##### UN/CEFACT

United Nations

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

Methodology and Technology Programme Development Area Specifications Domain

OpenAPI 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 OpenAPI Naming and Design Rules technical specification defines an architecture and a set of rules necessary to specify, describe and implement APIs based on an OpenAPI specification to consistently express business information. It is based on the OpenAPI specification and the UN/CEFACT Core Components Technical Specification. This specification describes the requirements that UN/CEFACT compliant APIs should fulfil. It will be used by other organisations who are interested in maximizing inter- and intra- industry interoperability.

この OpenAPI Naming and Design Rules 技術仕様は、OpenAPI 仕様に基づいて API を指定、記述、および実装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、ビジネス情報を一貫して表現します。 これは、OpenAPI 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。 この仕様では、UN/CEFACT 準拠の API が満たすべき要件について説明します。 これは、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使用されます。

**<Wiki OpenAPI>**

The OpenAPI Specification, previously known as the Swagger Specification, is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services.[1] Previously part of the Swagger framework, it became a separate project in 2016, overseen by the OpenAPI Initiative, an open-source collaboration project of the Linux Foundation.[2] Swagger and some other tools can generate code, documentation, and test cases given an interface file.

以前は Swagger 仕様として知られていた OpenAPI 仕様は、RESTful Web サービスを記述、生成、消費、視覚化するための機械可読インターフェース ファイルの仕様です。[1] 以前は Swagger フレームワークの一部でしたが、2016 年に別のプロジェクトになり、Linux Foundation のオープンソース コラボレーション プロジェクトである OpenAPI イニシアチブによって監督されました[2]。 Swagger およびその他のツールは、インターフェイス ファイルを指定して、コード、ドキュメント、およびテスト ケースを生成できます。

[Abstract 2](#_bookmark0)

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

1. [INTRODUCTION 8](#_bookmark9)
   1. [Objectives 8](#_bookmark10)
   2. [Requirements 8](#_bookmark11)
   3. [Dependencies 8](#_bookmark12)
   4. [Caveats and Assumptions 8](#_bookmark13)
   5. [Guiding Principles 9](#_bookmark14)
   6. [Interoperability 10](#_bookmark15)
2. [API NAMING AND DESIGN RULES 13](#_bookmark16)
   1. [Conformance and Compliance 13](#_bookmark17)
   2. [Design Rules 14](#_bookmark18)
      1. [Media type for structured data exchange 14](#_bookmark19)
      2. [Endpoints 14](#_bookmark20)
      3. [Discoverability 16](#_bookmark23)
      4. [Date and Time 16](#_bookmark24)
      5. [Using the UN/CEFACT semantics 16](#_bookmark25)
      6. [Operations 21](#_bookmark28)
      7. [Pagination 24](#_bookmark30)
      8. [Filtering 26](#_bookmark32)
      9. [Sorting 28](#_bookmark34)
      10. [API Responses and error handling 28](#_bookmark35)
      11. [Error Response Payload 34](#_bookmark37)
      12. [Design rule examples 36](#_bookmark38)
3. [WELL-DOCUMENTED APIS 37](#_bookmark39)
   1. [General considerations 37](#_bookmark40)
   2. [API Versioning 37](#_bookmark41)
      1. [Versioning Scheme 37](#_bookmark42)
      2. [URI Versioning 38](#_bookmark46)
      3. [Providing version information 38](#_bookmark47)
      4. [Robustness 40](#_bookmark48)
      5. [Deprecation and End of Life Policy 41](#_bookmark50)
   3. [Hypermedia 44](#_bookmark53)
      1. [Hypermedia - Linked Data 44](#_bookmark54)
      2. [HATEOAS 44](#_bookmark55)
      3. [Hypermedia Compliant API 45](#_bookmark57)
4. [API SECURITY 47](#_bookmark59)
5. [EVENT DRIVEN DATA EXCHANGE 49](#_bookmark60)
   1. [Callbacks 50](#_bookmark61)
   2. [Webhooks 50](#_bookmark62)
   3. [Security guideline for callbacks (informative) 50](#_bookmark63)
      1. [Subscription setup (informative) 51](#_bookmark64)
      2. [Performing a subscription call (informative) 52](#_bookmark67)
6. [APPENDIX A: EXAMPLES 54](#_bookmark68)
7. [APPENDIX B: NAMING AND DESIGN RULES LIST (NORMATIVE) 55](#_bookmark69)
8. [APPENDIX C: GLOSSARY 60](#_bookmark70)

### Document History

### Change Log

**Table 1 – Document history**

|  |  |  |
| --- | --- | --- |
| **Phase** | ***Status*** | ***Date Last Modified*** |
| Draft development | First draft | 06 September 2022 |
| Ready for approval | First version | 13 September 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** |
| 30 May 2022 | 0.3 |  | First draft TOC |
| 07 June 2022 | 0.4 |  | Drafted up to chapter 3.2.7 |
|  | 0.5 |  | Drafted up to chapter 3.2.9 |
| 20 June 2022 | 0.6 |  | Completion up to chapter 6 |
| 05 Sept 2022 | 0.7 | 1.6  2.6  R 1  R 16  6.3  7  Appendix A Appendix B Appendix C | Considering public review comments |
| 13 Sept 2022 | 1.0 |  | Minor corrections |

**Table 2 - Document change log**

### OpenAPI 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) OpenAPI 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 OpenAPI 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:

UN/CEFACT OpenAPI Naming and Design Rules Technical Specification のこのバージョンは、標準開発組織 (SDO) 間の収束を促進するために作成されました。 これは、次の組織と緊密に連携して開発されました。

* + - Digital Container Shipping Association
    - 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

+Inf

|

category of the given rule as defined in section [3.1](#_bookmark17). A identify rules that are informative and not normative.

may be added to

**Courier** All words appearing in **bolded courier font** are values, objects or keywords.

Representation of non-printable characters like white space 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 OpenAPI 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 OpenAPI specifications and recommendations.
    - The wider membership of the other UN/CEFACT Groups who participate in the process of creating and maintaining UN/CEFACT OpenAPI specifications.
    - Designers of tools who need to design OpenAPI specifications adhering to the rules defined in this document.
    - Designers of OpenAPI specifications 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 OpenAPI 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 OpenAPI 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することを目的とした一連のドキュメントの一部です。

Taking any layer of the UN/CEFACT Reference Data Models to create conformant OpenAPI specifications 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 (SC-RDM), 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 コア コンポーネント技術仕様バージョン 2.01 に従って、UN/CEFACT 参照データ モデルの任意のレイヤーを使用して、準拠した OpenAPI 仕様を作成します。 これには、Buy-Ship-Pay や Accounting などの包括的な RDM と、Supply-Chain-Reference-Data-Model (SC-RDM)、Multi-Modal-Transport-Reference-Data-Model (MMT-RDM) などのコンテキスト化が含まれます。 ) ロード コンサインメント ノート (eCMR) や原産地証明書 (COO) などの単一メッセージの実装に至るまで。

### Requirements

Users of this specification should have an understanding of basic data modelling concepts, basic business information exchange concepts and basic (REST) API 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 (REST) API の概念を理解している必要があります。

### Dependencies

This document depends on

1. UN/CEFACT Core Components Technical Specification Version 2.01.
2. JSON Schema Naming and Design Rules Technical Specification.

### Caveats and Assumptions

Specifications created as a result of employing this specification should be made publicly available as OpenAPI specification 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仕様は、広く無料でアクセス可能で検索可能なライブラリで OpenAPI 仕様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 the data structures used in an OpenAPI specification as expressions of Reference Data Models, non-CCTS developers can also use it for other logical data models and information exchanges.

この仕様は、OpenAPI 仕様で使用されるデータ構造を参照データ 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。

Standards foster interoperability. In the creation of this specification and definition of design principles, several sources were taken into account in the following order:

標準は相互運用性を促進します。 この仕様の作成と設計原則の定義では、次の順序でいくつかの情報源が考慮されました。

* + 1. The OpenAPI 3.1.0 specification
    2. Standards defined by internet standard organisations as RFCs
    3. The DCSA API Design Principles 1.0
    4. The json:api specification
    5. Experts experience

### Guiding Principles

1. OpenAPI Creation

UN/CEFACT OpenAPI design rules support OpenAPI specification creation through handcrafting as well as automatic generation.

UN/CEFACT OpenAPI 設計ルールは、手作業による OpenAPI 仕様の作成と自動生成をサポートします。

1. Tool Use and Support

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

UN/CEFACT OpenAPI の設計では、作成、管理、保存、またはプレゼンテーション用の高度なツールが利用可能であることを前提としていません。

1. Technical Specifications

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

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

1. OpenAPI Specification

UN/CEFACT OpenAPI Naming and Design Rules will be fully conformant with the OpenAPI specification recommendation.

UN/CEFACT OpenAPI 命名規則と設計規則は、OpenAPI 仕様の推奨事項に完全に準拠します。

1. Interoperability

The number of ways to express the same information in a UN/CEFACT OpenAPI specification is to be kept as close to one as possible.

UN/CEFACT OpenAPI 仕様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。

1. Maintenance

The design of an UN/CEFACT OpenAPI specification must facilitate maintenance.

UN/CEFACT OpenAPI 仕様の設計は、保守を容易にする必要があります。

1. Context Sensitivity

The design of an UN/CEFACT OpenAPI specification must ensure that context- sensitive document types are not precluded.

