**UN/CEFACT**

United Nations

Centre for Trade Facilitation and Electronic Business

(UN/CEFACT)

Methodologies and Technologies Programme Development Area

Specifications Domain

Guidance Document

JSON-LD Vocabulary Publishing

**SOURCE:** RDM2API Project Team

**ACTION:** Draft for Public Review

**DATE:** 10 January 2021

**STATUS:** **First revision 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.

Contents

[Change Log 4](#_Toc61884716)

[Glossary 4](#_Toc61884717)

[1 Introduction 6](#_Toc61884718)

[2 Requirements 6](#_Toc61884719)

[3 Reference Data Model Representation 8](#_Toc61884720)

[3.1 Entity mapping 8](#_Toc61884721)

[3.2 Data Types 8](#_Toc61884722)

[3.3 De-duplication 9](#_Toc61884723)

[3.4 Primary identifier mapping 9](#_Toc61884724)

[3.5 UN/CEFACT metadata 9](#_Toc61884725)

[3.6 Business domain granularity 10](#_Toc61884726)

[3.7 Versioning 10](#_Toc61884727)

[4 Code List Representation 11](#_Toc61884728)

[4.1 Code Mapping 11](#_Toc61884729)

[4.2 Extended code lists 11](#_Toc61884730)

[4.3 Identifiers 11](#_Toc61884731)

[4.4 Namespaces 12](#_Toc61884732)

[4.5 Documentation discovery 12](#_Toc61884733)

[4.6 Data modelling 12](#_Toc61884734)

[4.7 Linking properties to code lists 13](#_Toc61884735)

[5 Vocabulary and Context Architecture 14](#_Toc61884736)

[5.1 @Context files for Semantic Subsets 14](#_Toc61884737)

[5.2 @Context files for version management 15](#_Toc61884738)

[5.3 File naming 15](#_Toc61884739)

[6 Conformance Criteria 16](#_Toc61884740)

[6.1 NDR Conformance 16](#_Toc61884741)

[6.2 Semantic Conformance 16](#_Toc61884742)

[6.3 Interface Conformance 16](#_Toc61884743)

[7 Reference Implementation 17](#_Toc61884744)

[8 JSON-LD Explainer 18](#_Toc61884745)

[8.1 Linked Data 18](#_Toc61884746)

[8.2 JSON-LD 18](#_Toc61884747)

[8.3 Non-Breaking Retro Fitting 19](#_Toc61884748)

[9 RDF Schema (RDFS) explainer 21](#_Toc61884749)

[9.1 Classes 21](#_Toc61884750)

[9.2 Properties 21](#_Toc61884751)

[9.3 Inferencing 22](#_Toc61884752)

[9.4 RDFS for existing code-lists 22](#_Toc61884753)

**Document History**

|  |  |  |
| --- | --- | --- |
| **Phase** | ***Status*** | ***Date Last Modified*** |
|  | First draft | 10 Sept 2020 |
|  | Final draft | 15 Sept 2020 |
|  | First revision | 10 Jan 2021 |
|  |  |  |

Figure 1 – Document History

### Change Log

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

| **Date of Change** | **Version** | **Paragraph Changed** | **Summary of Changes** |
| --- | --- | --- | --- |
| 10 Jan 2021 | 1.0 | 3.2 | Added data type naming simplification |
| 10 Jan 2021 | 1.0 | 3.3 | Added de-duplication of ASBIE as well as BBIE |
| 10 Jan 2021 | 1.0 | 4.7 | Added rules for linking properties to code lists where relevant |
| 10 Jan 2021 | 1.0 | 5 | Added vocabulary architecture section including   * Semantic subsets * Versioning * Fine / path naming |
| 10 Jan 2021 | 1.0 | 7 | Added link to unece.org reference implementation |

Figure 2 - Document Change Log

### Glossary

| **Term** | **Definition** |
| --- | --- |
| 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 – eg “consignee” as a relationship between “consignment” and “party” |
| BBIE | Basic Business Information Entity – a term from CCTS that describes a property of a class such as party.name |
| 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” |
| IRI | Internationalised Resource Identifiers – a version of the IETF URI specification that support international character sets. |
| 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 (eg RDM) to another (eg JSON-LD) |
| 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 |
| RDFS | RDF Schema – an XML schema for RDF documents. |
| RDM | Reference Data Model- a UN/CEFACT semantic output. |
| SHACL | A W3C technical specification – the SHApes Constraint Language – used to validate the structure of published semantic graphs (vocabularies.) |
| 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. |

Figure 2 - Glossary

# Introduction

This JSON-LD NDR guidance document forms part of a suite of documents that aim to support modern web developers to make use of UN/CEFACT semantics. It is the foundation specification that provides a mechanism for consistent publishing of Reference Data Models (RDM) and Code Lists as machine readable vocabularies.

