edi3 JSON-LD NDR 1.0 Specification

## Status

Raw

## Glossary

|  |  |
| --- | --- |
| Phrase | Definition |
|  |  |

This service depends on - TBA.

The TBA specification depends on this document. Note, TBA.

## Licence

All material published on edi3.org including all parts of this specification are the intellectual property of the UN as per the [UN/CEFACT IPR Policy](https://www.unece.org/fileadmin/DAM/cefact/cf_plenary/plenary12/ECE_TRADE_C_CEFACT_2010_20_Rev2E_UpdatedIPRpolicy.pdf).

This Specification is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. See http://www.gnu.org/licenses.

## Change Process

This document is governed by the [2/COSS](http://rfc.unprotocols.org/spec:2/COSS/) (COSS).

## Language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

# Introduction

The JSON-LD project aims to let not just humans, but also machines understand the semantics of CEFACT.

## 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 (Uniform Resource Identifier). An IRI identifies *the thing*, as opposed to a URL which locates a page describing the *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://edi3.org/specs/edi3-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://edi3.org/specs/edi3-transport/develop/context.jsonld might look like:

{  
 "@context": {  
 "consignment": {  
 "@id": "https://edi3.org/specs/edi3-transport/develop/vocab/Consignment",  
 "@type": "@id"  
 },  
 "includedConsignmentItem": "https://edi3.org/specs/edi3-transport/develop/vocab/Consignment#ConsignmentItem",  
 "consignmentItem": "https://edi3.org/specs/edi3-transport/develop/vocab/ConsignmentItem",  
 "utilizedTransportEquipment": "https://edi3.org/specs/edi3-transport/develop/vocab/Consignment#utilizedTransportEquipment",  
 "transportEquipment": {  
 "@id": "https://edi3.org/specs/edi3-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://edi3.org/specs/edi3-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://edi3.org/specs/edi3-transport/develop/vocab/Consignment#ConsignmentItem> \_:b1 .  
<https://www.maersk.com/tracking/123456789> <https://edi3.org/specs/edi3-transport/develop/vocab/Consignment#utilizedTransportEquipment> \_:b3 .  
\_:b0 <https://edi3.org/specs/edi3-transport/develop/vocab/Consignment> <https://www.maersk.com/tracking/123456789> .  
\_:b1 <https://edi3.org/specs/edi3-transport/develop/vocab/ConsignmentItem> \_:b2 .  
\_:b3 <https://edi3.org/specs/edi3-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://edi3.org/specs/edi3-transport/develop/vocab/Consignment",  
 "@type": "@id"  
 },  
 "goods": "https://edi3.org/specs/edi3-transport/develop/vocab/Consignment#ConsignmentItem",  
 "goodsItem": "https://edi3.org/specs/edi3-transport/develop/vocab/ConsignmentItem",  
 "containers": "https://edi3.org/specs/edi3-transport/develop/vocab/Consignment#utilizedTransportEquipment",  
 "container": {  
 "@id": "https://edi3.org/specs/edi3-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"  
 }  
 }]  
 }  
}

# Requirements

This specification is part of a suite of documents that collectively provids the neccessary tools and methods for data modellers to produce high quality API designs based on UN/CEFACT semantics.

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 piurpose 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 vocuabularies. This is the foundation specification that makes UN/CEFACT semantics accessible and consumable for web developers. This specification will have achieved it’s 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 vocuabulary (not a new duplicated vocabulary) at each version increment.
3. developer freindly. The published output must be redable and consumable by any developer that is familiar with JSON-LD and should no 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-duplication. In JSON-LD a “property” such as “consignment.consignor”is a primary entity and has attributes like “domain” (ie which classes may include this proiperty) 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. what else?

# Naming & Design Rules

## RDM mapping

Current version of vocabualry was automatically generated from the CEFACT Buy-Ship-Pay Reference Data Model xls file, following 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

### De-duplication

The above grouping rules may lead to deduplication of several CEFACT BIEs 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 to be unambiguous.

### Primary identifier mapping

Some 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 entitiy in the graph.

## UN/CEFACT metadtaa

We provide and publish the machine-readable RDF representation of The CEFACT Buy-Ship-Pay RDM Business Information Elements, preserving their types, inheritance heirarchy and metadata. All rdfs classes and properties in edi3 vocabulary are linked with corresponding BIEs by its identifier. This link can be used to implement a software which automatically maps CEFACT RDM messages to RDF format. So that interoperability between existing systems which use CEFACT RDM and new Linked Data based systems is preserved.

The example rdfs property from the edi3 vocabulary, with linked CEFACT RDM BIEs:

{  
 "@id": "edi3:consignorTradeParty",  
 "@type": "rdfs:Property",  
 "rdfs:domain": "edi3:Consignment",  
 "rdfs:range": "edi3:Party",  
 "edi3:cefactElementMetadata": [  
 {  
 "@id": "cefact:Referenced\_SupplyChain\_Consignment.Consignor.Trade\_Party",  
 "@type": "edi3:AssociationBIE",   
 "edi3:cefactUNId": "cefact:UN01011054",  
 "edi3:cefactBieDomainClass": "cefact:Referenced\_SupplyChain\_Consignment.Details",  
 "edi3:cefactBusinessProcess": "Buy-Ship-Pay"  
 },  
 {  
 "@id": "cefact:SupplyChain\_Consignment.Consignor.Trade\_Party",  
 "@type": "edi3:AssociationBIE",   
 "edi3:cefactUNId": "cefact:UN01004212",  
 "edi3:cefactBieDomainClass": "cefact:SupplyChain\_Consignment.Details",  
 "edi3:cefactBusinessProcess": "Buy-Ship-Pay"  
 },  
 ]  
}

## Business domain granularity

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

{  
 "@id": "edi3:consignorTradeParty",  
 "@type": "rdfs:Property",  
 "edi3:businessDomain": "Trade"  
}

TODO: The formal process of assigning the business domain to the vocabulary terms is to be decided.