UN/CEFACT OpenAPI 仕様の設計では、コンテキスト依存のドキュメント タイプが除外されないようにする必要があります。

1. Ease of implementation

An UN/CEFACT OpenAPI specification 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 OpenAPI 仕様は、設計対象の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。

### Interoperability

Decades of cross-industry and cross-national harmonisation of B2B and B2A processes have gone into the development of the semantic UN/CEFACT reference data models by thousands of experts.

何十年にもわたる B2B および B2A プロセスの業界横断的および国境を越えた調和は、何千人もの専門家によるセマンティック UN/CEFACT 参照データ モデルの開発に費やされてきました。

This tremendous achievement does not exist a second time in this scope and depth. The clear path from semantic definition to syntax - and not vice versa - means that these semantic data models are syntax-neutral and can thus be used not only with current syntaxes but also with future ones.

この驚異的な成果は、この範囲と深さで 2 度と存在しません。 セマンティック定義から構文への明確なパス (およびその逆ではない) は、これらのセマンティック データ モデルが構文中立であり、現在の構文だけでなく将来の構文でも使用できることを意味します。

For this purpose, either they are mapped directly into a (UN/CEFACT) syntax via NDR specifications, or they can be mapped to data models and syntaxes of other sectors.

この目的のために、それらは NDR 仕様を介して (UN/CEFACT) 構文に直接マップされるか、他のセクターのデータ モデルと構文にマップされます。

The ideal of a REST API envisages the fully automatable connection of an API consumer to an API provider. In practice, this is often not the case today, as the corresponding standards for the design of an API, the scope and depth of the documentation and the modelling of processes and data in B2B and B2A communication via WebAPIs are still in their infancy. The keyword here is interoperability.

REST API の理想は、API コンシューマーから API プロバイダーへの完全に自動化された接続を想定しています。 API の設計、ドキュメンテーションの範囲と深さ、および WebAPI を介した B2B および B2A 通信におけるプロセスとデータのモデリングに対応する標準は、まだ初期段階にあるため、実際には、これは今日では当てはまらないことがよくあります。 ここでのキーワードは相互運用性です。

In classic EDI implementations (e.g. EDIFACT or XML), a variety of industry standards exist. With their help, the following dimensions of interoperability are promoted:

従来の EDI 実装 (EDIFACT や XML など) には、さまざまな業界標準が存在します。 彼らの助けを借りて、相互運用性の次の側面が促進されます。

* + 1. Business process interoperability: the business partners have the same understanding of the basic process flow, for example in the Order2Cash - process.

1. ビジネス プロセスの相互運用性: ビジネス パートナーは、Order2Cash プロセスなどの基本的なプロセス フローについて同じ理解を持っています。

* + 1. Semantic interoperability: the business partners have the same understanding of the technical terms. For example, the definition of consignment and shipment is the same for all business partners.

2. セマンティックな相互運用性: ビジネス パートナーは、技術用語について同じ理解を持っています。 たとえば、委託と出荷の定義は、すべての取引先で同じです。

* + 1. Syntax interoperability: a uniform syntax (e.g. UN/CEFACT XML) is used.

3.構文の相互運用性: 統一された構文 (UN/CEFACT XML など) が使用されます。

* + 1. Contextualisation interoperability: Industry standards define how individual requirements are to be handled. Ideally, it is agreed that as few different contextualisations (consideration of individual requirements) as possible should take place. This means that information that is only required by some recipients will be read over by the remaining recipients instead of carrying out an individual implementation.

4. コンテキスト化の相互運用性: 業界標準は、個々の要件を処理する方法を定義します。 理想的には、可能な限り異なるコンテキスト化 (個々の要件の考慮) を行うべきであるということが合意されています。 これは、一部の受信者のみが必要とする情報が、個別の実装を実行する代わりに、残りの受信者によって読み取られることを意味します。

* + 1. Interoperability of transmission: Business partners agree on uniform transmission methods as well as associated security measurements such as SFTP, OFTP2 or AS2. This dimension often plays a lesser role in classic EDI implementations, as the transmission of data usually takes place from one sender to one recipient at a time. EDI is usually optimised for mass data.

5. 送受信の相互運用性: ビジネス パートナーは、SFTP、OFTP2、または AS2 などの関連するセキュリティ対策と同様に、統一された送信方法について合意します。 通常、データの送信は一度に 1 人の送信者から 1 人の受信者に行われるため、従来の EDI 実装では、このディメンションはあまり役割を果たしません。 通常、EDI は大量データ用に最適化されています。

When implementing a WebAPI, the same requirements for interoperability exist in principle.

WebAPI を実装する場合、相互運用性に関する同じ要件が原則として存在します。

An essential difference of previous WebAPIs is the approach to connect mas users to an API.

以前の WebAPI との本質的な違いは、マス ユーザーを API に接続するアプローチです。

For example, a map, route or booking service should be used by as many users as possible at the same time.

たとえば、地図、ルート、または予約サービスは、できるだけ多くのユーザーが同時に使用する必要があります。

The REST principle of composability also means that different services (possibly from different providers) are often combined into an overall solution for processing with WebAPIs. For example, in a flight booking service, the capacities, conditions and tickets are allocated by the airlines, payment service providers are connected, and often a specialised billing service that correctly calculates the different tax constellations for cross-border flights.

構成可能性の REST 原則は、さまざまなサービス (おそらくさまざまなプロバイダーから) が、WebAPI で処理するための全体的なソリューションに結合されることが多いことも意味します。 たとえば、フライト予約サービスでは、キャパシティ、条件、および航空券が航空会社によって割り当てられ、支払いサービス プロバイダーが接続され、多くの場合、国境を越えたフライトのさまざまな税金コンステレーションを正しく計算する特殊な請求サービスが行われます。

The aspect that many consumers have to use one API (billing service) as well as one consumer has to use many APIs with the same processes (contingent with airlines) extends the interoperability requirements for WebAPIs.

多くの消費者が 1 つの API (請求サービス) を使用しなければならず、1 人の消費者が同じプロセス (航空会社によって異なる) で多くの API を使用しなければならないという側面は、WebAPI の相互運用性要件を拡張します。

Interoperability of API design: This specification deals with the aspect of API design interoperability. Uniform methods and rules in API design simplify the understanding of APIs, errors during implementation are minimised, the handling of error messages is standardised and the implication of similar APIs in a cross- organisational (B2B) network is promoted.

API 設計の相互運用性: この仕様は、API 設計の相互運用性の側面を扱います。 API 設計における統一された方法とルールにより、API の理解が簡素化され、実装中のエラーが最小限に抑えられ、エラー メッセージの処理が標準化され、組織間 (B2B) ネットワークにおける同様の API の影響が促進されます。

* + 1. Service interoperability: uniform endpoints in mapping the same process requirements promote B2B communication via WebAPIs.

6. サービスの相互運用性: 同じプロセス要件のマッピングにおける統一されたエンドポイントは、WebAPI を介した B2B 通信を促進します。

The following table shows how the seven dimensions of interoperability can be achieved in WebAPIs:

次の表は、相互運用性の 7 つの側面を WebAPI で実現する方法を示しています。

|  |  |  |  |
| --- | --- | --- | --- |
| **Dimension of**  **interoperability** | | **Guideline** | |
| **Business process interoperability** | | Within UN/CEFACT, business process interoperability is achieved by implementing the harmonised business requirement  specifications (BRS). | |
| **Semantic interoperability** | | The CCTS and its derived semantic Reference Data Models (RDMs) are the basis for this dimension for UN/CEFACT users. The UN/CEFACT Vocabulary, the JSON Schema artefacts, and the UN/CEFACT XML standards implement these semantic  requirements in the respective syntax. | |
| **Syntax interoperability** | | When a user group agrees on the use of a uniform data exchange syntax, this dimension is achieved. When creating an OpenAPI specification, it should be noted that the syntax to be used must always be modelled as a JSON schema, even if the later exchange syntax is an XML format, for example. It is defined in an OpenAPI specification. | |
| **Contextualisation interoperability** | | An implementation guideline (for example, of a particular industry) defines how contextualisations are to be applied to the data or  message structures to be exchanged. | |
| **Interoperability of transmission** | | This dimension is also specified in an implementation guideline. In particular, it also includes security aspects including authorisation  and authentication. | |
| **Interoperability of API design** | | This NDR specification defines the interoperability of API design. Among others, it includes rules for filtering, pagination and error  handling. | |
| **Service interoperability** | | A good OpenAPI specification especially focuses on service interoperability. The interoperability of APIs designed to be implemented by several business partners can be fostered if the services are well designed. For instance, a user group agrees on a set of services with a minimum subset. If a provider does not support a specific service it is still implemented, but always responds with a **501 Method not implemented** HTTP response code that includes a **HTTP Link Header** to the corresponding  documentation | |

**Table 3: Interoperability of WebAPIs**

# API Naming and Design Rules

### Conformance and Compliance

Designers of OpenAPI specifications in governments, private sector, and other standards organisations external to the UN/CEFACT community have found this specification suitable for adoption.

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 OpenAPI 仕様の設計者は、この仕様が採用に適していることを発見しました。

To maximize reuse and interoperability across this wide user community, the rules in this specification have been categorised to allow these other organisations to create conformant OpenAPI specifications while allowing for discretion or extensibility in areas that have minimal impact on overall interoperability.