The UN/CEFACT vocabulary is currently published as a CSV file (the reference data models) and variously as CSV, XML, PDF or HTML (the code lists). The core purpose of this specification is to define the naming and design rules for consistent publishing of both the reference data models and code lists as JSON-LD vocabularies. This is the foundation specification that makes UN/CEFACT semantics accessible and consumable for web developers.

A picture containing sitting, monitor, control, silver

Description automatically generated

Related specifications provide guidance on how to leverage the vocabularies to design high quality APIs (Application programming Interfaces).

# Requirements

This specification will have achieved its purpose when UN semantics are published and consumable in a similar way to other well-established vocabularies such as schema.org.

Within this primary goal, there are several more detailed requirements

1. unambiguous. The NDR must define unambiguous rules for publishing UN/CEFACT constructs such as ABIEs, ASBIES, BBIEs, etc. as JSON-LD vocabulary constructs.
2. governed. The UN/CEFACT RDMs and code lists are updated on a regular basis (roughly once per 6 months). The JSON-LD publishing process should allow updates to the vocabulary (not a new duplicated vocabulary) at each version increment.
3. developer friendly. The published output must be read-able and consumable by any developer that is familiar with JSON-LD and should not require any understanding of UN/CEFACT library management terms and processes (eg they should not need to know what an ABIE is). Schema.org provides the most widely used JSON-LD vocabulary in use today and so is a good guide for what the published UN/CEFACT output should look like.
4. de-duplicated. In JSON-LD a “property” such as “consignment.consignor” is a primary entity and has attributes like “domain” (i.e. which classes may include this property) and “range” (ie what is the value domain of this property). In the UN/CEFACT RDMs the “class” is the primary entity and properties can only belong to a class. Furthermore it is common for the RDM to define several version of the same class intended for use in different contexts (eg “consignment” and “referenced.consignment”). There is usually significant overlap between the properties of these classes. This means that the same semantic vocabulary item occurs multiple times. The JSON-LD vocabulary must de-duplicate without losing the usage context.
5. Codes included. The published vocabulary should include the full enumeration of all UN/CEFACT code lists including those from Recommendations and from the EDIFACT libraries. Code lists should be published independently but should be referenced by the JSON-LD properties (aka BBIEs) for which the code represents the allowed value range.
6. Discoverable. The full vocabulary should be published and maintained at a permanent discoverable URL in the unece.un.org domain.

# Reference Data Model Representation

This chapter describes guidance rules for publishing reference data models such as the UN/CEFACT buy-ship-pay as a JSON-LD vocabulary.

## Entity mapping

The JSON-LD vocabulary maps to UN/CEFACT Buy-Ship-Pay Reference Data Model entities according to the rules listed below.

* ABIEs are grouped by Object Class Term as RDFS Classes
* BBIEs are grouped by Property Term Qualifier(s) + Property Term + Datatype Qualifier(s) + Representation Term as RDFS Properties
* ASBIEs are grouped by Property Term Qualifier(s) + Property Term + Associated Object Class Term Qualifier(s) + Associated Object Class as RDFS Properties

For example:

A JSON-LD element with type rdfs:class - representing an ABIE

|  |
| --- |
| {  @id: "unece:Certificate",  @type: "rdfs:Class",  rdfs:comment: "A collection of data for a piece of written, printed or electronic matter that provides information or evidence about the product."  } |

A JSON-LD element with type rdfs:Property and rdfs:range as a simple type - representing a BBIE

|  |
| --- |
| {  @id: "unece:countryNameText",  @type: "rdfs:Property",  rdfs:range: "xsd:string",  rdfs:domain: [  "unece:Address",  "unece:Location"],  rdfs:comment: "A name, expressed as text, of the country for this trade address.",  } |

A JSON-LD element with type rdfs:Property representing and rdfs:range as another class – representing an ASBIE

|  |
| --- |
| {  @id: "unece:relatedSupplyChainConsignment",  @type: "rdfs:Property",  rdfs:range: "unece:Consignment",  rdfs:domain: "unece:TradeDelivery",  rdfs:comment: "A consignment, at line level, related to this line trade delivery."  } |

## Data Types

UN/CEFACT CDTs will map to XSD simple types. The simple types are referenced by the JSON-LD property term as rdfs:property with an rdfs:range with value drawn from the set of XSD simple types such as “xsd:string”. For example, refer to the “CountryNameText” example in section 3.2

UN/CEFACT QDTs will map to an rdfs:class that describes the QDT. This class is referenced as the rdfs:range of all properties with allowed values controlled by the QDT. For example, refer to the "rec20:kilogram\_per\_square\_meter" code in section 4.2.

