM-DDThh:mmZhh:mm/CCYY-MM-DDThh:mmZhh:mm" },**  **{ "const": "Zhh:mm" },**  **{ "const": "hh:mm/hhmm" },**  **{ "const": "hh:mm:ss/hh:mm:ss" },**  **{ "const": "hh:mm:ssZhh:mm/hh:mm:ssZhh:mm" },**  **{ "const": "CC" },**  **{ "const": "YY" },**  **{ "const": "CCYY" },**  **{ "const": "CCYY-S" },**  **{ "const": "CCYY-Q" },** |

|  |
| --- |
| **{ "const": "YY-MM" },**  **{ "const": "CCYY-MM" },**  **{ "const": "YY-MM-A" },**  **{ "const": "CCYY-MM-A" },**  **{ "const": "YY-Www" },**  **{ "const": "CCYY-Www" },**  **{ "const": "YY/YY" },**  **{ "const": "CCYY/CCYY" },**  **{ "const": "YY-S/YY-S" },**  **{ "const": "CCYY-S/CCYY-S" },**  **{ "const": "YY-P/YY-P" },**  **{ "const": "CCYY-P/CCYY-P" },**  **{ "const": "YY-Q/YY-Q" },**  **{ "const": "CCYY-Q/CCYY-Q" },**  **{ "const": "YY-MM/YY-MM" },**  **{ "const": "CCYY-MM/CCYY-MM" },**  **{ "const": "YY-MM-DDThh:mm/YY-MM-DDThh:mm" },**  **{ "const": "YYWww/YYWww" },**  **{ "const": "CCYYWww/CCYYWww" },**  **{ "const": "YY-MM-DD/YY-MM-DD" },**  **{ "const": "CCYY-MM-DD/CCYY-MM-DD" },**  **{ "const": "CCYY-MM-DDThh:mm/CCYY-MM-DDThh:mm" },**  **{ "const": "NThh:mm/NThh:mm" },**  **{ "const": "S" },**  **{ "const": "P" },**  **{ "const": "M" },**  **{ "const": "H" },**  **{ "const": "A" },**  **{ "const": "N" },**  **{ "const": "G" }**  **]**  **}** |

#### Other Qualified Data Types

In the CCTS code and identifier lists are specified as qualified data types (QDT). They base on the UDT **codeType or idType** The UDT codeType and as before described idType offers the ability to state code list or identification scheme specific properties like the publishing agency or the used code list version or schema version.

CCTS コードと識別子リストでは、修飾データ型 (QDT) として指定されています。 それらは UDT codeType または idType に基づいています。UDT codeType および前述の idType は、コード リストまたは発行機関や使用されているコード リスト バージョンまたはスキーマ バージョンなどの識別スキーム固有のプロパティを示す機能を提供します。

Not in every code list and identification scheme or qualified data type, all of these properties are applicable, which is taken into account.

すべてのコード リストと識別スキームまたは修飾されたデータ型ではなく、これらのすべてのプロパティが適用可能であり、考慮されます。

|  |
| --- |
| [R 28|1] |
| Each QDT that does not fall under section [3.5.4](#_bookmark43) SHALL be restricted according to its definition applying the method described in section [3.6.1.](#_bookmark49) |
| [Example]  "unitMeasureType": {  "title": "Unit\_ Measure. Type",  "description": "A numeric value determined by measuring an object along with the specified unit of measure.",  "$ref" : "#/$defs/udt/measureType", "required": ["unitCode"], "properties": {  "unitCodeListVersionId": false  }  } |

|  |
| --- |
| [R 29|1] |
| Each QDT SHALL be represented in a subschema. If code or id values are specified locally, they SHALL be as a **oneOf** combination of **const** definitions. They SHALL NOT be specified as **enum** arrays. Each code value SHALL be represented as a **string** type. If the values of codes and ids are organised in code and identification schemes the corresponding  JSON schema SHALL refer to the appropriate code list or identification scheme. |

|  |
| --- |
| [R 30|1] |
| Each code list and identification scheme SHALL be specified in a separate JSON schema file.  A JSON schema file SHALL be created for each code list and identification scheme being used. Its name SHALL represent the name of the code list or identification scheme and SHALL be unique with the following form:  **<Code List Agency Name>\_<Code List Name or Identifier>.json**  **<Identification Scheme Agency Name>\_<Identification Scheme Name or Identifier>.json**  Where:   * All special characters SHALL be removed from the name. A period . in the version number is replaced by the letter **p**. * <Code List Agency Name> – Agency that maintains the code list. * <Identification Scheme Agency Name> – Agency that maintains the identification scheme. * <Code List Name or Identifier> – If a code list identifier exists in the UNTDID, the identifier is given in the format **UNTDID<identifier>**. Else, the code list name is stated as assigned by the publishing agency. * <Identification Scheme Name or Identifier> – If an identification scheme identifier exists in the UNTDID, the identifier is given in the format **UNTDID<identifier>**. Else, the identification scheme name is stated as assigned by the publishing agency.   The file SHALL be placed in a subfolder **codelists** of the export path. The **$id**  property SHALL reflect this subfolder structure. |
| [Example] UNECE\_UNTDID-1001.json  OpenPEPPOL\_DocumentTypes.json |

|  |
| --- |
| [R 31|2] |
| It is a clear goal to keep the JSON schema artefacts as compatible with code lists and identification schemes as possible. For this reason the code list version and identification scheme version is neither part of the .json filename nor part of the type name. Nevertheless, it is part of the $id, so that JSON schema files can be used for differentiating versions if needed. If for some reason more than one version of a code list or identification scheme needs to be used in a specific scenario, the **<Code List Version>** or  **<Identification Scheme Version>** SHOULD be added to the file name in the following format: |

|  |
| --- |
| **<Code List Agency Name>\_<Code List Name or Identifier>\_<Code List Version>.json**  **<Identification Scheme Agency Name>\_<Identification Scheme Name or Identifier>\_<Identification Scheme Version>.json** |

Since the invention of JSON, there has been repeated discussion about whether JSON should support comments in schema files. In terms of its basic concept, JSON is data-only and it was deliberately decided not to support comments. Nevertheless, as versioning progressed, annotations such as description and $comment were introduced. The latter is supposed to be ignored by parsers and should not be used to present information to schema users. Instead $comment is only intended to contain information for future schema developers e.g. to highlight schema maintenance information[10](#_bookmark47). A much-discussed topic for years is the commenting of enums.

JSON が発明されて以来、JSON がスキーマ ファイルでコメントをサポートすべきかどうかについて、繰り返し議論されてきました。 JSON は基本的な概念としてはデータのみであり、意図的にコメントをサポートしないように決定されました。 それにもかかわらず、バージョン管理が進むにつれて、説明や $comment などの注釈が導入されました。 後者はパーサーによって無視されることが想定されており、スキーマ ユーザーに情報を提示するために使用するべきではありません。 代わりに、$comment は、将来のスキーマ開発者向けの情報を含めることのみを目的としています。 スキーマのメンテナンス情報を強調する10。 何年もの間よく議論されてきたトピックは、列挙型のコメントです。

JSON Schema does not support comments in the .JSON file analogous to the double slash in languages like C or the hashtag as in PHP. Some JSON editors support such comments proprietarily. However, usually only one of the two variants, which often correspond to the conventions of one's own programming language. Since there is consequently no universal convention, the UNECE JSON Schema code and identifier lists dispense with such proprietary comments.

JSON スキーマは、C などの言語の二重スラッシュや PHP のハッシュタグに相当する .JSON ファイル内のコメントをサポートしていません。 一部の JSON エディターは、このようなコメントを独自にサポートしています。 ただし、通常は 2 つのバリアントのうちの 1 つだけであり、多くの場合、独自のプログラミング言語の規則に対応しています。 その結果、普遍的な慣習がないため、UNECE JSON スキーマ コードと識別子リストでは、そのような独自のコメントが省略されています。

This NDR technical specification is created with the goal of applicability of the JSON schema artefacts for use in OpenAPI specifications. This means that for the implementer of such a specification, the documentation of the individual code or identifier values is important.