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準拠した OpenAPI 仕様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.

したがって、アプリケーションが規範的なセクション、規則、および定義の内容に準拠している場合、アプリケーションはこの技術仕様に完全に準拠していると見なされます。

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

|  |  |
| --- | --- |
| **Category** | **Description** |
| **1** | Rules, which must not be violated. Else, compliance and interoperability are  lost. |
| **2** | Rules, which may be modified, while still conformant to the NDR structure. If all rules of categories 1 and 2 are followed, the API is fully compliant. If rules of category 2 are modified the API is not compliant anymore, but still conformant. |
| **Inf** | Rules that are informative only. If a different implementation is chosen this does not have any impact on the compliance and conformance of the implementation towards this specification. |

**Table 4 - Conformance categories**

|  |
| --- |
| [R 2|1] |
| All API specifications based on this OpenAPI Naming and Design Rules technical specification SHALL be compliant to the OpenAPI 3.1.x specification. |

|  |
| --- |
| [R 3|1] |
| An API specification claiming conformance to this specification SHALL define schema components as described in the JSON Schema Naming and Design Rules Technical Specification. |

### Design Rules

#### Media type for structured data exchange

|  |
| --- |
| [R 4|1] |
| Request body content and Response content used to transfer structured data information SHALL use the **application/json** media type for JavaScript Object Notation (JSON). This rule MAY only be deviated from, if the API implements a conversion service from or to JSON in another media type.  Additional media types (e.g. **text/xml**) to transfer structured data information MAY be used. If non-structured information is transferred any valid media type MAY be used. |

|  |
| --- |
| [R 5|1] |
| Encoding SHALL be UTF-8. |

#### Endpoints

|  |
| --- |
| [R 6|2] |
| The structure of the paths defined within APIs SHOULD be meaningful to the consumers. Paths SHOULD follow a predictable, hierarchical structure to enhance understandability and therefore usability. |

|  |
| --- |
| [R 7|1] |
| The API URLs SHOULD follow the standard naming convention as described below: |
| https://{env}.api.{dnsdomain}/v{m}/{service}/{resource}/{id}/{sub- resource}?{query} |
| The components are described as follows. If a rule is mandatory for a specific component of the URL it SHALL be applied to any conformant API specification, even if the basic URL structure is different from the one described above (e.g. if **api** is not used as a prefix to the  **dnsdomain**).   * https:// SHALL be used as the web protocol. * {env} indicates the environment (e.g. **test**, **sandbox** or **dev**) and is usually omitted for production environment. * {dnsdomain} is the DNS domain of the API implementer (e.g. **unece.org**) * {service} is a logical grouping of API functions that represent a business service domain (e.g. **transport**). The {service} component is optional. * v{m} is the major version number of the API specification. This component SHALL be stated in the URL. It MAY be provided at a different place in the URL (e.g. as a prefix to the domain). * {resource} is the plural noun representing an API resource (e.g. **consignments**) |

|  |
| --- |
| * {id} is the unique identifier for the resource defined as a path parameter. Path parameters SHALL be used to identify a resource. This component is not part of the path if an operation is performed on a collection of the resource. * {sub-resource} is an optional sub-resource. Only used when there are contained collections or actions on a main resource (e.g. **consignmentItem**). * {query} is a list of additional parameters like filters that determine the results of a search (e.g. **consignments?loadingPort=AUSYD**). |

|  |
| --- |
| [R 8|1] |
| The total number of characters in the URL, including the path and the query, SHALL NOT exceed 2000 characters in length including any formatting codes such as commas, underscores, question marks, hyphens, plus or slashes. |

|  |
| --- |
| [R 9|1] |
| Endpoints SHALL NOT be actions. Services and resources SHALL consist of nouns. HTTP verbs SHALL be used for actions (See chapter 3.2.6). |

|  |
| --- |
| [R 10|1] |
| Kebab-case[2](#_bookmark21) SHALL be used in services. |

|  |
| --- |
| [R 11|1] |
| Lower camelCase[3](#_bookmark22) SHALL be used in resources, path parameters and query parameters. |

|  |
| --- |
| [R 12|1] |
| Path parameters and query parameters with a relation to property names SHALL be consistent with property names. |

|  |
| --- |
| [R 13|1] |

2 Kebab-case is a naming rule for a technical representation of identifiers consisting of several words. Hyphens are used to connect words. Example: **this identifier** is written as **this-identifier** in kebap-case.

3 CamelCase is a naming rule for a technical representation of identifiers consisting of several words. White spaces are removed and every new word begins with a capital letter. Example: **this identifier** is written as **thisIdentifier** in camelCase. Lower camelCase means that the identifier must start with a small letter.

|  |
| --- |
| Query parameters SHALL be URL safe[4](#_bookmark26). |

|  |
| --- |
| [R 14|1] |
| Resource names SHALL be pluralised. Resource names SHOULD be consistent with schemas. If a schema is defined in singular, nevertheless the resource SHALL be pluralized. If the plural of a resource is non-standard, you MAY choose a more appropriate noun in its plural form. |

|  |
| --- |
| Examples for good endpoints: |
| * /employees * /customers * /products |

#### Discoverability

One of the REST design principles is service discoverability. The OpenAPI specification supports them via links. They SHALL be implemented via HTTP headers.

REST 設計原則の 1 つは、サービスの発見可能性です。 OpenAPI 仕様は、リンクを介してそれらをサポートしています。 これらは、HTTP ヘッダーを介して実装する必要があります。

#### Date and Time

The date and time representation in the CCL supports an ISO8601 subset with only a few exceptions. Those exceptions may be present in the content body of a request or a response.

CCL の日付と時刻の表現は、いくつかの例外を除いて ISO8601 サブセットをサポートしています。 これらの例外は、リクエストまたはレスポンスのコンテンツ本文に存在する場合があります。

|  |
| --- |
| [R 15|1] |
| Query parameters SHALL use ISO8601 compliant date and time representations that are defined in **UNTDID 2379 json** as defined in the JSON schema NDR technical specification. To represent a specific date, time or date-time the format SHALL comply with the JSON  schema definition for date, time or date-time. |

#### Using the UN/CEFACT semantics

Decades of harmonisation and standardisation of business requirements resulted in the UN/CEFACT reference data models (RDM). These exist across different domains like Buy- Ship-Pay, Agriculture, Regulatory or Audit and Accounting.

数十年にわたるビジネス要件の調和と標準化により、UN/CEFACT 参照データ モデル (RDM) が生まれました。 これらは、Buy-Ship-Pay、農業、規制、または監査と会計などのさまざまなドメインに存在します。

As one example the Buy-Ship-Pay RDM contains subsets e.g. for multimodal transport (MMT-RDM) and the supply chain (SC-RDM). Over time, hundreds of business document structures were harmonised and standardised on a semantic model level.

一例として、Buy-Ship-Pay RDM にはサブセットが含まれています。 マルチモーダル輸送（MMT-RDM）およびサプライチェーン（SC-RDM）向け。 時間の経過とともに、何百ものビジネス ドキュメント構造がセマンティック モデル レベルで調和および標準化されました。

Different Syntax Naming and Design rules allow an automated creation and mapping of those semantic models to certain syntaxes such as XML.

さまざまな構文命名規則と設計規則により、これらのセマンティック モデルの自動作成と、XML などの特定の構文へのマッピングが可能になります。

In the world of web APIs, the transmission of document structures is considered obsolete. If the limitations of REST principles are to be applied to a web API, business document structures are unsuitable for a RESTful implementation.

Web API の世界では、ドキュメント構造の送信は時代遅れと見なされています。 REST 原則の制限を Web API に適用する場合、ビジネス ドキュメントの構造は RESTful な実装には適していません。

These structures contradict the basic principle of loose coupling of resources. Instead, the exchange of information should be resource-based, where resources are information blocks leading in their combination to the complete information (e.g. business document).

これらの構造は、リソースの疎結合の基本原則に反します。 代わりに、情報の交換はリソースベースである必要があります。ここで、リソースは情報ブロックであり、それらを組み合わせて完全な情報 (ビジネス文書など) に導きます。

Nevertheless, there are often limitations in B2B information exchange that make it difficult to completely move away from document structures. This includes technical reasons, procedural reasons, but also legal reasons.

それにもかかわらず、B2B 情報交換には多くの場合、文書構造から完全に移行することを困難にする制限があります。 これには、技術的な理由、手続き上の理由だけでなく、法的な理由も含まれます。

If the basic processes of communication between organisations are not changed, a shift purely to resource-based information exchange leads to a new level of media disruption and consistency challenges.

組織間のコミュニケーションの基本的なプロセスが変更されない場合、純粋にリソースベースの情報交換への移行は、新しいレベルのメディアの混乱と一貫性の課題につながります。

If both the sending and receiving systems work on the basis of document structures (e.g. an invoice), then an intermediate, purely resource-based transmission leads to a number of challenges, such as the archiving obligation of such documents that exists in many countries to ensure subsequent verification.

送信システムと受信システムの両方が文書構造 (請求書など) に基づいて機能する場合、中間の純粋にリソースベースの送信は、多くの国に存在するそのような文書のアーカイブ義務など、多くの課題につながります。 その後の検証を確実にします。

On the other hand, if networks of platforms (e.g. for logistics) are established, a resource- based exchange can still be useful for certain purposes. For example, a platform could exist for a marketplace where free delivery capacities by carriers can be offered and booked. The division by resources usually leads to the need for identity providers and the clarification of the question of the single source of trust for individual resources.

