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

Stuff about ABIE/BBIE etc -> JSON-LD here

## 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/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?

### Examples of more complex hierarchical codelists

TODO

## @context granularity

Stuff about granularity of graph publishing here - ie one graph per serpately goverened thing in the source

## primary @id mapping

stuff about mapping entity ID to JSON-LD @id here

## de-duplication

stuff about de-duplication of properties here

## versioning

stuff about version updates here

## 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",  
 "rdfs: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"  
 },  
 ]  
}

# 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>