この NDR 技術仕様は、JSON スキーマ アーティファクトを OpenAPI 仕様で使用できるようにすることを目的として作成されています。 これは、そのような仕様の実装者にとって、個々のコードまたは識別子の値のドキュメントが重要であることを意味します。

Starting with OpenAPI 3.1 the preferred representation of code lists is an **oneOf** combination of **const** definitions. This allows code names and definitions to be added directly to the definition of each individual code. In addition, further amendments like adding validity periods for individual code values become possible.

OpenAPI 3.1 以降では、コード リストの推奨される表現は、const 定義の oneOf の組み合わせです。 これにより、コード名と定義を個々のコードの定義に直接追加できます。 さらに、個々のコード値の有効期間を追加するなどのさらなる修正が可能になります。

|  |
| --- |
| [R 32|1] |
| The **description** property of the JSON schema specifying a code or identifier list SHALL list the copyright notice information as defined in the CCL. This includes the code  or identifier list name, code or identifier list agency, code or identifier list version, and copyright information. |

10 See https://json-schema.org/understanding-json-schema/reference/generic.html#comments

|  |
| --- |
| [R 33|2] |
| The **title** property of the subschema specifying the **const** definitions holding the values of a code or identifier list SHOULD be the code name value in English language.  The **description** property of the subschema specifying the **const** definitions holding  the values of a code or identifier list SHOULD be the code definition value in English language. |

The following rule defines the representation of code and identifier lists as files.

|  |
| --- |
| [R 34|1] |
| Code lists SHALL be represented in a subschema of the corresponding schema file with the following naming convention:  **$defs/codeList/<Code List Name or Identifier>Type**  with <Code List Name or Identifier> – If a code list identifier exists in the UNTDID, the identifier is given in the format untdid<identifier>. Else, the code list name is stated as assigned by the publishing agency with special characters removed. |

The following example shows a complete code list JSON schema file content.

|  |
| --- |
| [Example]  {  "$schema": "https://json-schema.org/draft/2020-12/schema", "$id": "https://service.unece.org/trade/uncefact/json-  schema/D22A/UNECE\_UNTDID-3131",  "title": "Address type code",  "description": "<<copyright notice information>>", "$defs": {  "codeList": { "untdid3131Type": {  "title": "Address type code", "oneOf": [  {  "const": "1",  "title": "Postal Address"  },  {  "const": "2",  "title": "Fiscal Address"  },  {  "const": "3",  "title": "Physical Address"  },  {  "const": "4",  "title": "Business Address"  },  {  "const": "5",  "title": "Delivery To Address"  },  {  "const": "6",  "title": "Residential Address"  },  { |

|  |
| --- |
| "const": "7",  "title": "Mail To Address"  },  {  "const": "8",  "title": "Postbox Address"  }  ]  }  }  }  } |

|  |
| --- |
| [R 35|1] |
| Identification schemes SHALL be represented in a subschema of the corresponding schema file with the following naming convention:  **$defs/identificationScheme/<Indentification Scheme Name or Identifier>Type**  with < Identification Scheme Name or Identifier> – If an identification scheme identifier exists in the UNTDID, the identifier is given in the format untdid<identifier>. Else, the code or identification scheme name is stated as assigned by the publishing agency with special characters removed. |

The following example shows a complete identification scheme JSON schema file content.

|  |
| --- |
| [Example]  {  "$schema": "https://json-schema.org/draft/2020-12/schema", "$id": "https://service.unece.org/trade/uncefact/json-  schema/D22A/ISO\_639-1-2A", "title": "Language identifier",  "description": "<<copyright notice information>>", "$defs": {  "identificationScheme": { "iso639-1-2AType": {  "title": "Language identifier", "oneOf": [  {  "const": "AR",  "title": "ARABIC"  },  {  "const": "AS", "title": "ASSAMESE"  },  {  "const": "AV",  "title": "AVARIC"  },  {  "const": "AY",  "title": "AYMARA"  },  {  "const": "AZ", "title": "AZERBAIJANI"  },  { |

|  |
| --- |
| "const": "BA, "title": "BASHKIR"  },  {  "const": "BE", "title": "BELARUSIAN"  }  ]  }  }  }  } |

### Restriction and Extension

#### Restriction

The CCTS defines methods of restriction to create e.g. industry specific profiles of the CCL.

CCTS は、作成する制限の方法、すなわちCCL の業界固有のプロファイルを定義します。

One output of this process are the Reference Data Models (RDMs) being published like the Supply-Chain-Reference-Data Model (SCRDM) or the Multi-Modal-Transport- Reference-Data-Model (MMT-RDM).

このプロセスの成果の 1 つは、サプライ チェーン参照データ モデル (SCRDM) やマルチモーダル輸送参照データ モデル (MMT-RDM) のように公開される参照データ モデル (RDM) です。

For data transmission via messages, the method of restriction is also used to restrict cardinalities and values of code or identifier list (p.s. qualified data types are being created in case of restricting values of code or identifier list). A significant part of the standardisation activity of UN/CEFACT has been dealing with this very issue for many years.

メッセージを介したデータ送信の場合、制限の方法は、カーディナリティとコードまたは識別子リストの値を制限するためにも使用されます (p.s. コードまたは識別子リストの値を制限する場合は、修飾されたデータ型が作成されます)。 UN/CEFACT の標準化活動の重要な部分は、まさにこの問題に長年取り組んできました。

As defined in rule [[R 10|1]](#_bookmark27) for each individual layer of data models a separate JSON schema file is published.

ルール [R 10|1] で定義されているように、データ モデルの個々のレイヤーごとに個別の JSON スキーマ ファイルが公開されます。

|  |
| --- |
| [R 36|1] |
| Restrictions to CCTS objects SHALL be represented in a subschema as follows: |
| **Cardinalities** |
| * From 0..1 to 1..1 |
| [Example] "toBeRestrictedType": {  "type": "object", "properties": {  "id": { "type": "string" }  }  },  "restrictingType": {  "$ref": "#/$defs/toBeRestrictedType", "required": ["id"]  } |
| * From 0..1 to 0..0 (forbidden) |
| [Example] "toBeRestrictedType": {  "type": "object", "properties": {  "id": { "type": "string" },  "name": { "type": "string" }  } |

|  |
| --- |
| },  "restrictingType": {  "$ref": "#/$defs/toBeRestrictedType", "properties": {  "id": false  }  } |
| * From 0..unbounded to 0..n with n < unbounded |
| [Example with n=2] "toBeRestrictedType": {  "type": "object", "properties": {  "id": {  "type": "array",  "items": { "type": "string }  }  }  },  "restrictingType": {  "$ref": "#/$defs/toBeRestrictedType", "properties": {  "id": { "maxItems": 2 }  }  } |
| * From 0..unbounded to n..unbounded |
| [Example with n=2] "toBeRestrictedType": {  "type": "object", "properties": {  "id": {  "type": "array",  "items": { "type": "string }  }  }  },  "restrictingType": {  "$ref": "#/$defs/toBeRestrictedType", "properties": {  "id": { "minItems": 2 }  }  } |
| **Restriction of value ranges** |
| [Example restricting content to values with exact 2 fraction digits] "restrictingType": {  "allOf": [  { "$ref": "UNECE-BasicComponents.json#/$defs/udt/amountType" },  { "properties": {  "content": { "pattern": "^.\*\..{2}$" }  }  }  ]  } |
| **Restriction of const** |
| [Example restricting content to a code list subset] "addressType": {  "type": "object", "properties": {  "countryId": { "$ref": "UNECE- BasicComponents.json#/$defs/qdt/countryIdType"} |

|  |
| --- |
| }  },  "restrictingType": { "allOf": [  { "$ref": " #/$defs/addressType" },  { "properties": {  "countryId": { "const": ["CH", "DE", "FR"] }  }  }  ]  } |