一方、プラットフォームのネットワーク（物流など）が確立された場合でも、リソースベースの交換は特定の目的に役立ちます。 たとえば、キャリアによる無料の配送容量を提供および予約できる市場向けのプラットフォームが存在する可能性があります。 リソースによる分割は、通常、ID プロバイダーの必要性と、個々のリソースの信頼の単一ソースに関する問題の明確化につながります。

At UN/CEFACT, there are two basic JSON-based publications of semantic data models: the UN/CEFACT vocabulary, and the UN/CEFACT JSON schema publication.

UN/CEFACT では、セマンティック データ モデルの基本的な JSON ベースの出版物が 2 つあります。UN/CEFACT ボキャブラリと UN/CEFACT JSON スキーマ出版物です。

## Using the UN/CEFACT JSON schema publication

JSON schema is the natural partner of an OpenAPI specification, as OpenAPI relies on JSON schema. The UN/CEFACT JSON schemas are published in two variants:

OpenAPI は JSON スキーマに依存しているため、JSON スキーマは OpenAPI 仕様の自然なパートナーです。 UN/CEFACT JSON スキーマは、次の 2 つのバリアントで公開されています。

* + 1. Streamlined stand-alone JSON schemas for the individual business documents. Those schemas contain every definition relevant for a specific business document and its applied contextualisation.

7. 個々のビジネス ドキュメント用の合理化されたスタンドアロン JSON スキーマ。 これらのスキーマには、特定のビジネス文書とその適用された文脈化に関連するすべての定義が含まれています。

* + 1. A JSON schema library of the different RDMs and their related business document structures. This variant uses an inheritance and validation technique supported by JSON schema. The basic data structures define the information blocks needed together in the reference data model. Subsets and contextualisation for the individual applications (e.g. MMT-RDM, SC-RDM, Invoice ...) are then formed on this basis.
    2. さまざまな RDM とそれに関連するビジネス ドキュメント構造の JSON スキーマ ライブラリ。 この亜種は、JSON スキーマでサポートされている継承と検証の手法を使用します。 基本的なデータ構造は、参照データ モデルで一緒に必要とされる情報ブロックを定義します。 次に、個々のアプリケーション (MMT-RDM、SC-RDM、請求書など) のサブセットとコンテキスト化が、これに基づいて形成されます。

The JSON schemas are published in the official UN/CEFACT repository. They can be used in two different ways:

JSON スキーマは、公式の UN/CEFACT リポジトリで公開されています。 これらは 2 つの異なる方法で使用できます。

First by referencing the needed data types directly from the repository. This leads to a maximum on interoperability. In an OpenAPI specification, it is easily possible to further contextualise (including extension) the JSON subschemas to the needed requirements of the specific process. This explicitly lets the users "tick off" unneeded optional attributes or supplementary components, restrict code lists or add user defined properties in a standardised and transparent way.

まず、必要なデータ型をリポジトリから直接参照します。 これにより、相互運用性が最大限に高まります。 OpenAPI 仕様では、JSON サブスキーマを特定のプロセスに必要な要件に合わせてさらにコンテキスト化 (拡張を含む) することが容易に可能です。 これにより、ユーザーは、不要なオプションの属性や補足コンポーネントを明示的に「チェック」したり、コード リストを制限したり、標準化された透過的な方法でユーザー定義のプロパティを追加したりできます。

Additionally, maintenance becomes quite easy. If the API is to be updated to a newer version of the JSON schema publication, only the reference needs to be updated.

さらに、メンテナンスが非常に簡単になります。 API を新しいバージョンの JSON スキーマ パブリケーションに更新する場合は、参照のみを更新する必要があります。

Alternatively, the JSON schemas can be downloaded to a local system or repository. In that case it is needed to update or remove the **"$id"** properties of the schemas, as they link to the official UN/CEFACT repository.

または、JSON スキーマをローカル システムまたはリポジトリにダウンロードすることもできます。 その場合、公式の UN/CEFACT リポジトリにリンクしているため、スキーマの「$id」プロパティを更新または削除する必要があります。

The way in which the JSON schemas are defined allow a very simple transmission from using document-based structures to resource-based structures.

JSON スキーマが定義される方法により、ドキュメント ベースの構造からリソース ベースの構造への非常に単純な伝達が可能になります。

On the RDM level, all ABIEs (data classes) are defined. For every RDM exists a master document structure. All of the business documents are derived from this. The hierarchic structure connects the different ABIEs through ASBIEs including cardinality information. At every single ASBIE node, the JSON schema publication allows to replace the provision of a substructure by the URN of the corresponding resource:

RDM レベルでは、すべての ABIE (データ クラス) が定義されます。 すべての RDM には、マスター ドキュメント構造が存在します。 すべてのビジネス文書はこれに由来します。 階層構造は、カーディナリティ情報を含む ASBIE を介してさまざまな ABIE を接続します。 すべての単一の ASBIE ノードで、JSON スキーマの発行により、サブ構造のプロビジョニングを対応するリソースの URN に置き換えることができます。

Let us assume you want to define an API to manage transport capacity booking. In a classic message-based scenario, you would define how those messages are interchanged. In many case you would design a **POST** and **GET** or **POST**, subscribe and **GET** scenario.

輸送能力の予約を管理するための API を定義するとします。 従来のメッセージ ベースのシナリオでは、これらのメッセージを交換する方法を定義します。 多くの場合、POST と GET のシナリオ、または POST、サブスクライブと GET のシナリオを設計します。

Those scenarios need envelope-information around the message information in order to tell the API who the ultimate receiver is, who the sender is etc. In addition the message is quite complex and contains many sub-resources with details. Those include for instance "requester", "shipTo", "receiver", "carrier", "consignment-items" etc.

これらのシナリオでは、最終的な受信者や送信者などを API に伝えるために、メッセージ情報の周りにエンベロープ情報が必要です。さらに、メッセージは非常に複雑で、詳細を含む多くのサブリソースが含まれています。 これらには、たとえば、「依頼者」、「発送先」、「受取人」、「運送業者」、「委託品」などが含まれます。

If this scenario is planned to move towards a (more) resource-based information exchange it is very easy to do so.

このシナリオが (より多くの) リソースベースの情報交換に移行する予定である場合、それは非常に簡単です。

First, you have to identify which of your sub-resources should become stand-alone. Let us assume you want to manage trade party information master data as a single resource. In that case, you can specify a schema under **components/schemas** named **tradePartyType** and simply define it as a reference to the contextualised data type of the corresponding RDM or even the corresponding business document structure.

まず、どのサブリソースをスタンドアロンにする必要があるかを特定する必要があります。 取引先情報マスター データを 1 つのリソースとして管理するとします。 その場合、tradePartyType という名前のコンポーネント/スキーマの下にスキーマを指定し、対応する RDM または対応するビジネス ドキュメント構造のコンテキスト化されたデータ型への参照として単純に定義できます。

The following example shows, how the document structure can be restricted to resource usage as well.

次の例は、ドキュメント構造をリソースの使用にも制限する方法を示しています。

|  |  |
| --- | --- |
| Example for a tradePartyType under components/schemas: | |
| "tradePartyType": {  "description": "Trade party definition according to MMT-RDM", "$ref": "https://raw.githubusercontent.com/uncefact/spec-  JSONschema/main/JSONschema2020-12/library/BuyShipPay/D22A/UNECE- MMTContextCCL.json#/$defs/tradePartyType"  }  "tradePartyType": {  "description": "Trade party definition according to the Multimodal Transport Booking Recipient",  "$ref": "https://raw.githubusercontent.com/uncefact/spec- JSONschema/main/JSONschema2020-12/library/BuyShipPay/D22A/UNECE- MultimodalTransportBooking.json#/exchangedDocument/recipient"  }  "multimodalTransportBooking": {  "title": "Multimodal Transport Booking",  "description": "Restrict business document to resource usage for recipient",  "allOf": [  { "$ref": "https://raw.githubusercontent.com/uncefact/spec- JSONschema/main/JSONschema2020-12/library/BuyShipPay/D22A/UNECE- MultimodalTransportBooking.json/#" },  {  "properties": { "exchangedDocument": {  "properties": {  "recipient": { "type": "string", "format": "uri" }  }  }  }  }  ]  } | |

## Using the UN/CEFACT vocabulary

The UN/CEFACT vocabulary uses the JSON-LD format in order to be conformant with the publication on schema.org.

UN/CEFACT 語彙は、schema.org の出版物に準拠するために JSON-LD 形式を使用します。

The publication in JSON-LD follows a different approach. JSON-LD is a graph representation of context-enhanced semantic ABIE-representations derived from the combination of the corresponding RDMs. By applying the appropriate context, the subset of the defined graph can be used.

JSON-LD での公開は、別のアプローチに従います。 JSON-LD は、対応する RDM の組み合わせから派生したコンテキスト拡張セマンティック ABIE 表現のグラフ表現です。 適切なコンテキストを適用することにより、定義されたグラフのサブセットを使用できます。

JSON-LD cannot directly be used and linked to in an OpenAPI specification. According to the maintenance body of the OpenAPI specification, this is not intended to change in the　near future.