Additionally, in order to improve readability of the JSON-LD vocabulary and improve alignment with other vocabularies;

* Properties with CDT type “Text” will have the suffix “Text” removed from the JSON-LD property name – so, for example, “carrierProvidedInformationText” becomes “carrierProvidedInformation”
* Properties with CDT type “Identifier” will have the suffix renamed to “Id” – so, for example, “freightForwarderAssignedIdentifier” becomes freightForwarderAssignedId

## De-duplication

The above grouping rules may lead to deduplication of several CEFACT ABIEs into single class or property. For example, both *SupplyChain\_Consignment* and *Referenced\_SupplyChain\_Consignment* BIEs get merged into one *Consignment* class.

Such deduplication is necessary to make the RDFS modelling guidelines unambiguous.

When two such classes are merged then any duplicated associations from the same source class to the merged target classes should also be merged. For example, when “Referenced\_SupplyChain\_ConsignmentItem” and “SupplyChain\_ConsignmentItem” are merged to become just “SupplyChain\_ConsignmentItem” then the ASBIEs “Included\_Rererenced\_SupplyChain\_ConsignmentItem” and “Included\_ SupplyChain\_ConsignmentItem” should be merged to become just “Included\_ SupplyChain\_ConsignmentItem”.

## Primary identifier mapping

Some UN/CEFACT BIEs have explicit primary identifier property, for example Referenced\_SupplyChain\_Consignment.Identification.Identifier. These properties are omitted in the RDF vocabulary, as RDF data model makes the primary identifier an inherent attribute for each entity in the graph.

## UN/CEFACT metadata

We provide and publish the machine-readable RDF representation of The UN/CEFACT Buy-Ship-Pay RDM Business Information Elements, preserving their types, inheritance hierarchy and metadata. All rdfs classes and properties in unece vocabulary are linked with corresponding BIEs by its identifier. This link can be used to implement a software which automatically maps UN/CEFACT RDM messages to RDF format. So that interoperability between existing systems which use UN/CEFACT RDM and new Linked Data based systems is preserved.

The example rdfs property from the unece vocabulary, with linked UN/CEFACT RDM BIEs:

|  |
| --- |
| {  "@id": "unece:consignorTradeParty",  "@type": "rdfs:Property",  "rdfs:domain": "unece:Consignment",  "rdfs:range": "unece:Party",  "unece:cefactElementMetadata": [  {  "@id": "cefact:Referenced\_SupplyChain\_Consignment.Consignor.Trade\_Party",  "@type": "unece:AssociationBIE",   "unece:cefactUNId": "cefact:UN01011054",  "unece:cefactBieDomainClass": "cefact:Referenced\_SupplyChain\_Consignment.Details",  "unece:cefactBusinessProcess": "Buy-Ship-Pay"  },  {  "@id": "cefact:SupplyChain\_Consignment.Consignor.Trade\_Party",  "@type": "unece:AssociationBIE",   "unece:cefactUNId": "cefact:UN01004212",  "unece:cefactBieDomainClass": "cefact:SupplyChain\_Consignment.Details",  "unece:cefactBusinessProcess": "Buy-Ship-Pay"  },  ] } |

## Business domain granularity

The vocabulary terms are annotated with the logical business domain which this term belongs to:

|  |
| --- |
| {  "@id": "unece:consignorTradeParty",  "@type": "rdfs:Property",  "unece:businessDomain": "Trade" } |

The business domain is drawn from the governed list of domains such as “trade”, “transport”, “regulatory”, and “core” that are managed by UN/CEFACT architecture processes.

## Versioning

The vocabulary files are updated at whatever frequency the owning domain chooses – usually every 6 months. Each BIE is annotated with the date when it was created, current active status, and the date of deprecation.

|  |
| --- |
| {  "@id": "SupplyChain\_Consignment.Consignor.Trade\_Party",  "@type": "unece:AssociationBIE",   "@unece:cefactUNId": "cefact:UN01004212",  "unece:currentStatus":"deprecated",  "unece:createdDate": "01-04-2017",  "unece:deprecatedDate": "21-03-2020" } |

Each rdfs class and property in the vocabulary is annotated the same way. The rdfs class or property can only be deprecated when all the RDM BIEs it is linked to are deprecated.

Every time when the vocabulary is updated from the new version of BSP RDM, the new JSON-LD context file for this vocabulary is created, and published at the new permanent URL, e.g https://unece.org/vocab/2020.09/context.json

# Code List Representation

Code-lists are as important to semantic interoperability as the terms in the reference data model vocabularies. Therefore, they must form part of the JSON-LD publishing guideline.