## Versioning

The vocabulary is updated every 6 months, following the maintainance cycle of the CEFACT BSP RDM. Each BIE is annotated with the date when it was created, current activestatus, and the date of deprecation.

{  
 "@id": "SupplyChain\_Consignment.Consignor.Trade\_Party",  
 "@type": "edi3:AssociationBIE",   
 "@edi3:cefactUNId": "cefact:UN01004212",  
 "edi3:currentStatus":"deprecated",  
 "edi3:createdDate": "01-04-2017",  
 "edi3:deprecatedDate": "21-03-2020"  
}

Each rdfs class and property in the vocabualry 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 vocabualry is created, and published at the new permanent url, e.g https://edi3.org/vocab/2020.09/context.json

TODO: the exact url for the context is to be decided.

## Code list representation

Domain-specific parts of data model may be goverened and published separately, some of these vocabularies are called “codelists”. Such vocabularies sometimes have fairly flat and simple organization, for ex. iso-3166 country codes. But others may have quite complex hierarchical structure with additional metadata, for ex. WCO Harmonized System nomenclature.

In this section we describe the recommended format for publishing codelists using rdf 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://edi3.org/vocab/2020.09/context.json",  
 "@graph": [  
 { "@id": "iso:AU", "rdfs:label": "Australia" },  
 { "@id": "iso:US", "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:

* Standartized 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

### Motivation

TODO: does it belong here?

To avoid interoperability disruptions, it is important for communicating systems to have consistent and up-to-date data model to operate on. Given that codelists are regularly updated, maintaining interoperability between several separately developed business applications is a challenging task.

Unfortunately maintainers often publish codelists in proprietary or machine-unfriendly formats like xls, pdf and html, which require tedious human processing to translate and implement in business logic. It would be beneficial to have an authoritative source of vocabulary definitions in a machine readable format to enable automated processing and allow existing systems to have always up-to-date and consistent view on the data they produce/consume.

TODO: we could discuss somewhere automated mechanisms to distribute updates of machine-readable vocabularies. Differences between pushapproaches, for ex. CDN vs infrastructure based on WEBSUB hubs?

### Identifiers

Business application data usually reference entities defined in the codelist 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 namespacing and discovery, briefly highlighted below.

#### Namespaces

Sometimes several concurrent codelists 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 codelists, 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 codelist 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://edi3.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 dereferencing 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 codelist 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 datatyped 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> <edi3:unitSymbol> "kg/m²" .  
<rec20:kilogram\_per\_square\_meter> <edi3:uneceRec20Code> "28" .

Which corresponds to the json-ld

{  
 "@id": "rec20:kilogram\_per\_square\_meter",  
 "rdfs:comment": "Unit of surface density, areic mass",  
 "edi3:uneceRec20Code": "28",  
 "edi3: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 *edi3:uneceRec20Code* are abbreviated http url identifiers of predicates (properties), which are defined in the rdfs and edi3 vocabularies (see rdfs properties definition below).

#### 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

<edi3:MeasurementUnit> <rdf:type> <rdfs:Class> .  
  
<edi3:NormativeUnit> <rdf:type> <rdfs:Class> .  
<edi3:NormativeUnit> <rdfs:subClassOf> <edi3:MeasurementUnit> .  
  
<edi3:NormativeEquivalentUnit> <rdf:type> <rdfs:Class> .  
<edi3:NormativeEquivalentUnit> <rdfs:subClassOf> <edi3:MeasurementUnit> .  
  
<edi3:InformativeUnit> <rdf:type> <rdfs:Class> .  
<edi3:InformativeUnit> <rdfs:subClassOf> <edi3:MeasurementUnit> .

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

<rec20:kilogram\_per\_square\_meter> <rdf:type> <edi3:NormativeUnit> .  
<rec20:fahrenheit> <rdf:type> <edi3: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

<edi3:unitSymbol> <rdf:type> <rdfs:Property> .  
<edi3:unitSymbol> <rdfs:domain> <edi3:MeasurementUnit> .  
<edi3:unitSymbol> <rdfs:range> <xsd:string> .

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

<rec20:kilogram\_per\_square\_meter> <edi3: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 heirarchy and property domainprovided in the vocabulary.

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

<rec20:kilogram\_per\_square\_meter> <rdf:type> <edi3: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> <edi3: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 typeson the instances, and use more generic properties instead of more specific subproperties.

#### RDFS for existing codelists

Some existing codelists 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 codelist 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/codelists/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 codelist machine-readable in many usecases, it can be improved. Proper use of RDFS annotations can make the codelist 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 heirarchy 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 exctracted to form the stable core vocabualry, 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 codelist users.

# Examples

These are some simple examples of how this could look for the three classes referenced in the above examples: \* <https://edi3.org/specs/edi3-transport/develop/vocab/Consignment> \* <https://edi3.org/specs/edi3-transport/develop/vocab/ConsignmentItem> \* <https://edi3.org/specs/edi3-transport/develop/vocab/TransportEquipment>