JSON-LD は、OpenAPI 仕様で直接使用およびリンクすることはできません。 OpenAPI 仕様の保守団体によると、これは近い将来に変更される予定はありません。

In addition, the JSON-LD does not specify the cardinalities and subsets for the different contexts of business document structure definitions. Therefore, a web developer implementing an API for business related intra-organisational information exchange needs a reasonable knowledge of the underlying processes.

さらに、JSON-LD は、ビジネス ドキュメント構造定義のさまざまなコンテキストのカーディナリティとサブセットを指定しません。 したがって、ビジネス関連の組織内情報交換用の API を実装する Web 開発者は、基礎となるプロセスに関する合理的な知識を必要とします。

On the other hand, JSON-LD unfolds immense power wherever (publicly) available data is to be automatically crawled, filtered and evaluated.

一方、JSON-LD は、(公開されている) 利用可能なデータが自動的にクロール、フィルター処理、および評価される場合に、計り知れない力を発揮します。

Examples of this are applications such as flight-radar, online search for recipes or searches for goods over the boundaries of online shops with specific criteria.

この例としては、フライト レーダー、レシピのオンライン検索、または特定の基準によるオンライン ショップの境界を越えた商品の検索などのアプリケーションがあります。

In order to use the JSO In those scenarios, the individual resources get into focus, as well as their relationships (links) to other resources. The business-related-interdependencies are not part of the definitions themselves. Adding state machines in definitions could help with this. Unfortunately, currently there does not exist a widely supported exchange format for this kind of information[5](#_bookmark27).

これらのシナリオでは、個々のリソースだけでなく、他のリソースとの関係 (リンク) にも焦点が当てられます。 ビジネス関連の相互依存関係は、定義自体の一部ではありません。 定義にステート マシンを追加すると、これに役立ちます。 残念ながら、現在、この種の情報を広くサポートする交換フォーマットは存在しません。

The JSON-LD vocabulary, additional tooling must be used, as there does not exist a direct support in OpenAPI specifications. As a proof-of-concept, in the JSON-LD vocabulary publication, a sample implementation is included to import the vocabulary into a UML design tool. Here the first conversion from JSON-LD to UML is performed.

JSON-LD 語彙は、OpenAPI 仕様で直接サポートされていないため、追加のツールを使用する必要があります。 概念実証として、JSON-LD 語彙の出版物には、語彙を UML 設計ツールにインポートするためのサンプル実装が含まれています。 ここで、JSON-LD から UML への最初の変換が実行されます。

Now the designing of the API can be performed within the UML-Tool. Some assumptions are made how to define which operations should be supported for each of the specified endpoints.

これで、API の設計を UML ツール内で実行できるようになりました。 指定されたエンドポイントごとにどの操作をサポートする必要があるかを定義する方法について、いくつかの仮定が行われます。

Having defined this a second conversion from the UML-Tool to the OpenAPI specification format is performed.

これを定義すると、UML ツールから OpenAPI 仕様形式への 2 回目の変換が実行されます。

## Using other (standardised) data structures

In chapter [2.6,](#_bookmark15) seven dimension of interoperability for WebAPIs are defined. From a global cross-industry perspective, full interoperability can only be achieved if for all of the dimensions the implementation rules are clearly defined.

2.6 章では、WebAPI の相互運用性の 7 つの次元が定義されています。 グローバルな業界横断的な観点から、完全な相互運用性は、すべての次元について実装ルールが明確に定義されている場合にのみ実現できます。

In the context of UN/CEFACT, this means that the UN/CEFACT semantic definitions as well as the UN/CEFACT syntaxes must be used to be fully compliant.

UN/CEFACT のコンテキストでは、これは、UN/CEFACT セマンティック定義と UN/CEFACT 構文を完全に準拠するために使用する必要があることを意味します。

However, this NDR specification is syntax-neutral, as it defines basic requirements for the design of an OpenAPI specification in a B2B context. The stipulations in this specification can thus also promote interoperability between APIs that use a different syntax or divergent semantic specifications within a (closed) user group. Therefore, the following rule is defined as a conformance criterion:

ただし、この NDR 仕様は、B2B コンテキストでの OpenAPI 仕様の設計に関する基本的な要件を定義しているため、構文中立です。 したがって、この仕様の規定は、異なる構文を使用する API 間の相互運用性や、(クローズドな) ユーザー グループ内で異なるセマンティック仕様を促進することもできます。 したがって、次のルールが適合基準として定義されます。

|  |
| --- |
| [R 16|1] |
| A prerequisite for an OpenAPI specification and its implementation to be fully compliant with this NDR TS is the use of UN/CEFACT semantics and UN/CEFACT syntax (e.g.  UN/CEFACT XML, UN/CEFACT JSON Schema, and UN/CEFACT Vocabulary).  An OpenAPI specification that does not use UN/CEFACT syntax or UN/CEFACT  semantics may still be conformant to this NDR TS if it meets the criteria specified in [R 1|1]. |

#### Operations

|  |
| --- |
| [R 17|1] |
| Endpoints are RECOMMENDED to support CRUD operations. (Create, Read, Update, Delete). If an endpoint is not intended to support e.g. a delete operation, it SHALL return the HTTP response codes as defined in chapter [3.2.10](#_bookmark35). |

|  |  |
| --- | --- |
| **HTTP Method** | **Description** |
| *GET* | To *retrieve/read* a resource. |
| *POST* | To *create* a new resource or to *execute* an operation on a resource that changes the state of the system e.g. send a message. |
| *PUT* | To *replace* a resource with another supplied in the request. |
| *PATCH* | To perform a *partial update* to a resource. |
| *DELETE* | To *delete* a resource. |
| *HEAD* | For retrieving metadata about the request, e.g. how many results *would* a query return? (Without actually performing the query). This can be used to follow a link-chain in an HATEOS implementation as well. An example is shown in chapter [4.3.2.](#_bookmark55) |
| *OPTIONS* | Used to determine if a CORS (cross-origin resource sharing)  request can be made. This is primarily used in front-end web applications to determine if they can use APIs directly. |

## Collection of Resources

The following operations are applicable for a collection of resources:

次の操作は、リソースのコレクションに適用できます。

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP**  **method** | **Resource Path** | **Operation** | **Examples** |
| GET | */resources* | Get a collection of the resource | GET */employees* or  GET */employees?status=open* |
| HEAD | */resources* | Get header and link information of the resource collection, e.g.  for pagination | HEAD /employees or  HEAD /employees?birthday=2022-04-16 |

**Note**

Creating or updating multiple resource instances in the same request is not standardised and thus should be avoided. There are factors such as receipt acknowledgement and how to handle partial success in a set of batches that must be considered on a case-by-case basis.

## Single Resource

The following operations are applicable for a single resource:

次の操作は、単一のリソースに適用できます。

|  |  |  |
| --- | --- | --- |
| **HTTP method** | **Resource Path** | **Operation** |
| GET | */resources/{id}* | Get the instance corresponding to the resource ID |
| PUT | */resources/{id}* | To update a resource instance by replacing it – "*Take this new thing and* \_ **put** \_ *it there*" |
| DELETE | */resources/{id}* | To delete the resource instance based on the resource e.g. id |
| HEAD | */resources/{id}* | Get header and link information of the resource. |
| PATCH | */resources/{id}* | Perform changes such as add, update, and delete to the specified attribute(s). Is used often to perform  partial updates on a resource |

## Idempotency

An idempotent HTTP method is an HTTP method that can be called many times without different outcomes. In some cases, secondary calls will result in a different response code, but there will be no change of state of the resource.

冪等 HTTP メソッドは、異なる結果なしで何度も呼び出すことができる HTTP メソッドです。 場合によっては、2 次呼び出しによって異なる応答コードが返されますが、リソースの状態は変化しません。

As an example, when you invoke N similar DELETE requests, the first request will delete the resource and the response will be 200 (OK) or 204 (No Content). Further requests will return 404 (Not Found). Clearly, the response is different from first request, but there is no change of state for any resource on server side because the original resource is already deleted.

例として、同様の DELETE リクエストを N 回呼び出すと、最初のリクエストでリソースが削除され、レスポンスは 200 (OK) または 204 (No Content) になります。 それ以降のリクエストでは 404 (Not Found) が返されます。 明らかに、応答は最初の要求とは異なりますが、元のリソースは既に削除されているため、サーバー側のリソースの状態は変化しません。

|  |  |
| --- | --- |
| **HTTP Method** | **Is Idempotent** |
| *GET* | True |
| *POST* | False |
| *PUT* | True |
| *PATCH* | False |
| *DELETE* | True |
| *HEAD* | True |
| *OPTIONS* | True |

**Table 5 – Idempotency of operations**

|  |
| --- |
| [R 18|1] |
| APIs SHALL adhere to the idempotency of operations specified in the list above. |

|  |
| --- |
| [R 19|1] |
| APIs SHOULD implement the **Idempotency-Key**[**6**](#_bookmark29)HTTP header field and the corresponding  implementation advice in order to make non-idempotent operations like POST and PATCH fault-tolerant. |

6 https://[www.ietf.org/archive/id/draft-ietf-httpapi-idempotency-key-header-01.txt](http://www.ietf.org/archive/id/draft-ietf-httpapi-idempotency-key-header-01.txt)

#### Pagination

Querying an API with a GET can theoretically result in a huge return collection. Image querying the API of one of the big internet search engines without pagination. Hundreds of millions of results would have to be downloaded and displayed on a single page. That API would be unusable. Pagination helps to keep the data load to a reasonable amount and at the same time supports security aspects.