However, code lists have a different structure to the RDMs and so have a separate set of publishing rules. Some code-lists have fairly flat and simple organization, for example iso-3166 country codes. But others may have quite complex hierarchical structure with additional metadata, for example the UN/LOCODEs.

## Code Mapping

In this section we describe the recommended format for publishing code-lists using rdfs and JSON-LD data model. The vocabulary definitions are represented as [flattened JSON-LD](https://www.w3.org/TR/json-ld/#flattened-document-form) graph:

|  |
| --- |
| {  "@context": "https://unece.org/vocab/2020.09/context.json",  "@graph": [  { "@id": "iso:AU", “@type”:”unece:CountryCodes”, "rdfs:label": "Australia" },  { "@id": "iso:US", “@type”:”unece:CountryCodes”, "rdfs:label": "United States of America" },  **...**  ] } |

RDF graph data model and JSON-LD representation may be the best format for machine-readable vocabularies available today. Some prominent features are:

* Standardised way to cross-reference, reuse and extend terms from multiple separately governed vocabularies
* Support for internationalized strings
* Supports hierarchical model of classes and properties
* Consistent library of simple data types like bool, int, date and time
* JSON-LD is designed to be easily interpreted by human developer, compared to older formats like xml

## Extended code lists

Some code lists contain multiple attributes per code. For example the codes of UNECE Rc 20 include properties like “conversionFactor” and “unitSymbol”. These are represented in JSON-LD as follows.

|  |
| --- |
| {  "@id": "rec20:kilogram\_per\_square\_meter",  "@type": "unece:NormativeUnit",  "rdf:label": "kilogram per square metre",  "rdf:comment": "Unit of surface density, areic mass",  "unece:uneceRec20Code": "28",  "unece:conversionFactor": "kg/m²",  "unece:unitSymbol": "kg/m²"  } |

## Identifiers

Business application data usually reference entities defined in the code-list vocabulary by its identifier. For example, “AU” is an identifier of Australia, defined by iso-3166.

While arbitrary string like “AU” may be good enough identifier in many scenarios, [best practices for data on the web](https://www.w3.org/TR/dwbp/#DataIdentifiers) is to use http URLs as primary identifiers. The advantages of http URLs are name-spacing and discovery, briefly highlighted below.

## Namespaces

Sometimes several concurrent code-lists exist, which describe similar concepts, for ex. [Vehicle Plate Country codes](https://en.wikipedia.org/wiki/International_vehicle_registration_code) vs iso-3166 country codes. Quite often business applications data have to use a mix of multiple code-lists, rendering the used identifiers ambiguous.

To resolve the identifiers ambiguity, we recommend using http URLs based on the domain name which is under control of the authoritative group which maintains the code-list vocabualry. For example in place of UNECE rec.21 code “1A”, the http URL “https://www.unece.org/uncefact/rec21#1A” can be used.

For human convenience, most RDF syntaxes support URL shortening. For example, the JSON-LD representation can use default vocabualry or namespace prefix defined in the context:

Example: default vocabulary makes “1A” to expand to the full URL “https://www.unece.org/uncefact/rec21#1A”

|  |
| --- |
| {  "@context": {   "@vocab": "https://www.unece.org/uncefact/rec21#",  "typeCode": {"@type": "@vocab"}  },  "@id": "http://maersk.com/packages/171346",  "typeCode": "1A"  }  Example: prefix definition makes “rec21:1A” to expand to the full URL “https://www.unece.org/uncefact/rec21#1A”  {  "@context": {   "@vocab": "https://unece.org/vocab#",  "rec21": "https://www.unece.org/uncefact/rec21#",  "typeCode": {"@type": "@id"}  },  "@id": "http://maersk.com/packages/171346",  "typeCode": "rec21:1A"  } |

## Documentation discovery

It is recommended that referencing vocabulary term identifier URL in the web browser result (or redirect to) the page, where the human-readable definition of this term can be found.

## Data modelling

Entities in the code-list vocabulary might have associated metadata, such as human-readable definition, comments, symbolic representation, and links to other related entities. Each entity can be seen as a node in the RDF graph with assigned primary identifier (http URL), and other data-typed or identifier nodes linked to it. In the Subject-Predicate-Object RDF representation, the kg/m² measurement unit can be defined as:

|  |
| --- |
| <rec20:kilogram\_per\_square\_meter> <rdfs:comment> "Unit of surface density, areic mass" . <rec20:kilogram\_per\_square\_meter> <unece:unitSymbol> "kg/m²" . <rec20:kilogram\_per\_square\_meter> <unece:uneceRec20Code> "28" . |

Which corresponds to the JSON-LD

|  |
| --- |
| {  "@id": "rec20:kilogram\_per\_square\_meter",  "rdfs:comment": "Unit of surface density, areic mass",  "unece:uneceRec20Code": "28",  "unece:unitSymbol": "kg/m²" } |

Predicates used to associate entity with related metadata have their own primary identifier (http URL). In the example above the context is omitted, but implied that *rdfs:comment* and *unece:uneceRec20Code* are abbreviated http URL identifiers of predicates (properties), which are defined in the rdfs and unece vocabularies (see rdfs properties definition below).