The same type of restriction can be applied if restrictions are defined on a lower level.

|  |
| --- |
| [Example]  {  "$defs": { "restriction": {  "allOf": [  {  "$ref": "#/$defs/levelOne"  },  {  "properties": { "oneFirst": {  "properties": { "twoFirst": false  }  }  }  }  ]  },  "levelOne": { "type": "object", "properties": {  "oneFirst": {  "$ref": "#/$defs/levelTwo"  },  "oneSecond": {  "type": "string"  }  }  },  "levelTwo": { "type": "object", "properties": {  "twoFirst": {  "type": "string"  }, |

|  |
| --- |
| "twoSecond": {  "type": "string"  }  }  }  }  } |

**Figure 2: Example for second level restrictions**

#### Extension

The CCTS does not support extensions. Therefore, no NDR rules analogous to the Restrictions chapter can be set up for the CCTS that extend cardinalities, value ranges or **enum**. Should an implementation nevertheless require such an extension, the result is no longer compliant with the artefacts according to this technical specification. Technically, this can be achieved by combining a schema with **anyOf**.

CCTS は拡張機能をサポートしていません。 したがって、カーディナリティ、値の範囲、または列挙を拡張する CCTS に対して、制限の章に類似した NDR ルールを設定することはできません。 それにもかかわらず、実装がそのような拡張を必要とする場合、結果はこの技術仕様によるアーティファクトに準拠しなくなります。 技術的には、これはスキーマを anyOf と組み合わせることで実現できます。

However, especially when implementing OpenAPI specifications, extensions to the properties are needed. For example, to add metadata to the API endpoints.

ただし、特に OpenAPI 仕様を実装する場合は、プロパティの拡張が必要です。 たとえば、メタデータを API エンドポイントに追加します。

|  |
| --- |
| [R 37|1] |
| The BasicComponents SHALL define a JSON subschema for extension as follows: |
| "$defs": { "extensibleType": {  "patternProperties": { "^x-": true}  }  } |

The **extensibleType** allows users to add their own JSON properties to the JSON subschemas. The only rule they have to follow is that they must start with **x-**. This makes it compliant to the extension method defined in the OpenAPI specification. An example can be found in the next section in rule [[R 42|1].](#_bookmark54)

extensibleType を使用すると、ユーザーは独自の JSON プロパティを JSON サブスキーマに追加できます。 従わなければならない唯一の規則は、x- で始まらなければならないということです。 これにより、OpenAPI 仕様で定義されている拡張メソッドに準拠します。 例は、ルール [R 42|1] の次のセクションにあります。

#### Publication and reusing contextualization

The CCL is undergoing a continuous development. This way it contains definitions that are not used any more in newer versions. In order to prevent confusion with published data types that are not used any more the RDM level is the lowest export level for any UN/CEFACT publication.

CCL は継続的に開発されています。 このようにして、新しいバージョンでは使用されなくなった定義が含まれています。 もはや使用されていない公表済みのデータ タイプとの混同を避けるために、RDM レベルは、UN/CEFACT 公表物の最低のエクスポートレベルです。

|  |
| --- |
| [R 38|1] |
| The base of all JSON schema exports SHALL be the RDM level. This means that each underlying CCL basic data type SHALL be profiled and contextualised according to the RDM definition. Only data types that are used in an RDM SHALL be exported. |

If the rules defined in this section are applied to the entire CCL, the resulting JSON artefacts can become complex and very large. This approach creates a high level of traceability of the restrictions and ensures a consistent (re-)use of the individual data types.

このセクションで定義されたルールが CCL 全体に適用される場合、結果として得られる JSON アーティファクトは複雑で非常に大きくなる可能性があります。 このアプローチにより、制限の高レベルのトレーサビリティが作成され、個々のデータ型の一貫した (再) 使用が保証されます。

In a practical application of an API, however, these libraries can be unnecessarily large. Especially if only a subset of the CCL is used.

ただし、API の実際のアプリケーションでは、これらのライブラリが不必要に大きくなる可能性があります。 特に、CCL のサブセットのみが使用されている場合。

Therefore, it can be useful to export "snapshots" of the required (sub-) structures as JSON artefacts. The procedure here corresponds to the XML design principle "Venetian blind": Only one JSON schema file is created, which contains all the required data types for the use case. All properties that are not required are not even exported. Restrictions are kept to a minimum. Compliance with the CCL is mandatory.

したがって、必要な (サブ) 構造の「スナップショット」を JSON アーティファクトとしてエクスポートすると便利です。 ここでの手順は、XML 設計原則「ベネチアン ブラインド」に対応しています。ユース ケースに必要なすべてのデータ型を含む JSON スキーマ ファイルが 1 つだけ作成されます。 必須ではないすべてのプロパティはエクスポートされません。 制限は最小限に抑えられます。 CCL への準拠は必須です。

|  |
| --- |
| [R 39|2] |
| A user community may decide to create "snapshot" JSON schema artefacts for a specific subset of the CCL. A "snapshot" JSON schema artefact SHALL contain all relevant data types that are needed to define the subset. The "snapshot" JSON schema artefact MAY contain additional restrictions and extensions. |

Together with the "snapshot" export, there exist three possible ways of creating JSON schema artefacts:

|  |  |
| --- | --- |
| **Export variant** | **Description** |
| Library export | The library export creates one JSON schema file for each level of contextualisation as they are defined by the UN/CEFACT standards. It creates one large CCL JSON schema representation as a foundation. On top of it, it creates one JSON schema file contextualising and restricting the CCL to the defined RDMs and document-centric structures. Each level may already use restricted data types that are restricted exactly at that level. This needs to be considered when creating this type of export.  **Pro**  The complete CCL, all RDMs as well as all (document-centric) message structure definitions are exported as defined by UN/CEFACT standards. A maximum of re-usable data structures and definitions are created. It assures by design that any implementation is consistent and ready for any process- amendment.  **Contra**  Any implementation needs to handle the huge CCL library as a base import as well as the multi-layer-restrictions as they are defined by UN/CEFACT standards. For example, the eCMR message is defined as a contextualisation of a master message structure for all document-centric messages defined by UN/CEFACT. The contained data structure is process specific contextualisation of a multi modal transport reference data model. The MMT-RDM is a transport specific contextualisation of the Buy-Ship- |

|  |  |
| --- | --- |
|  | Pay reference data model. Moreover, this again is a contextualisation of the underlying CCL.  Thus, an implementation could get rather complex while at the same time achieving a maximum compliance level. |
| Subset export | The subset export follows the same principles as the library export with one major difference: Only the needed data structures of the selected subset are exported. All other data structures are omitted. This way the file size and content is reduced to a minimum set of information, while at the same time keeping all relations available. It is important that all levels of restrictions be taken into account. Only the result of applying all levels of restrictions in hierarchical order is represented in the resulting technical artefact.  **Pro**  In addition to the arguments defined in the library export, the subset export is easier to handle in respect of file size and quantity of data objects.  **Contra**  The complexity of layers of contextualisation is still the same as with the library export. Amendments of the subset lead to changes in the underlying objects. Only those data objects are exported that are needed for a specific subset. When the scope of the subset is widened in a future version, it may need additional objects in the underlying data structures. This means that implementations of the subset need to be updated at all players at the same time. |
| Snapshot export | Content wise the snapshot export is equal to the subset export. The main difference is that the multi-layer-contextualisation over a set of several JSON schema files is removed. Only one single JSON schema file is created that contains all necessary data structures of the snapshot objects. It is comparable with the XML "Ventian Blind" approach. Underlying data objects are still defined (like a party data type). However, they only contain schema objects being used in the snapshot selection.  **Pro**  The complexity for the given snapshot is reduced to a minimum. Only one single self-contained JSON schema file is created. The JSON schema file can easily be used by all common JSON tools as well as OpenAPI design tools. The exported data structures are compliant to the UN/CEFACT standards and reflect "the compilation" of all restrictions and contextualisation.  **Contra**  One self-contained JSON schema file is created for each individual snapshot. If this approach is used in a pre-defined environment, it works quite well. Thus, it is important to clearly define the snapshot content in advance.  Things start to get complicated if in one implementation more than one self- contained JSON schema files are used. Let us assume that for example one self-contained JSON schema file is created for each document-centric |