GET を使用して API をクエリすると、理論的には膨大なコレクションが返される可能性があります。 ページネーションなしで、大手インターネット検索エンジンの 1 つの API をクエリする画像。 何億もの結果をダウンロードして、1 つのページに表示する必要があります。 その API は使用できなくなります。 ページネーションは、データの負荷を妥当な量に維持するのに役立ち、同時にセキュリティ面をサポートします。

Historically, many APIs use offset pagination. A maximum page size (e.g. 20) is specified and the clients requests the starting record or the page number. However, this approach leads to fuzzy results: Suppose an API is supposed to return a list of all planned transport movements of a certain carrier ordered by destination. The first page of results is returned accurately. Before the client requests the next page or set of records, three possible things can happen.

歴史的に、多くの API はオフセット ページネーションを使用しています。 最大ページ サイズ (例: 20) が指定され、クライアントは開始レコードまたはページ番号を要求します。 ただし、このアプローチはあいまいな結果につながります。API が、特定の運送業者の計画されたすべての輸送移動のリストを目的地順に並べて返すことになっているとします。 結果の最初のページが正確に返されます。 クライアントが次のページまたは一連のレコードを要求する前に、3 つの可能性があります。

* The databank does not change at all. Then the next page of records is accurate.

• データバンクはまったく変更されません。 次に、レコードの次のページが正確です。

* A record is added to the database, which falls under the result list of the first page, which the client already received. In that case, the last result of the previous page is returned as the first result of the second page. The list therefore contains a duplicate.

• レコードがデータベースに追加されます。これは、クライアントが既に受信した最初のページの結果リストに該当します。 その場合、前のページの最後の結果が 2 ページ目の最初の結果として返されます。 したがって、リストには重複が含まれています。

* In the opposite case, a planned transport movement that has already been returned to the client on the first page is deleted. The first data record of the second page therefore moves to the previous page. If the client now queries the next page, this data record is not transmitted at all.

• 逆の場合、最初のページですでにクライアントに返された計画済みのトランスポート移動は削除されます。 したがって、2 ページ目の最初のデータ レコードは前のページに移動します。 クライアントが次のページを照会すると、このデータ レコードはまったく送信されません。

As an inter-organisational data exchange cannot accept this type of results, an alternative solution for pagination is needed. The solution to this problem is the so called keyset-based or cursor-pagination[7](#_bookmark31). In addition, cursor-pagination is much more time-efficient on large datasets than offset-pagination.

組織間のデータ交換はこの種の結果を受け入れることができないため、ページネーションの代替ソリューションが必要です。 この問題の解決策は、いわゆるキーセットベースまたはカーソルページネーション です。 さらに、大規模なデータセットでは、オフセット ページネーションよりもカーソル ページネーションの方がはるかに時間効率が高くなります。

|  |
| --- |
| [R 20|1] |
| If pagination is used in an API, keyset-based pagination (cursor-pagination) SHALL be used. This means that the consumer cannot request a specific page, instead the consumer has to select a page-link provided by the server. The server SHALL provide links in the HTTP response header to the previous and next page and SHOULD provide links to the first and last page. More links MAY be provided.  The cursor-value is a string, created by the server using whatever method it likes. It identifies a point in a list of results for a query containing filters and sorting parameters for a specific moment in time. Therefore, it divides the list into those that fall before the cursor and those that fall after the cursor. There may optionally be one result that falls "on" the cursor. |

7 [https://jsonapi.org/profiles/ethanresnick/cursor-pagination/,](https://jsonapi.org/profiles/ethanresnick/cursor-pagination/) https://medium.com/swlh/how-to-implement- cursor-pagination-like-a-pro-513140b65f32

Cursor-pagination assures a consistent data set for a query with filtering/sorting criteria at a specific moment in time. If another consumer performs the same query a moment later, he may get a different data set.

カーソルページネーションは、特定の時点でのフィルタリング/ソート基準を使用して、クエリの一貫したデータセットを保証します。 しばらくして別の消費者が同じクエリを実行すると、別のデータ セットを取得する可能性があります。

|  |
| --- |
| [R 21|1] |
| GET requests on collection results SHOULD implement pagination. The default and maximum page size SHOULD be 100, if not specified on the endpoint. If SHOULD be smaller, if the resulting page load is large. The default page size MAY be changed per endpoint. A consumer SHOULD be able to override the default page size.  If the filter, sorting and/or page size used is changed when getting a result, the pagination SHALL BE reset to the first page.  The query parameters described in the following table SHALL be used, rules SHALL be applied. |

|  |  |  |
| --- | --- | --- |
| **Type** | **Explanation** | **Example** |
|  |  |  |
| *Page size* | Overrides the default page size defined by the server / specification. | Example for the first query:  GET /transportMovements? carrier=ABC &status=PLANNED  &sort=estimatedTimeOfArrival  &pageSize=50 |
|  |  |  |
|  |  |  |
| *Current page* | A link to the current page. | Link: <https://api.unece.org/ transportMovements? cursor=XXX>;  rel="current" |
|  |  |  |
|  |  |  |
| *First page* | A link to the first page. If it is the first page the link MAY be omitted. | Link: <https://api.unece.org/ transportMovements? cursor=XXX>; rel="first" |
|  |  |  |
| *Next page* | A link to the next page. If it is the last page, the link to the next page MAY be omitted. Otherwise, a null link shall be provided. | Link: <https://api.unece.org/ transportMovements? cursor=XXX>; rel="next"  Link: <null>; rel="next" |
|  |  |  |
|  |  |  |
| *Previous page* | A link to the previous page. If it is the first page, the link to the previous page MAY be omitted. Otherwise, a null link shall be  provided. | Link: <https://api.unece.org/ transportMovements? cursor=XXX>; rel="prev" |

|  |  |  |
| --- | --- | --- |
| **Type** | **Explanation** | **Example** |
|  |  |  |
| *Last page* | A link to the last page. If it is the last page, the link to the last page MAY be omitted. Otherwise, a  null link shall be provided. | Link: <https://api.unece.org/ transportMovements? cursor=XXX>; rel="last" |

When multiple links are given, they are separated by comma.

|  |
| --- |
| Example for a combination of Links:  Link:  <https://api.unece.org/transportMovements?cursor=XXX>; rel="current",  <https://api.unece.org/transportMovements?cursor=YYY>; rel="first",  <https://api.unece.org/transportMovements?cursor=ZZZ>; rel="next",  <https://api.unece.org/transportMovements?cursor=LLL>; rel="last" |

#### Filtering

Providing the ability to filter and sort collections in an API allows your consumers greater flexibility and controls on how they choose to consume a conformant API.

API でコレクションをフィルター処理およびソートする機能を提供することで、コンシューマーは適合 API を使用する方法をより柔軟に制御できるようになります。

|  |
| --- |
| [R 22|1] |
| Sorting and filtering SHALL be done using query parameters. Using a path parameter is  only allowed to identify a specific resource. |

## Output Selection

Consumers can specify the attributes they wish to return in the response payload by specifying the attributes in the query parameters

コンシューマーは、クエリ パラメーターで属性を指定することにより、応答ペイロードで返したい属性を指定できます。

|  |
| --- |
| Example that returns only the *first\_name* and *last\_name* fields in the response:  *?attributes=first\_name,last\_name* |

## Simple Filtering

Attributes can be used to filter a collection of resources.

属性を使用して、リソースのコレクションをフィルタリングできます。

will filter out the collection of resources with the attribute **last\_name** that matches **Citizen**.

?last\_name=Citizen

will filter out the

?last\_name=Citizen&date\_of\_birth=1999-12-31

collection of resources with the attribute **last\_name** that

matches **Citizen** and **date\_of\_birth** that matches 31st of December 1999.

|  |
| --- |
| [R 23|1] |
| As a general guide, filtering SHOULD be done with case insensitivity. Whether you choose  to filter with case insensitivity or not SHALL be clearly documented. |

The equal **=** operator is the only supported operator when used in this technique. For other operators and conditions next section.

## Advanced filtering with LHS Operators

There are situations where simple filtering does not meet the needs and a more comprehensive approach is required. Use the reserved keyword filters to define a more complex filtering logic. The general pattern is

単純なフィルタリングではニーズを満たせず、より包括的なアプローチが必要な場合があります。 予約済みキーワード フィルターを使用して、より複雑なフィルター ロジックを定義します。 一般的なパターンは

**/path?property[operator]=value&property[operator]=value**

The **=** sign in this case is there to maintain URL query string compatibility with RFC 3986. However, the API service will use the operator inside the brackets for the actual comparison. A logical AND combines all query conditions.

この場合の = 記号は、RFC 3986 との URL クエリ文字列の互換性を維持するために存在します。ただし、API サービスは、実際の比較のために括弧内の演算子を使用します。 論理 AND は、すべてのクエリ条件を結合します。

The following operators are supported:

* + - * + **[gte]** Greater than or equalled to
        + **[egt]** Equalled to or greater than
        + **[gt]** Greater than
        + **[lt]** Less than
        + **[lte]** Less than or equalled to
        + **[elt]** Equalled to or less than
        + **[ne]** Not equalled

|  |
| --- |
| Example for filtering with LHS attributes:  /path?creation\_date[gt]=2020-11-30 |

## Rich Query with Lucene Syntax

|  |
| --- |
| [R 24|1] |
| If an application needs to support a richer search and filter capability that includes logical operators, fuzzy search, grouping, and so on, API MAY apply a query string according to lucene query syntax[8](#_bookmark33). In that case, the filtering and query parameters normally are  transmitted in the request body. |

8 <https://lucene.apache.org/core/2_9_4/queryparsersyntax.html>

## GraphQL

When API implementers would like to allow their clients rich flexibility to define response data sets that might include data from multiple APIs with rich filtering capability then a GraphQL query interface could be provided.