## Linking properties to code lists

In some cases, a JSON-LD property value domain is controlled be a code list and it is important that the vocabulary should reference the specific code list rather than assigning a more generic “code” type. This is achieved by specifying the @id of the code list class in the schema:rangeIncludes attribute of the property – as shown in bold in the example below.

|  |
| --- |
| {  @id: "unece:transportServicePriorityCode",  @type: "rdf:Property",  **schema:rangeIncludes:**  **{**  **@id: "unece:UNECECL4219Code"**  **},**  schema:domainIncludes:  {  @id: "unece:Service"  },  rdfs:comment: "The code specifying the priority of this transport service.",  rdfs:label: "transportServicePriorityCode",  unece:cefactElementMetadata:  [  {  @id: "cefact:Transport\_Service.Priority.Code",  @type: "unece:BasicBIE",  unece:TDED: "4219",  unece:cefactUNId: "cefact:UN01004844",  unece:cefactBieDomainClass: "cefact:Transport\_Service.Details",  unece:cefactBusinessProcess: "In All Contexts",  rdfs:comment: "The code specifying the priority of this transport service."  }]} |

# Vocabulary and Context Architecture

The high level architecture of the UN/CEFACT JSON-LD vocabulary is shown below.

Diagram, timeline

Description automatically generated

The core vocabulary files are managed by the UN/CEFACT business domain that owns them and are published as a set of non-overlapping vocabulary files. New terms and/or code values can be added at any time by the owning authority.

JSON-LD context files are used to manage semantic subsets and specific versions as described below.

## @Context files for Semantic Subsets

The UN/CEFACT information modelling approach has the idea of semantic subsets of large models that are relevant for a specific context. For example

* The Buy-Ship-Pay (BSP) reference data model (RDM) is a subset of the full Core Component Library.
* The Multi-Modal Transport (MMT) RDM is a subset of the BSP.
* Further subsets can be created for specific scenarios like “all properties that are of interest in road transport”

The idea of context specific library subsets are easily represented in JSON-LD using the @context reference. A simple context file is shown below. This example contains two elements (“Address” and “Consignment”)

|  |
| --- |
| **@context**: {  **@vocab**: "unece",  **unece**: "https://unece.org/vocab/bsp/#",  **Address**: {  **@id**: "unece:Address"  },  **Consignment**: {  **@id**: "unece:Consignment"  }, |

Also, with JSON-LD, a context file can reference several vocabularies. So it is possible to compose a context file as a subset of multiple vocabularies. For example, perhaps there is a cross-border certificate context that draws from both an agriculture vocabulary and a supply chain vocabulary

|  |
| --- |
| **@context**: {  **@vocab**: "unece",  **unece**: " [https://unece.org/vocab/bsp/#](https://unece.org/vocab/bsp/)",  **agricultiure**: " https://unece.org/vocab/agriculture/#,  **PhytosanitaryCertificate**: {  **@id**: "agriculture:PhytoSanitaryCertificate"  },  **Consignment**: {  **@id**: "unece:Consignment"  }, |

## @Context files for version management

Vocabulary properties and classes have identifying and resolvable URIs – for example <https://service.unece.org/trade/uncefact/vocabulary/unece/#countryName> represents a country name and is used in the address class. When a vocabulary is updated with a new property, this does not change the meaning or permanent URI of existing properties. Therefore, it is not feasible to publish a new vocabulary for each change. Instead, existing vocabulary files need to be updated with additions and deprecations.

At the same time, implementers need confidence in the stability of a vocabulary so that they can say “I support BSP-19b” for example. This is achieved by publishing a JSON-LD @context file for each version. The context file contains a set of pointers to vocabulary items. So, for example, a new term can be added to BSP in 2020 but the BSP-19b @context file will not include it and so remains a stable reference for a point-in-time view of the vocuabulary.

## File naming

Vocabulary files are named and published as follows:

https://{organisationDomain}/vocabulary/{businessDomain}.jsonld

Examples

* https://service.unece.org/trade/uncefact/vocabulary/transport.jsonld
* https://service.unece.org/trade/uncefact/vocabulary/rec20.jsonld

Context files are named & published as follows:

https://{organisationDomain}/vocabulary/{businessDomain}/{contextName}-context.jsonld

Examples

* https://service.unece.org/trade/uncefact/vocabulary/supplyChain/bsp19a-context.json-ld
* https://service.unece.org/trade/uncefact/vocabulary/locodes-au/20b-context.jsonld

Please refer to the API Town plan governance document for further details on vocabulary naming.