|  |  |
| --- | --- |
|  | message (as it is done with XML schema files). Each of those JSON schema files defined the underlying data types (e.g. party). In an OpenAPI specification, it is not so easy to combine those multiple schema files into one single OpenAPI file as it may come to conflicts between the underlying data types. The reason is that the same data type with the same name may have a diverging contextualisation between the different JSON schema files. |

**Table 8: Export variants**

|  |
| --- |
| [R 40|1] |
| A UNECE publication SHALL provide a library export on a server being able to handle the necessary requirements for a global community accessing the published artefacts.  In addition, UNECE SHOULD provide an additional snapshot export for each contextualised document ABIE. |
| [Note]  As the $id property of a JSON schema must represent a valid URL aspects of scalability of the provided service have to be taken into account. One option could be to provide the publication in a GIT-compliant repository. |

### ABIE and BBIE representation in JSON Schema

#### General handling of ABIEs and BBIEs

|  |
| --- |
| [R 41|1] |
| Each ABIE SHALL be represented in a JSON subschema. ABIEs that are marked as deprecated from a former version SHALL NOT be represented in a JSON subschema. |
| [Note]  For example an ABIE is defined to be deprecated starting in version D20B. When the JSON schema artefacts for version D21A are exported, the ABIE SHALL NOT be represented in this export. |

|  |
| --- |
| [R 42|1] |
| All ABIE representations in JSON subschemas SHALL include a reference to the  **extensibleType**. |
| [Example] "abieType": {  "title": "The Dictionary Entry Name", "description": "The description", "type": "object",  "properties": {  "p1": { "type": "string" }  }, |

|  |
| --- |
| "required": ["p1"],  "$ref": "UNECE-BasicComponents.json#/$defs/extensibleType", "unevaluatedProperties": false  }  }  [Example of a valid JSON object]  {  "p1": "value",  "x-addedStringProperty": "added value",  "x-addedObjectProperty": { "content": "a123"}  }  [Example of an invalid JSON object]  {  "p1": "value", "addedStringProperty": "added value"  } |

|  |
| --- |
| [R 43|2] |
| Extension property names SHOULD follow the same naming conventions as defined in this technical specification. |

#### ASBIE representation in JSON Schema supporting document based and resource-based information

The CCTS was invented for the purpose of standardising and modelling classic EDI messages. Even today, document-based data exchange is still predominant, especially in the B2B and B2A environment.

CCTS は、従来の EDI メッセージの標準化とモデル化を目的として考案されました。 今日でも、特に B2B および B2A 環境では、ドキュメント ベースのデータ交換が依然として主流です。

As described at the beginning of this technical specification, REST APIs are characterised by the fact that they are not based on the exchange of business documents, but on the management of resources. This means that, for example, business partner information can be managed separately from transaction data such as an invoice or a transport order. In CCTS, these are all the places where ABIEs are associated with each other in the form of ASBIEs.

この技術仕様の冒頭で説明したように、REST API はビジネス ドキュメントの交換ではなく、リソースの管理に基づいているという特徴があります。 これは、たとえば、ビジネスパートナ情報を、請求書や輸送指図などの取引データとは別に管理できることを意味します。 CCTS では、これらはすべて、ABIE が ASBIE の形式で相互に関連付けられる場所です。

With the aim of supporting REST APIs via the JSON schema artefacts, it is precisely at this point that the option of switching from document-centred to resource-centred data exchange must be supported.

JSON スキーマ アーティファクトを介して REST API をサポートする目的で、まさにこの時点で、ドキュメント中心からリソース中心のデータ交換に切り替えるオプションをサポートする必要があります。

Resource-based data management means that resources must have unique identifiers. Therefore, only those ABIEs can be converted to resources that have a unique identifier. Using this unique identifier represented as an URI, the information about a buyer in an order can be retrieved following the URI to the party information of the buyer.

リソースベースのデータ管理とは、リソースに一意の識別子が必要であることを意味します。 したがって、一意の識別子を持つリソースに変換できるのは、これらの ABIE のみです。 URI として表されるこの一意の識別子を使用して、注文の購入者に関する情報を、購入者のパーティ情報への URI に従って取得できます。

|  |
| --- |
| [R 44|1] |
| The BasicComponents SHALL define a JSON subschema for resource based data exchange as follows: |

|  |
| --- |
| "$defs": { "resourceType": {  "type": "string",  "format": "uri"  }  } |

|  |
| --- |
| [R 45|1] |
| All ASBIEs whose ABIEs contain an identifier SHALL be modelled using an **oneOf**  choice between the **resourceType** and the associated ABIE. All other ASBIEs SHALL be referenced directly.  In both cases, the defined cardinality SHALL be observed.  To stay focused, title, description etc. are not shown in the following example. |
| [Example] "$defs": {  "invoiceType": { "type": "object", "properties": {  "buyer": {  "oneOf": [  { "$ref": "UNECE-BasicComponents.json#/$defs/resourceType" },  { "$ref": "#/$defs/partyType" }  ]  }  },  "required": [ "buyer" ],  "$ref": "UNECE-BasicComponents.json#/$defs/extensibleType", "unevaluatedProperties": false  },  "partyType": { "type": "object", "properties": {  "id": {  "type": "array", "items": {  "$ref": "UNECE-BasicComponents.json#/$defs/udt/identifierType"  }  },  "name": { "type": "string" },  "postalTradeAddress": { "$ref": "#/$defs/addressType" }  },  "$ref": "UNECE-BasicComponents.json#/$defs/extensibleType", "unevaluatedProperties": false  },  "addressType": { "type": "object", "properties": {  "street": { "type": "string"},  "city": { "type": "string"},  "postalCode": { "type": "string"},  "countryCode": { "$ref": "UNECE- BasicComponents.json#/$defs/qdt/countryIdType"}  },  "$ref": "UNECE-BasicComponents.json#/$defs/extensibleType", "unevaluatedProperties": false  }  } |

### Fostering implementation

#### Compatibility with JSON schema draft before version 2020-12

As stated earlier in the document, version 2020-12 of the JSON schema Draft is used in this NDR.

ドキュメントで前述したように、JSON スキーマ ドラフトのバージョン 2020-12 がこの NDR で使用されます。

Nevertheless, many previous JSON schema Draft versions are still in use in practice and tool support for the current version is not yet very high. In some places, the previous version and the current version are not compatible. This means that tools that do not yet support the latest version could very likely have difficulties in use.

それにもかかわらず、多くの以前の JSON スキーマのドラフト バージョンはまだ実際に使用されており、現在のバージョンのツール サポートはまだそれほど高くありません。 一部の場所では、以前のバージョンと現在のバージョンは互換性がありません。 これは、最新バージョンをまだサポートしていないツールを使用すると、問題が発生する可能性が非常に高いことを意味します。

However, this NDR must be independent of the capability of certain tools. Furthermore, it is not intended to be based on an old version that has already been revised due to practical requirements.