API の実装者が、豊富なフィルタリング機能を備えた複数の API からのデータを含む可能性のある応答データ セットをクライアントが柔軟に定義できるようにしたい場合は、GraphQL クエリ インターフェイスを提供できます。

GraphQL is a different architecture to RESTful APIs, is especially tailored to queries across multiple entities, and allows clients to specify exactly which data elements they would like in the response. If you find yourself building very complex RESTful queries then you should consider GraphQL as an alternative.

GraphQL は RESTful API とは異なるアーキテクチャであり、特に複数のエンティティにまたがるクエリに合わせて調整されており、クライアントは応答で必要なデータ要素を正確に指定できます。 非常に複雑な RESTful クエリを作成していることに気付いた場合は、代わりに GraphQL を検討する必要があります。

GraphQL is not discussed further in this RESTful API design guide.

GraphQL については、この RESTful API 設計ガイドではこれ以上説明しません。

#### Sorting

Providing data in specific order is often the requirement from client applications and hence it is important to provide the flexibility for clients to retrieve the data in the order they need it.

特定の順序でデータを提供することは、多くの場合、クライアント アプリケーションの要件であるため、クライアントが必要な順序でデータを取得できる柔軟性を提供することが重要です。

Sorting SHOULD be limited to specified fields. The sort direction MAY be omitted. The default sort direction is ascending. A colon is used to separate the field name and the sort direction. Multiple sort fields are separated by comma .

[R 25|1]

,

:

|  |  |
| --- | --- |
| **Query Parameter** | **Description** |
| *sort=name sort=name:asc* | Sort by the name field in ascending order. |
| *sort=name:desc* | Sort by the name field in descending order. |
| *sort=yearOfBirth,name:dec* | Sort by year of birth in ascending order. If two equal  years exist, sort the names by birth year in descending order. |

**Table 6: Sort examples**

#### API Responses and error handling

|  |
| --- |
| [R 26|1] |
| HTTP response codes SHALL be used.  The following table defines HTTP response codes supported by conformant APIs. The column **Response** indicates whether an additional error response payload is  RECOMMENDED to be returned as described in chapter [0](#_bookmark36). |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Status** | **Response** | **When to use** |
| 200 | OK | No | The request was successfully processed |
| 201 | Created | No | The resource was created. The **Location** HTTP response header SHALL be returned to indicate where the newly created resource is accessible. |
| 202 | Accepted | No | The request was accepted, and is processed asynchronously. |
| 204 | No content | No | The server successfully processed the request and is not returning any content. There is no need for the client to move to a different location. |
| 400 | Bad Request | Yes | The server cannot process the request (such as malformed request syntax, size too large, invalid request message framing, or deceptive request routing, invalid values in the request). For sensitive information, a code **404 Not found** MAY be returned instead. |
| 401 | Unauthorised | Yes | The request could not be authenticated. For sensitive information, a code **404 Not found** MAY be returned instead. |
| 403 | Forbidden | Yes | The request was authenticated but is not authorised to access the resource. For sensitive information, a code **404 Not found** MAY be returned instead. |
| 404 | Not found | Yes | The resource was not found. |
| 405 | Not Allowed |  | The method is not implemented for this resource. The response MAY include an **Allow** HTTP response header containing a list of valid methods for the resource. |
| 408 | Request Timeout | No | The request timed out before a response was received. A  **Retry-After** HTTP response header is RECOMMENDED to be returned. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Status** | **Response** | **When to use** |
| 415 | Unsupported Media Type | Yes | This status code indicates that the server refuses to accept the request because the content type specified in the request is not supported by the server |
| 422 | Unprocessable Entity |  | This status code indicates that the server understands the content type of the request entity, and the syntax of the request entity is correct, but it was unable to process the contained instructions. |
| 429 | Too Many Requests |  | There have been too many requests (by the consumer). A **Retry-After** HTTP response header is RECOMMENDED to be returned. A response body MAY be returned containing information about the reason for the response code. A possible reason may be if a quota of requests for the day / hour / month etc. was exceeded. |
| 500 | Internal Server error |  | An internal server error. The response body may contain error messages. The response body SHALL not reveal any server configuration information (e.g. version, paths, database used, etc.). |
| 501 | Method Not Implemented |  | It indicates that the request method is not supported by the server and cannot be handled for the requested resource. Implementing this response code allows a higher interoperability between API implementations based on the same specification, if a specific server does not support one of the specified methods (yet). A **Link** HTTP response header is RECOMMENDED to point to the specific documentation. |
| 503 | Service unavailable |  | It indicates that the service is unavailable (e.g. due to maintenance reasons). A **Retry-After** HTTP response  header is RECOMMENDED to be returned. |

**Table 7: HTTP response codes**

|  |  |
| --- | --- |
| [R 27|1] | |
| The following table defines which HTTP response codes SHALL be supported for a specific HTTP request method by conformant APIs. Column **Use** indicates how a conformant API supports the specified http response code:   * **M** the code SHALL be supported * **MA** SHALL be supported for requests where the response is handled asynchronous, for instance due to forwarding or processing time. In that case, a **Location** HTTP response header SHALL be gives that points to the respective resource. In addition, a **Retry- After** HTTP response header is RECOMMENDED to be returned. * **R** the code is recommended to be supported**.**   The default response code for a positive response is marked in **bold**. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP**  **Request method** | **Code** | **Status** | **Use** |
|  | **200** | **OK** | M |
| **GET** | |  |  |
|  | 202 | Accepted | MA |
|  | 400 | Bad Request | R |
|  | 401 | Unauthorised | M |
|  | 403 | Forbidden | M |
|  | 404 | Not found | M |
|  | 405 | Not Allowed | M |
|  | 408 | Request Timeout | R |
|  | 415 | Unsupported Media Type | M |
|  | 429 | Too Many Requests | R |
|  | 500 | Internal Server error | M |
|  | 503 | Service unavailable | R |
|  | **201** | **Created** | M |
| **POST** | |  |  |
|  | 202 | Accepted | MA |
|  | 400 | Bad Request | M |
|  | 401 | Unauthorised | M |
|  | 403 | Forbidden | M |
|  | 408 | Request Timeout | R |
|  | 415 | Unsupported Media Type | M |
|  | 422 | Unprocessable Entity | R |
|  | 429 | Too Many Requests | R |
|  | 500 | Internal Server error | M |
|  | 503 | Service unavailable | R |
|  | 202 | Accepted | MA |
| **PATCH** |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP**  **Request method** | **Code** | **Status** | **Use** |
|  | **204** | **No content** | M |
|  | 400 | Bad Request | M |
|  | 401 | Unauthorised | M |
|  | 403 | Forbidden | M |
|  | 404 | Not found | M |
|  | 405 | Not Allowed | M |
|  | 408 | Request timeout | R |
|  | 415 | Unsupported Media Type | M |
|  | 422 | Unprocessable Entity | M |
|  | 429 | Too Many Requests | R |
|  | 500 | Internal Server error | M |
|  | 503 | Service unavailable | R |
|  | 202 | Accepted | MA |
| **PUT** | |  |  |
|  | **204** | **No content** | M |
|  | 400 | Bad Request | M |
|  | 401 | Unauthorised | M |
|  | 403 | Forbidden | M |
|  | 404 | Not found | M |
|  | 405 | Not Allowed | M |
|  | 408 | Request Timeout | R |
|  | 415 | Unsupported Media Type | M |
|  | 422 | Unprocessable Entity | M |
|  | 429 | Too Many Requests | R |
|  | 500 | Internal Server error | M |
|  | 503 | Service unavailable | R |
|  | 202 | Accepted | MA |
| **DELETE** |  |  |  |
|  | **204** | **No content** | M |
|  | 400 | Bad Request | M |
|  | 401 | Unauthorised | M |
|  | 403 | Forbidden | M |
|  | 404 | Not found | M |
|  | 405 | Not Allowed | M |
|  | 408 | Request timeout | R |
|  | 415 | Unsupported Media Type | M |
|  | 422 | Unprocessable Entity | M |
|  | 429 | Too Many Requests | R |
|  | 500 | Internal Server error | M |
|  | 503 | Service unavailable | R |

#### Error Response Payload

For some errors, returning the HTTP status code is enough to convey the response. Additional error information can be supplemented in the response body. For example; HTTP 400 Bad request is considered too generic for a validation error and more information must be provided in the response body.