# Conformance Criteria

Any standard specification should indicate how to achieve conformance so that implementers can make valid conformance claims and so that consumers can trust the quality of implementations.

There are three types of conformance that are relevant to the RDM2API framework:

## NDR Conformance

NDR conformance tests whether the published JSON-LD vocabulary structure conforms to the requirements in this document. For example, it would test whether a class or property definition has the CefactElementMetadata Object with the right attributes. This is primarily a quality control on the published UN/CEFACT output and not generally of concern to end users.

A typical way to test that conformance would be via a SHACL shapes graph. It is expected that, if this guidance note should continue to become a full UN/CEFACT technical specification then a corresponding SHACL graph to validate published output will be developed.

## Semantic Conformance

Semantic conformance tests whether a particular JSON-LD linked data graph meets some rules specified by a consumer. This is very similar to the way google validates linked data structures of a recipe published to the web using <https://search.google.com/test/rich-results>. So, for example, a community might define mandatory and recommended requirements for the data about a consignment. This kind of conformance is also tested using a SHACL graph.

## Interface Conformance

Interface conformance tests whether a specific simple JSON document conforms to an open API specification. Since JSON-LD is also just JSON, a document can also be validated against a schema. UN/CEFACT will publish standard openAPI specifications for common use cases such as voyage schedules, bills of lading, certificates of origin, and so on. These interfaces will be defined using terms from standard UN/CEFACT JSON-LD vocabularies.

# Reference Implementation

In order to ensure that this guidance material is accurate and workable, an actual implementation has been developed alongside the document development. The implementation is browsable at

**https://service.unece.org/trade/uncefact/vocabulary/unece/**

# JSON-LD Explainer

## Linked Data

Machines build “knowledge graphs” by linking data together. This is done by use of RDF (Resource Description Framework). RDF specifies expressing data as so-called triples, defining subject-predicate-object. A super simple example of a knowledge graph might be:

* The author’s name is Nis.
* Nis lives in Denmark.

From this, a machine would be able to build this kind of simple knowledge graph: Author - name is - Nis - lives in - Denmark. And by parsing this graph, determine for example that “the author lives in Denmark”.

All subjects, predicates and (some) objects are identified on the web by a IRI (Internationalised Resource Identifier). An IRI identifies *the thing*, as opposed to a URL which locates a page *describing the thing*. While this is important to machines, there can be a bit of a clash here as to how humans best read data. For example <https://schema.org/author> <https://schema.org/givenName> Nis. is hardly as readable as ”Presenter’s name is Nis”.

## JSON-LD

Luckily, RDF comes in many flavors. The one which we will focus on here is JSON-LD, which among its advantages is that it is useful for both humans and machines. Also, JSON-LD is based on JSON, which practically is the grammar of any modern API.

JSON-LD works by injecting a “context” and some other linked data aspects into a normal JSON. All injections are prefixed with an “@” indicating a JSON-LD keyword.

Let’s consider an example:

|  |
| --- |
| {  "@context": "https://unece.org/specs/unece-transport/develop/context.jsonld",  "consignment": {  "bookingNumber": "123456789",  "@id": "https://www.maersk.com/tracking/123456789",  "includedConsignmentItem": [  {  "consignmentItem": {  "information": "Mangos and bananas",  "grossWeight": {  "Value": "12000", "Unit": "Kgs"  }  }  }  ],  "utilizedTransportEquipment": [  {  "transportEquipment": {  "identification": "MSKU0134962",  "@id": "https://app.bic-boxtech.org/containers?search=MSKU0134962"  }  }  ]  } } |

Most of this is just a basic json: a consignment with some consignmentItems and a transportEquipment. All expressed in somewhat nice CEFACT lingo which is useful for humans, but just meaningless strings to a computer. The JSON-LD parts @context and @id changes that.

The @id tags inject IRIs to properly identify the consignment and transportEquipment, respectively referencing appropriate APIs from the carrier and BIC.