ただし、この NDR は、特定のツールの機能とは無関係である必要があります。 さらに、実際の要件のためにすでに改訂された古いバージョンに基づくことを意図したものではありません。

The following rules describe how to achieve higher compatibility for such tools. Since these rules limit the possibilities of the generated JSON schema, these rules should only be applied transitionally, at most until the publication of CCTS library version D25A.

次の規則は、そのようなツールの互換性を高める方法を説明しています。 これらのルールは、生成された JSON スキーマの可能性を制限するため、これらのルールは、せいぜい CCTS ライブラリ バージョン D25A の公開まで、一時的にのみ適用する必要があります。

|  |
| --- |
| [R 46|2] |
| This rule can be applied transitionally up to and including the publication of library version D25A. Thereafter, modelling according to this rule is no longer conform to this NDR. It shall be applied to the following rules:  [[R 29|1]](#_bookmark46):  A list of coded values or identifiers SHALL be modelled using **enum**, and Not as **const**. The description of each coded value or identifier SHALL be put in the **description** of  the corresponding type as well as a comment after each **enum** value as shown in the following example: |
| [Example]  {  "$schema": "https://json-schema.org/draft/2020-12/schema", "$id": "https://service.unece.org/trade/uncefact/json-  schema/D22A/UNECE\_UNTDID-3131",  "title": "Address type code",  "description": "<<copyright notice information>>", "$defs": {  "codeList": { "untdid3131Type": {  "title": "Address type code", "description": "Applicable codes:   * '1' – Postal Address * '2' – Fiscal Address * '3' – Physical Address * '4' – Business Address * '5' – Delivery To Address * '6' – Residential Address * '7' – Mail To Address * '8' – Postbox Address", "enum": [   "1", # Postal Address "2", # Fiscal Address "3", # Physical Address  "4", # Business Address |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| } | } | } | } | ] | "5", # Delivery To Address "6", # Residential Address "7", # Mail To Address  "8" # Postbox Address |
| [[R 27|1]](#_bookmark44) SHALL be modelled accordingly. | | | | | |

|  |
| --- |
| [R 47|2] |
| This rule can be applied transitionally up to and including the publication of library version D25A. Thereafter, modelling according to this rule is no longer conform to this NDR. It shall be applied to the following rules:  [[R 42|1]](#_bookmark54):  The reference to the **extensibleType** in an ASBIE relationship SHALL be embedded  in an **allOf**: |
| [Example] "abieType": {  "title": "The Dictionary Entry Name", "description": "The description", "type": "object",  "properties": {  "p1": { "type": "string" }  },  "required": ["p1"], "allOf": [{  "$ref": "UNECE-BasicComponents.json#/$defs/extensibleType"  }],  "unevaluatedProperties": false  }  } |

#### Hints for tool developers and designers when specifying real-life guidelines

Even if it were technically possible, data exchange standards are usually not implemented to the full extent. Instead, subsets tailored to the exact problem, e.g. the reference data models, are defined and implemented.

技術的に可能であったとしても、通常、データ交換標準は完全には実装されていません。 代わりに、正確な問題に合わせて調整されたサブセット。 参照データ モデルが定義され、実装されます。

The aim of using standards is always to create interoperability between the systems, processes and organisations involved. Different levels of interoperability can be distinguished, but these will not be discussed in detail in the course of this document. One possible definition can be found, for example, in the European standard EN16931 for electronic invoices to public sector customers.

標準を使用する目的は常に、関連するシステム、プロセス、および組織間の相互運用性を作成することです。 さまざまなレベルの相互運用性を区別できますが、このドキュメントでは詳しく説明しません。 考えられる定義の 1 つとして、たとえば、公共部門の顧客に対する電子請求書に関する欧州規格 EN16931 があります。

From an organisational and process perspective, the most important and at the same time the most minimal goal is to achieve semantic interoperability. Without this, an automated, cross-organisational exchange of information is not possible.

組織とプロセスの観点から、最も重要であると同時に最も最小限の目標は、セマンティックな相互運用性を実現することです。 これがなければ、自動化された組織間の情報交換は不可能です。

If (only) semantic interoperability is taken as a maxim, the frequently different syntax representations in (fixed) technical data structures often lead to a high mapping effort. A good example of this are the many industry and company profiles of one and the same EDIFACT message that exist in practice today.

セマンティックな相互運用性 (のみ) が格言として取られる場合、(固定された) 技術データ構造の頻繁に異なる構文表現は、多くの場合、多大なマッピング作業につながります。 これの良い例は、今日実際に存在する 1 つの同じ EDIFACT メッセージの多くの業界および企業プロファイルです。

In order to solve this problem, two different approaches are mainly pursued in practice:

* + - 1. The definition of reference data models and reference message structures.
      2. The definition of semantic vocabularies based on JSON-LD.

　この問題を解決するために、実際には主に 2 つの異なるアプローチが追求されています。

1. 参照データ モデルと参照メッセージ構造の定義。

2. JSON-LD に基づく意味語彙の定義。

The first approach is followed by these JSON Schema NDR. It is more conservative because it can directly satisfy the many requirements that exist today from the exchange of business documents. Nevertheless, the modelling also enables use on the basis of resources: the information components in the data models can be declared as resources and thus used in modern technologies such as REST APIs while maintaining the harmonisation work carried out over decades.

最初のアプローチの後に、これらの JSON スキーマ NDR が続きます。 ビジネス文書の交換から今日存在する多くの要件を直接満たすことができるため、より保守的です。 それにもかかわらず、モデリングはリソースに基づいた使用も可能にします。データモデルの情報コンポーネントはリソースとして宣言できるため、REST API などの最新のテクノロジーで使用しながら、数十年にわたって行われてきた調和作業を維持できます。

The second approach includes the idea that (any) technical formats can link to the semantic definitions and thus technical interoperability can be achieved "on-the-fly". Depending on the application scenario, this can be implemented well in practice and offers corresponding advantages. In particular, since the implementer can use generalisation and inheritance to model the information components according to his own requirements and ensure semantic interoperability by means of a link to the vocabulary. This approach is all the more relevant the more organisations or data providers are involved in the respective process. The implementation of a booking portal or tracking and tracing applications are good examples where this approach shows its strengths.

2 番目のアプローチには、(任意の) 技術フォーマットをセマンティック定義にリンクできるため、技術的な相互運用性を「オンザフライ」で実現できるという考えが含まれます。 アプリケーション シナリオに応じて、これは実際に適切に実装でき、対応する利点が得られます。 特に、実装者は一般化と継承を使用して、独自の要件に従って情報コンポーネントをモデル化し、語彙へのリンクによってセマンティックな相互運用性を確保できるためです。 このアプローチは、より多くの組織またはデータ プロバイダーがそれぞれのプロセスに関与するほど、より関連性が高くなります。 予約ポータルまたは追跡および追跡アプリケーションの実装は、このアプローチの強みを示す良い例です。

However, this approach poses new challenges for the implementer if, for example, he wants to map the exchange of legally required business documents in a legally compliant manner in this way. The first approach is also preferable if there are data forwarders in the processes, i.e. the actual information is not exchanged synchronously between two participants. If information is routed via several intermediate stations (e.g. a solution provider network), meta information is needed for this purpose, which requires a minimum hierarchical message structure. (e.g. envelope with the routing information and the actual exchange information contained therein).

ただし、このアプローチは、たとえば、法的に必要なビジネス文書の交換をこのように法的に準拠した方法でマッピングしたい場合、実装者に新たな課題をもたらします。 プロセスにデータ フォワーダーがある場合、つまり実際の情報が 2 つの参加者間で同期的に交換されない場合も、最初のアプローチが適しています。 情報が複数の中間ステーション (ソリューション プロバイダー ネットワークなど) を介してルーティングされる場合、この目的のためにメタ情報が必要であり、これには最小限の階層メッセージ構造が必要です。 (たとえば、ルーティング情報と実際の交換情報が含まれるエンベロープ)。

In summary, the implementation effort should correspond to the implementation requirements. During implementation, the scope of information for data exchange in the　respective process is defined.

