edi3 JSON-LD NDR 1.0 Specification

# 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"  
 }  
 }]  
 }  
}

# EDI3 JSON-LD Project Output

Common semantics are defined many places throughout the internet. A very relevant example is https://schema.org/ which is extensively used for common stuff like defining Person, Address, Name and many other such fundamental aspects. More specialized semantics typically require specialized governance. In the case of trade and transport, that governance is CEFACT. In order to enable the web developers of the world to utilize our semantics, part of the edi3 ambition is to expose our semantics in a referenceable way as a library of IRIs.

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>

This should obviously be generated from the SCRDM model, part of the output of the RDM2API methodology. Similarly, the governance of the various high level groupings (in these examples /edi3-transport) should follow the town plan approach.

## Status

Dratf.

## 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.

# Billing Process

## State Lifecycle

# Related Material