The @context links to a jsonld file, defining the semantic meaning of each element of the JSON (note that the context does not have to be externalized to a referenced file like this, but can also just be included directly within the json data file). Here’s what https://unece.org/specs/unece-transport/develop/context.jsonld might look like:

|  |
| --- |
| {  "@context": {  "consignment": {  "@id": "https://unece.org/specs/unece-transport/develop/vocab/Consignment",  "@type": "@id"  },  "includedConsignmentItem": "https://unece.org/specs/unece-transport/develop/vocab/Consignment#ConsignmentItem",  "consignmentItem": "https://unece.org/specs/unece-transport/develop/vocab/ConsignmentItem",  "utilizedTransportEquipment": "https://unece.org/specs/unece-transport/develop/vocab/Consignment#utilizedTransportEquipment",  "transportEquipment": {  "@id": "https://unece.org/specs/unece-transport/develop/vocab/TransportEquipment",  "@type": "@id"  }  } } |

Here, the @context adds mapping from the human terms in the JSON to IRIs formally defining the semantics used. For example, https://unece.org/specs/unece-transport/develop/vocab/Consignment is the IRI for Consignment.

From this, a computer is able to build a model like this (here serialized as N-Quads):

|  |
| --- |
| <https://www.maersk.com/tracking/123456789> <https://unece.org/specs/unece-transport/develop/vocab/Consignment#ConsignmentItem> \_:b1 . <https://www.maersk.com/tracking/123456789> <https://unece.org/specs/unece-transport/develop/vocab/Consignment#utilizedTransportEquipment> \_:b3 . \_:b0 <https://unece.org/specs/unece-transport/develop/vocab/Consignment> <https://www.maersk.com/tracking/123456789> . \_:b1 <https://unece.org/specs/unece-transport/develop/vocab/ConsignmentItem> \_:b2 . \_:b3 <https://unece.org/specs/unece-transport/develop/vocab/TransportEquipment> <https://app.bic-boxtech.org/containers?search=MSKU0134962>. |

## Non-Breaking Retro Fitting

A clever aspect of JSON-LD is that it can be retrofitted “on top” of legacy JSONs. Adding the JSON-LD (@-prefixed) tags will not break your APIs.

For example, the following legacy JSON (which is much less aligned to CEFACT) will continue working, but generate the exact same machine model. Note that the @context in this example is embedded into the JSON itself:

|  |
| --- |
| {  "@context": {  "shipment": {  "@id": "https://unece.org/specs/unece-transport/develop/vocab/Consignment",  "@type": "@id"  },  "goods": "https://unece.org/specs/unece-transport/develop/vocab/Consignment#ConsignmentItem",  "goodsItem": "https://unece.org/specs/unece-transport/develop/vocab/ConsignmentItem",  "containers": "https://unece.org/specs/unece-transport/develop/vocab/Consignment#utilizedTransportEquipment",  "container": {  "@id": "https://unece.org/specs/unece-transport/develop/vocab/TransportEquipment",  "@type": "@id"  }  },  "shipment": {  "bookingNumber": "123456789",  "@id": "https://www.maersk.com/tracking/123456789",  "goods": [{  "goodsItem": {  "information": "Mangos and bananas",  "grossWeight": {"Value": "12000", "Unit": "Kgs"}  }  }],  "containers": [{  "container": {  "boxNb": "MSKU0134962",  "@id": "https://app.bic-boxtech.org/containers?search=MSKU0134962"  }  }]  } } |

# RDF Schema (RDFS) explainer

[RDF Schema](https://www.w3.org/TR/rdf-schema/) provides the mechanisms for describing groups of related resources (entities) and the relationships between these resources. The RDF Schema class and property system is similar to the type systems of object-oriented programming languages, and modelling languages like UML.

While abstract linked graph view on the data may be useful for simple cases, the RDF Schema provides familiar and powerful semantics of Classes and Properties on top of it.

## Classes

Continuing from the example of UNECE rec.20 codes for measurement units, we can divide entities into classes NormativeUnit, NormativeEquivalentUnit, InformativeUnit, which are all subclasses of MeasurementUnit. To achieve that, we should first define a class, and then we can assign that class to the entity (instance of that class). On the RDF level, classes just like instances have unique identifier an can be seen as nodes in the graph.

Example: Base class and its specific subclasses definition

|  |
| --- |
| <unece:MeasurementUnit> <rdf:type> <rdfs:Class> .  <unece:NormativeUnit> <rdf:type> <rdfs:Class> . <unece:NormativeUnit> <rdfs:subClassOf> <unece:MeasurementUnit> .  <unece:NormativeEquivalentUnit> <rdf:type> <rdfs:Class> . <unece:NormativeEquivalentUnit> <rdfs:subClassOf> <unece:MeasurementUnit> .  <unece:InformativeUnit> <rdf:type> <rdfs:Class> . <unece:InformativeUnit> <rdfs:subClassOf> <unece:MeasurementUnit> . |

Now all measurement units can be declared being an instance of appropriate class:

|  |
| --- |
| <rec20:kilogram\_per\_square\_meter> <rdf:type> <unece:NormativeUnit> . <rec20:fahrenheit> <rdf:type> <unece:NormativeEquivalentUnit> . |

## Properties

On the RDF graph level, predicates (properties) just like classes or instances have unique identifier an can be seen as nodes in the graph. Unlike classes or class instances, the property identifiers can also appear in the predicate position (link between the nodes), specifying the semantic of the relationship between subject and object. RDFS vocabulary allows to define a node as a property, and express restrictions on the valid types of the subject and object which is allowed to be linked by this property (domain and range).