要約すると、実装作業は実装要件に対応する必要があります。 実装中、それぞれのプロセスでのデータ交換に関する情報の範囲が定義されます。

It is also determined whether an ABIE or BBIE from an ASBIE should continue to be transmitted as a hierarchical structure or should be considered as an independent resource. Information on a business partner is given as an example.

また、ＡＳＢＩＥからのＡＢＩＥまたはＢＢＩＥが、階層構造として送信され続けるべきか、または独立したリソースと見なされるべきかが決定される。 取引先情報は一例です。

In typical business documents, this is transmitted in detail (hierarchically): for example, the name and full address of a buyer and seller. In a resource-based approach, the resource "party" would be managed independently and the corresponding places in the business document would only refer to it; similar to a customer number, location number or tax number.

典型的なビジネス文書では、これは詳細に (階層的に) 送信されます。たとえば、買い手と売り手の名前と完全な住所です。 リソースベースのアプローチでは、リソースの「関係者」は独立して管理され、ビジネス文書内の対応する場所はそれを参照するだけです。 顧客番号、ロケーション番号、または税番号に似ています。

The rules defined in this NDR allow the implementer or designer to decide flexibly at which point which form of modelling should be used. Thus, the JSON schema files according to these specifications represent templates for the implementation. With the help of appropriate tools, these JSON schema templates can be simplified during implementation: For the ASBIE, it is decided in each case whether the hierarchical structure or the resource- based structure should be retained.

この NDR で定義された規則により、実装者または設計者は、どの時点でどの形式のモデリングを使用する必要があるかを柔軟に決定できます。 したがって、これらの仕様に従った JSON スキーマ ファイルは、実装のテンプレートを表します。 適切なツールの助けを借りて、これらの JSON スキーマ テンプレートは実装中に簡素化できます。

The resulting contextualised JSON schemas are thus much smaller and easier to implement. Nevertheless, they are conform to the JSON schemas published according to this specification.

したがって、結果として得られるコンテキスト化された JSON スキーマは、はるかに小さく、実装が容易になります。 それでも、この仕様に従って公開されている JSON スキーマに準拠しています。

|  |
| --- |
| [R 48|2] |
| JSON schemas used in implementation MAY be contextualised subsets of the published JSON schema artefacts. This explicitly means that at the ASBIE level decisions have been made if a hierarchical or resource based approach is used. Consequently, the **OneOf** choice  between those two options may be optimised so that only the remaining reference is used: |
| [Example: Original] "provider": {  "title": "Document\_ Authentication. Provider. Trade\_ Party", "description": "The trade party providing this document  authentication.", "oneOf": [  { "$ref": "#/$defs/tradePartyType" },  { "$ref": "#/$defs/resourceType" }  ]  }  [Example: Contextualisation] "provider": {  "title": "Document\_ Authentication. Provider. Trade\_ Party", "description": "The trade party providing this document  authentication.",  "$ref": "#/$defs/tradePartyType"  } |

#### Referencing the Github Repository in an OpenAPI specification

**ドイツ語**

Der Entwicklungsprozess für jeden Datenaustausch kann wenigstens in die Phasen Design- Time und Run-Time unterschieden werden.

各データ交換の開発プロセスは、少なくとも設計時と実行時のフェーズに分けることができます。

Bei der Run-time, der eigentlichen Durchführung von Konvertierungen und Datenübertragungen ist Effizienz und Ressourcenschonung ein Wesentliches Ziel.

ランタイムでは、変換とデータ転送の実際の実装、効率、およびリソースの節約が重要な目標です。

Bei einer Run-Time-Umgebung ist der Verweis auf Ressourcen auf externen Servern, über die keine Kontrolle besteht, in der Regel als kritisch einzustufen.

ランタイム環境では、通常、制御できない外部サーバー上のリソースを参照することが重要です。

Dies ist während der Spezifikations- und Entwicklungszeit – der Design-Time, anders. Bei einer OpenAPI-Spezifikation handelt es sich um ein Dokument der Design-Time. Damit ist eine direkte Referenzierung der UN/CEFACT-Publikationen über das entsprechende Repository denkbar.

これは、仕様と開発時 (設計時) では異なります。 OpenAPI 仕様は設計時のドキュメントです。 したがって、対応するリポジトリを介して UN/CEFACT 出版物を直接参照することが考えられます。

Werden die JSON Schema Artefakte oder OpenAPI Templates direct aus einem github – Repository verlinkt, ist dabei zu beachten, dass die Raw-Variante der Quelle verwendet wird. Dadurch ändert sich auch der Link zu der entsprechenden Ressource.

JSON スキーマ アーティファクトまたは OpenAPI テンプレートが github リポジトリから直接リンクされている場合は、ソースの raw バージョンが使用されることに注意してください。 これにより、対応するリソースへのリンクも変更されます。

|  |
| --- |
| [Example]  For example, if a file is located in the following repository:  https://**github.com**/uncefact/spec-JSONschema/**blob**/main/ JSONschema2020-12/snapshot/BuyShipPay/D22A/Regulatory/eCert/PrefCoO/ UNECE-BSPPreferentialCoO.json  its raw variant is to be referenced via the following link:  https://**raw.githubusercontent.com**/uncefact/spec-JSONschema/main/ JSONschema2020-12/snapshot/BuyShipPay/D22A/Regulatory/eCert/PrefCoO/ UNECE-BSPPreferentialCoO.json |

# Appendix A: Examples

Printed JSON schema files of a realistic example can be very large, especially because of the code lists used. Therefore, we have not included an example here.

現実的な例の印刷された JSON スキーマ ファイルは、特にコード リストが使用されているため、非常に大きくなる可能性があります。 そのため、ここには例を含めていません。

However, examples can be found on the web at the following address:

ただし、例は Web の次のアドレスにあります。

<https://github.com/uncefact/spec-JSONschema/examples>

# Appendix B: Naming and Design Rules List

|  |  |
| --- | --- |
| **Rule #** | **Rule** |
| [R 1|1] | Conformance SHALL be determined through adherence to the content of the normative sections and rules. Furthermore, each rule is categorized to indicate the intended audience for the rule by the following:   1. Rules, which must not be violated. Else, conformance and interoperability is lost. 2. Rules, which may be modified, while still conformant to the NDR structure. |
| [R 2|1] | In the scope of this specification, a JSON schema is a file that complies to a JSON schema definition as defined at https://json-schema.org. It may include subschemas defined in the **$defs** section. A JSON schema fragment means both the overall  JSON schema as well as each of its included subschemas. |
| [R 3|1] | Each JSON schema SHALL be declared to be a “JSON Draft 2020-12 schema ” with  the appropriate **$schema** string property defined as **https://json- schema.org/draft/2020-12/schema.** |
| [R 4|2] | In section 3.8.1 a set of rules is defined that allows to achieve compatibility with many tools, that do not yet support JSON schema version 2020-12. This set of rules MAY be applied in a publication or the resulting schemas may be published as a  second set of JSON schemas marked as "deprecated compatibility set". |
| [R 5|1] | Each JSON schema SHALL contain a "**title**" annotation. It SHALL be an overall description title. |
| [R 6|1] | Each JSON schema SHALL contain a "**description**" annotation. It contains an  overall description for that file as well as copyright information. |
| [R 7|1] | Each declared Document and Library ABIE definitions and their BBIE and ASBIE members SHALL contain a "**title**" annotation and a "**description**" annotation. The "title" annotation SHALL be the CCTS Dictionary Entry name for the BIE. If there exists a contextualised business name, it SHALL be used instead. "**description**" annotation shall be the CCTS definition value. Members of  enums SHALL NOT contain the "**title**" or the "**description**" annotation. |
| [R 8|1] | The "**unevaluatedProperties**" property of each JSON schema fragment SHALL be set to false, excluding subschemas for primitive data types, unqualified data types and qualified data types. For subschemas specifying primitive data types,  unqualified data types or qualified data types the "**unevaluatedProperties**" property SHALL be stated as defined in this document. |
| [R 9|1] | The JSON schema file names SHALL NOT contain a version information. Differences in versions are only indicated by $id and the folder structure in which the JSON  schema artefacts are located. |
| [R 10|1] | Each JSON schema being published by user groups or standardisation organisations SHALL contain an identifier for the schema in the appropriate $id URI property.  JSON schema exports that are only used in a closed environment (e.g. for testing) do NOT NEED to contain the $id property. The URI SHALL follow the following format:  **"$id": "<basepath>/<variant>/<domain>/<version>[/<RDM>]/<BIE>"**  with <basepath> identifying the originator. For UNECE artefacts that is  "**https://github.com/uncefact/spec-JSONschema**"  <version> in the UNECE publication format e.g. "D22A"  <variant> representing the JSON schema draft version and the export variant. e.g. "JSONschema2020-12/library"  <domain> like "BuyShipPay"  <BIE> with one  - distinct name for each document ABIE without a file extension |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | * name for all BBIE components: "BasicComponents" * distinct name for every RDM set of Library ABIE components * distinct name for each extension collection   <RDM> For the snapshot variant additional structuring is allowed.  The JSON schema file name SHALL be build with the following format:  **<originator>-<abbreviation>.json**  with  **<originator>** identifying the originator. For UNECE artefacts, it SHALL be **UNECE**.  **<abbreviation>** identifying the RDM set of Library ABIE components  If a contextualised business name exists for a message structure, it SHALL be used instead. If a .json-File with this name already exists, the message model name  SHALL be added, separated by another hyphen. | | | | | |
| [R 11|1] | The BasicComponents JSON schema file SHALL contain all subschemas for primitive data types, unqualified data types as well as qualified data types. | | | | | |
| [R 12|1] | A property is a name/value pair inside a JSON object. The property name is the key or name part of the property. The property value is the value part of the property. | | | | | |
| [R 13|1] | JSON property names SHALL be derived from Dictionary Entry Names (DEN).  In e.g. in a BBIE the DEN contains the DEN of the surrounding ABIE, it SHALL be removed. If by applying the NDR rules words in the DEN are duplicated, the duplication SHALL be removed. | | | | | |
| [R 14|1] | Any special characters such full stops | **.** | and underscores | **\_** | SHALL be removed | |
| from the underlying Dictionary Entry Name. If a digit (0-9) was before and another | | | | | |
| digit after the white space, the white space SHALL be replaced by a hyphen - | | | | | . |
| [R 15|1] | JSON property names SHALL be lower camel-cased ASCII strings and JSON schema compliant: The character after a white space shall be a capital letter. Capital letters in the DEN SHALL NOT be kept. | | | | | |
| [R 16|1] | The abbreviations and acronyms SHALL be used as defined in [Table 4.](#_bookmark32)  [[R 15|1]](#_bookmark31) SHALL be taken into account. | | | | | |
| [R 17|1] | The Object Class Term **"Identification Scheme"** SHALL be represented as  **"Scheme"**. [[R 15|1]](#_bookmark31) SHALL be taken into account. | | | | | |
| [R 18|1] | Primitive data types (PDT) SHALL be represented in JSON schema, as stated in [Table 6.](#_bookmark38) They SHALL be placed under **$defs/pdt/**. | | | | | |
| [R 19|1] | Unqualified data types SHALL be represented in subschemas. **"Type"** as part of the  Dictionary Entry Name SHALL be retained. | | | | | |
| [R 20|1] | The CCTS content property SHALL be represented in a subschema with the name  **"content"**. Its data type SHALL use the underlying PDT. The content-property  SHALL be required. | | | | | |
| [R 21|1] | Property names of supplementary components SHALL NOT repeat the JSON subschemas property name. | | | | | |
| [R 22|1] | Supplementary components may reference to code lists and/or identification schemes. In this case, the JSON property SHALL reference the appropriate code list or identification scheme as defined in section 3.5.5 Other Qualified Data Types. | | | | | |
| [R 23|1] | Unqualified data types SHALL be represented in subschemas as shown in [Table 7.](#_bookmark42) The **title** and **description** properties are not shown in the following table. Instead, they are indicated with the placeholder **<title and description>** as those can change over time. They SHALL be published in alignment with rules [[R](#_bookmark21) [5|1],](#_bookmark21) [[R 6|1],](#_bookmark22) and [[R 7|1].](#_bookmark23)  They SHALL be placed under **$defs/udt**. | | | | | |
| [R 24|1] | The **"Date Mandatory\_ Date Time. Type"** SHALL be replaced by the  **formattedDateTimeType**. | | | | | |

|  |  |
| --- | --- |
| [R 25|1] | The **"Time Only\_ Formatted\_ Date Time. Type"** SHALL be replaced  by the **formattedDateTimeType**. |
| [R 26|1] | The **"Formatted\_ Date Time. Type"** SHALL be represented as follows.  **"formattedDateTimeType": {**  **<<title and description>> "oneOf": [**  **{ "type": "string", "format": "date-time" },**  **{ "type": "string", "format": "time" },**  **{ "type": "string", "format": "date" },**  **{ "type": "string", "format": "duration" },**  **{ "type": "object", "properties": {**  **"content": { "type": "string" }, "format": { "$ref": "UNECE\_UNTDID2379-**  **JSON.json#/$defs/codeList/untdid2379JsonType" }**  **},**  **"required": ["content", "format"]**  **}**  **]**  **}** |
| [R 27|1] | Based on the code list **"UNTDID 2379"** an additional code list **"UNTDID 2379 json"** SHALL be specified. All format definitions that are already represented in  their meaning by existing JSON date and time formats SHALL be omitted. This code list SHALL be maintained in accordance with UNTDID 2379. See R27 for details. |
| [R 28|1] | Each QDT that does not fall under section [3.5.4](#_bookmark43) SHALL be restricted according to its definition applying the method described in section [3.6.1.](#_bookmark49) |
| [R 29|1] | Each QDT SHALL be represented in a subschema. If code or id values are specified locally, they SHALL be as an oneOf combination of const definitions. They SHALL NOT be specified as enum arrays. Each code value SHALL be represented as a string type. If the values of codes and ids are organised in code and identification  schemes the corresponding JSON schema SHALL refer to the appropriate code list or identification scheme. |
| [R 30|1] | Each code list and identification scheme SHALL be specified in a separate JSON schema file.  A JSON schema file SHALL be created for each code list and identification scheme being used. Its name SHALL represent the name of the code list or identification scheme and SHALL be unique with the following form:  **<Code List Agency Name>\_<Code List Name or Identifier>.json**  **<Identification Scheme Agency Name>\_<Identification Scheme Name or Identifier>.json**  Where:   * All special characters SHALL be removed from the name. A period . in the version number is replaced by the letter **p**. * <Code List Agency Name> – Agency that maintains the code list. * <Identification Scheme Agency Name> – Agency that maintains the identification scheme. * <Code List Name or Identifier> – If a code list identifier exists in the UNTDID, the identifier is given in the format **UNTDID<identifier>**. Else, the code list   name is stated as assigned by the publishing agency. |