Example: define a property and its domain and range

|  |
| --- |
| <unece:unitSymbol> <rdf:type> <rdfs:Property> . <unece:unitSymbol> <rdfs:domain> <unece:MeasurementUnit> . <unece:unitSymbol> <rdfs:range> <xsd:string> . |

Now the property can be used to associate measurement unit with its symbol

|  |
| --- |
| <rec20:kilogram\_per\_square\_meter> <unece:unitSymbol> "kg/m²" . |

## Inferencing

The advanced data consumers can apply RDFS inferencing engine to enrich the input graph data with the additional links (triples), which was omitted, but implied by the rdfs class hierarchy and property domain provided in the vocabulary.

For example, the kilogram\_per\_square\_meter was defined as instance of NormativeUnit:

|  |
| --- |
| <rec20:kilogram\_per\_square\_meter> <rdf:type> <unece:NormativeUnit> . |

And after applying RDFS inferencing, the new association will be added, saying that kilogram\_per\_square\_meter is also an instance of the base MeasurementUnit class:

|  |
| --- |
| <rec20:kilogram\_per\_square\_meter> <rdf:type> <unece:MeasurementUnit> . |

While RDFS inferencing is powerful, it is also computationally complex and hard to implement. It is not recommended to rely on the data consumer inferencing capability in the published data and vocabularies. It is up for particular use case to decide on, but generally it is safer to explicitly declare all types on the instances and use more generic properties instead of more specific sub-properties.

## RDFS for existing code-lists

Some existing code-lists combine multiple entity attributes to be “flattened” into a list of unique identifiers. For example, UNECE Rec.21 for package types assigns a code “BO” to “Bottle, non-protected, cylindrical”, and “XH” to “Bag, textile, water resistant”. The brute-force way to express this code-list in machine-readable way would be 1) assign the full http URL to each code, 2) associate it with a human-readable description and 3) publish it as flattened graph JSON-LD

Example: simplest JSON-LD representation of UNECE Rec.21

|  |
| --- |
| {  "@context": {  "rec21": "https://unece.org/code-lists/rec21#",  "rdfs": "http://www.w3.org/2000/01/rdf-schema#"  },  "@graph": [  {  "@id": "rec21:1A",  "rdfs:comment": "Drum, steel",  },   {  "@id": "rec21:1B",  "rdfs:comment": "Drum, aluminium",  },  **...**   ] } |

While the example given above is valid and fulfills the requirement of making the code-list machine-readable in many use-cases, it can be improved. Proper use of RDFS annotations can make the code-list vocabulary significantly more convenient to maintain, comprehend and implement in the business logic.

The entities in the rec.21 vocabulary can be quite naturally interpreted as types of package, so they can be declared to be instances of *rdfs:Class*. Also, the primary identifier can be made more human-friendly:

|  |
| --- |
| {  "@id": "rec21:Drum\_steel",  "@type": "rdfs:Class",  "rdf:value": "1A" } |

Now the business data producer can assign appropriate rec.21 package class to the subject of interest:

|  |
| --- |
| {  "@id": "http://maersk.com/packages/b646-629",  "@type": "rec21:Drum\_steel" } |

Many entities in the rec.21 vocabulary could be seen as subclasses of the generic base class, for example all package types listed below can be made subclasses of generic *Pallet* base class:

Pallet, CHEP 100 cm x 120 cm  
Pallet, AS 4068-1993  
Pallet, ISO T11

The appropriate class hierarchy can help maintainers to organize and visualize the vocabulary and allow business logic applications to choose the generalization level they need to operate on.

Some entities in the rec.21 vocabulary mix class-level abstraction with properties, such as water resistance or physical dimensions. It would be more natural to define the properties which business data could use to express such attributes:

|  |
| --- |
| {  "@id": "rec21:width",  "@type": "rdfs:Property",  "rdfs:domain": "rec21:BasePackage",  "rdfs:range": "xsd:decimal",  "rdfs:comment": "physical width of the package, in millimeters" } |

In some cases, part of the vocabulary such as base classes and properties could be extracted to form the stable core vocabulary, while keeping other more specific and volatile subclasses and instances to be governed and published separately. Such distinction might be beneficial for maintaining long-term interoperability between code-list users.