|  |  |
| --- | --- |
|  | * <Identification Scheme Name or Identifier> – If an identification scheme identifier exists in the UNTDID, the identifier is given in the format **UNTDID<identifier>**. Else, the identification scheme name is stated as assigned by the publishing agency.   The file SHALL be placed in a subfolder **codelists** of the export path. The **$id**  property SHALL reflect this subfolder structure. |
| [R 31|2] | It is a clear goal to keep the JSON schema artefacts as compatible with code lists and identification schemes as possible. For this reason the code list version and identification scheme version is neither part of the .json filename nor part of the type name. Nevertheless, it is part of the $id, so that JSON schema files can be used for differentiating versions if needed. If for some reason more than one version of a code list or identification scheme needs to be used in a specific scenario, the **<Code List Version>** or **<Identification Scheme Version>** SHOULD be added to the file name in the following format:  **<Code List Agency Name>\_<Code List Name or Identifier>\_<Code List Version>.json**  **<Identification Scheme Agency Name>\_<Identification Scheme Name or Identifier>\_<Identification Scheme**  **Version>.json** |
| [R 32|1] | The **description** property of the JSON schema specifying a code or identifier list SHALL list the copyright notice information as defined in the CCL. This includes  the code or identifier list name, code or identifier list agency, code or identifier list version, and copyright information. |
| [R 33|2] | The **title** property of the subschema specifying the **const** definitions holding the values of a code or identifier list SHOULD be the code name value in English language. The **description** property of the subschema specifying the **const**  definitions holding the values of a code or identifier list SHOULD be the code definition value in English language. |
| [R 34|1] | Code lists SHALL be represented in a subschema of the corresponding schema file with the following naming convention:  **$defs/codeList/<Code List Name or Identifier>Type**  with <Code List Name or Identifier> – If a code list identifier exists in the UNTDID, the identifier is given in the format untdid<identifier>. Else, the code list name is stated as assigned by the publishing agency with special characters removed. |
| [R 35|1] | Identification schemes SHALL be represented in a subschema of the corresponding schema file with the following naming convention:  **$defs/identificationScheme/<Indentification Scheme Name or Identifier>Type**  with < Identification Scheme Name or Identifier> – If an identification scheme identifier exists in the UNTDID, the identifier is given in the format untdid<identifier>. Else, the code or identification scheme name is stated as assigned  by the publishing agency with special characters removed. |
| [R 36|1] | Restrictions to CCTS objects SHALL be represented in a subschema as follows: Cardinalities   * From 0..1 to 1..1 * From 0..1 to 0..0 (forbidden) * From 0..unbounded to 0..n with n < unbounded * From 0..unbounded to n..unbounded Restriction of value ranges   Restriction of enums |

|  |  |
| --- | --- |
| [R 37|1] | The BasicComponents SHALL define a JSON subschema for extension as follows: |
|  | "$defs": { "extensibleType": {  "patternProperties": { "^x-": true}  }  } |
| [R 38|1] | The base of all JSON schema exports SHALL be the RDM level. This means that each underlying CCL basic data type SHALL be profiled and contextualised  according to the RDM definition. Only data types that are used in an RDM SHALL be exported. |
| [R 39|2] | A user community may decide to create "snapshot" JSON schema artefacts for a specific subset of the CCL. A "snapshot" JSON schema artefact SHALL contain all relevant data types that are needed to define the subset. The "snapshot" JSON  schema artefact MAY contain additional restrictions and extensions. |
| [R 40|1] | A UNECE publication SHALL provide a library export on a server being able to handle the necessary requirements for a global community accessing the published artefacts.  In addition, UNECE SHOULD provide an additional snapshot export for each contextualised document ABIE. |
| [R 41|1] | Each ABIE SHALL be represented in a JSON subschema. ABIEs that are marked as deprecated from a former version SHALL NOT be represented in a JSON  subschema. |
| [R 42|1] | All ABIE representations in JSON subschemas SHALL include a reference to the  **extensibleType**. |
| [R 43|2] | Extension property names SHOULD follow the same naming conventions as defined in this technical specification. |
| [R 44|1] | The BasicComponents SHALL define a JSON subschema for resource based data exchange as follows:  "$defs": { "resourceType": {  "type": "string",  "format": "uri"  }  } |
| [R 45|1] | All ASBIEs whose ABIEs contain an identifier SHALL be modelled using an  **oneOf** choice between the **resourceType** and the associated ABIE. All other ASBIEs SHALL be referenced directly.  In both cases, the defined cardinality SHALL be observed. |
| [R 46|2] | This rule can be applied transitionally up to and including the publication of library version D25A. Thereafter, modelling according to this rule is no longer conform to this NDR. It shall be applied to the following rules: |
| [[R 29|1]](#_bookmark46):  A list of coded values or identifiers SHALL be modelled using **enum**, and Not as **const**. The description of each coded value or identifier SHALL be put in the **description** of the corresponding type as well as a comment after each **enum** value as shown in the example.  [[R 27|1]](#_bookmark44) SHALL be modelled accordingly. |
| [R 47|2] | This rule can be applied transitionally up to and including the publication of library version D25A. Thereafter, modelling according to this rule is no longer conform to this NDR. It shall be applied to the following rules: |

|  |  |
| --- | --- |
|  | [R 42|1]:  The reference to the **extensibleType** in an ASBIE relationship SHALL be  embedded in an **allOf.** |

# Appendix C: Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| ASCII | American Standard Code for Information Interchange |
| ABIE | Aggregate Business Information Entity – a term from CCTS that describes an information class such as “consignment” |
| API | Application Programming Interface – a term that references a machine- to-machine interface. |
| ASBIE | Association Business Information Entity – a term from CCTS that defines a directed relationship from source ABIE to target ABIE – e.g. “consignee” as a relationship between “consignment” and “party” |
| B2A | Business-to-Administration |
| B2B | Business to Business |
| BBIE | Basic Business Information Entity – a term from CCTS that describes a property of a class such as party.name |
| BIE | Business Information Entity |
| CCL | Core Component Library |
| CCT | Core Component Type |
| CCTS | Core Component Technical Specification – a UN/CEFACT specification document that described the information management metamodel. |
| CDT | Core Data Type. A value domain for a BBIE that is a simple type such as “text” or “code” |
| DEN | Dictionary Entry Name |
| EN16931 | Semantic data model of the core elements of an electronic invoice (the European Norm). |
| IETF | Internet Engineering Task Force |
| JSON | JavaScript Object Notation – an IETF document syntax standard in common use by web developers for APIs. |
| JSON-LD | JSON-Linked Data – a JSON standard for linked data graphs / semantic vocabularies. |
| NDR | Naming & Design Rules – a set of rules for mapping one representation (e.g. RDM) to another (e.g. JSON-LD) |
| OpenAPI | An open source standard, language-agnostic interface to RESTful APIs. |
| OWL | Web Ontology Language |
| PDT | Primitive data types |
| PHP | Hypertext Pre-processor |
| QDT | Qualified Data Type. A value domain for a BBIE that is a constrained version of a CDT. Most often used with the “code” type – for example “country code” |
| RDF | Resource Description Framework – a W3C semantic web standard |
| RDM | Reference Data Model- a UN/CEFACT semantic output. |
| RESTful API | See REST API |
| REST API | Representation State Transfer Application Programming Interface,  a.k.a. RESTful API |
| RFC | Request for Comments |
| SDO | Standards Development Organisation |
| UDT | Unqualified data type |

|  |  |
| --- | --- |
| **Term** | **Definition** |
| UN/CEFACT | United Nations Centre for Trade Facilitation and Electronic Business |
| UNECE | United Nations Economic Commission for Europe |
| URI | Uniform Resource Identifier – a namespace qualified string of characters that unambiguously identify a resource. AURL is one type of URI. |
| URL | Uniform Resource Locator – the web address of a resource. |
| UNTDID | United Nations Trade Data Interchange Directory |
| XML | Extensible Mark-up Language |
| XMI | Xml Metadata Interchange - a well-established OMG standard for exchange of UML models between different tools. |

**Table 9 - Glossary**