エラーによっては、HTTP ステータス コードを返すだけで十分に応答を伝えることができます。 追加のエラー情報は、応答本文で補足できます。 例えば; HTTP 400 の不正な要求は、検証エラーに対して一般的すぎると見なされ、応答本文で詳細情報を提供する必要があります。

|  |
| --- |
| [R 28|1] |
| An API SHALL implement an error response schema to allow a standardised error  handling. The response SHALL use the following JSON Schema. The JSON Schema MAY be extended. |
| {  "$schema": "https://json-schema.org/draft/2020-12/schema", "type": "object",  "properties": { "errors": {  "type": "array", "items": {  "type": "object", "properties": {  "id": { "type": "string",  "format": "uuid" }, "code": { "type": "string" },  "detail": { "type": "string" }, "source": {  "type": "object", "properties": {  "parameter": { "type": "string" },  "pointer": { "type": "string",  "format": "json-pointer" }  },  "unevaluatedProperties": false  },  "sourcePointer": { "type": "string",  "format":"json-pointer"}  },  "required": ["code", "detail"],  "patternProperties": { "^x-": true }, "unevaluatedProperties": false  },  "minItems": 1  }  },  "required": [ "errors" ],  "patternProperties": { "^x-": true }, "unevaluatedProperties": false  } |

The following definitions are applied:

|  |  |
| --- | --- |
| **Error response attributes** | **Description** |
| *id* | Identifier of the specific error |
| *detail* | A human-readable explanation specific to this occurrence of the problem. |
| *code* | An application-specific error code |
| *source* | An object containing computer processable information about the origin of the error. |
| *parameter* | The (query) parameter where the error was caused. |
| *pointer* | JSON Pointer [RFC6901] to the associated entity in the request  document [e.g. "/data" for a primary data object, or "/data/attributes/title" for a specific attribute]. |

**Table 8: Error response attributes**

|  |
| --- |
| Example for a 400 Bad Request error response:  {  "errors": [  {  "id": "86032cbe-a804-4c3b-86ce-ec3041e3effc", "code": "19283",  "detail": "Invalid value(s) in request input", "source": {  "parameter": "id"  }  }  ]  }  Example for a 503 Service unavailable error response:  Retry-After: Sat, 16 Apr 2022 15:00:00 GMT  {  "errors": [  {  "id": "45786a8f-452e-492f-a779-801b5d0bd0a7", "code": "19284", |

|  |
| --- |
| "detail": "The service is unavailable due to maintenance. Come back at 15:00 GMT.",  "source": {  "pointer": "#/resources/12345"  }  }  ]  } |

#### Design rule examples

|  |
| --- |
| Good examples |
| Get a list of voyages:  *GET* https://api.logistics.io/v1/transport/voyages  Filtering in a query:  *GET* https://api.logistics.io/v1/transport/voyages?departure\_location=AUBN E&date=2022-04-16  Get a single voyage:  GET https://api.logistics.io/v1/transport/voyages/N234  Create a new voyage:  POST https://api.logistics.io/v1/transport/voyages  {content body with voyage data in JSON format}  Update a voyage status:  PATCH https://api.logistics.io/v1/transport/voyages/N234/status  {content body status data in JSON format} |

# Well-documented APIs

### General considerations

|  |
| --- |
| [R 29|1] |
| The following rules are RECOMMENDED:   * The definitions in a conformant OpenAPI specification SHALL be considered as technical contracts between designers and developers and between consumers and providers. * Mock APIs SHOULD be created using the API description to allow early code integration for development. * The behaviour and intent of the API SHOULD be described with as much information as possible. * Operations SHOULD provide examples for request and response bodies. * Expected response codes and error messages SHOULD be provided in full. * Known issues or limitations SHOULD be clearly documented. * Expected performance, uptime and SLA/OLA SHOULD be clearly documented. * Although YAML is a supported file format of an OpenAPI specification, the JSON format SHOULD be used as the OpenAPI specification format. |

### API Versioning

#### Versioning Scheme

|  |
| --- |
| [R 30|1] |
| All APIs **SHALL** apply Semantic versioning 2.0.0[9](#_bookmark43): |
| MAJOR.MINOR.PATCH |
| The first version of an API SHALL start with a **MAJOR** version of 1.  Pre-release version[10](#_bookmark44) information and build metadata[11](#_bookmark45) version information SHALL NOT be used in API versioning. |

9 https://semver.org/spec/v2.0.0.html

10 https://semver.org/spec/v2.0.0.html#spec-item-9

11 https://semver.org/spec/v2.0.0.html#spec-item-10

Use the following guidelines when incrementing the API version number:

* **MAJOR** version when you make API changes that break backwards-compatibility,
* **MINOR** version when you add functionality in a backwards-compatible manner, and
* **PATCH** version when you make backwards-compatible bug fixes. A PATCH does not include new functionality.

#### URI Versioning

|  |
| --- |
| [R 31|1] |
| All APIs **SHALL** use URI versioning. They SHALL include the **MAJOR** version as part of the URI in the format of **'v{MAJOR}'** |
| Example: https://api.logistics.io/transport/v1/voyages |
| The minor and patch version SHALL NOT be used in the URI. |

#### Providing version information

APIs conforming to this technical specification are intended to be used with REST principles. Those mandate HATEOS (see chapter [4.3.2](#_bookmark55)) support. On major aspect is the self-descriptiveness of an API. Although a support of HATEOS is not required, providing basic metadata about the called API including version information is useful even in not RESTful scenarios.

この技術仕様に準拠する API は、REST 原則で使用することを目的としています。 HATEOS (第 4.3.2 章を参照) のサポートを義務付けています。 主な側面は、API の自己記述性です。 HATEOS のサポートは必須ではありませんが、呼び出された API に関する基本的なメタデータ (バージョン情報を含む) を提供すると、RESTful でないシナリオでも役立ちます。

|  |
| --- |
| [R 32|1] |
| A custom header named **API-Version** SHALL be added to any response of the API. It  SHALL be aligned with the URI version and SHALL state all three levels: |
| API-Version: 1.21.5 |

|  |
| --- |
| [R 33|1] |
| An API-Version custom header MAY be added to a request. If added, it SHALL only  contain the **MAJOR** version. |
| API-Version: 1 |

In order to easily provide information about an API in a standardised way, the following information can be retrieved from any conformant API:

API に関する情報を標準化された方法で簡単に提供するために、次の情報を適合 API から取得できます。

|  |
| --- |
| [R 34|1] |
| An API SHALL implement a response to a GET request to the base URI of the API. The  response SHALL use the following JSON Schema: |
| {  "$schema": "https://json-schema.org/draft/2020-12/schema", |

|  |
| --- |
| "type": "object", "properties": {  "title": { "type": "string" }, "version": {  "type": "string",  "pattern": "^\\d+(-.+)?\\.\\d+(-.+)?\\.\\d+(-.+)?$"  },  "status": { "type": "string",  "enum": ["DRAFT", "ACTIVE", "DEPRECATED", "RETIRED"]  },  "effective": { "type": "string",  "format": "date-time"  },  "specification": { "type": "string",  "format": "uri"  }  },  "required": [  "title", "version", "status", "effective", "specification"  ],  "$comment" : "Allow extensions to the API metadata", "patternProperties": {  "^x-": true  },  "unevaluatedProperties": false  } |

The following definitions are applied:

* **title**: The name of the API. It SHALL be identical to the API title defined in the OpenAPI specification
* **version**: The API version
* **status**: The operation status of the API. The following values are used:
  + **ACTIVE**: The API is in its productive phase. Maintenance or deprecation of specific services SHALL be indicated at the service level. The **effective** defines the moment in the past since when API is in its productive phase.
  + **DEPRECATED**: The complete API is going to its end-of-life phase. The **effective** defines the moment in the future when the API is intended to switch to **RETIRED**. The rules of deprecation (see chapter [4.2.5](#_bookmark50)) are applied additionally.
  + **RETIRED**: The complete API is to its end-of-life phase. The **effective** defines the moment in the past when the API was set to **RETIRED**. The rules of deprecation (see chapter [4.2.5](#_bookmark50)) are applied additionally.
* **effective**: The moment in time corresponding to the **status**.
* **specification**: A valid URI to the OpenAPI specification of the current API. This way the available services and data types become self-descriptive from their basic

structure. The OpenAPI specification SHOULD be public where possible and easily accessible to those that require it.

Additional metadata can be added to the response if required.

|  |
| --- |
| Example:  GET <https://api.uncefact.unece.org/v1/> HTTP 200 OK  content-type: application/json; charset=utf-8 API-Version: 1.0.0  {  "title": "UN/CEFACT Demo API", "version": "1.0.0",  "status": "ACTIVE",  "effective": "2022-06-02T23:00:00Z",  "specification": "https://service.unece.org/demo/demoAPI.json", "x-info" : "Additional information"  } |

During the draft, development or testing phase of an API sandbox environments are used to validate the intended functionality. For those kinds of APIs in development no additional state like **DRAFT** is provided.

|  |
| --- |
| [R 35|2] |
| APIs that are still in a **DRAFT** status SHOULD be placed in a sandbox environment. This  could be done by changing the basis URL accordingly. |
| Example for a productive base URL: https://api.uncefact.unece.org/v1/ Examples for a development base URL:  https://sandbox.api.uncefact.unece.org/v1/  https://staging.api.uncefact.unece.org/v1/ |

#### Robustness[12](#_bookmark49)

It is critical that APIs are developed with loose coupling in mind to ensure backwards compatibility for consumers.

|  |
| --- |
| [R 36|1] |
| Within a major release backward compatibility SHALL NOT be broken. |

12 https://en.wikipedia.org/wiki/Robustness\_principle

The following changes are deemed backwards compatible:

* Addition of a new optional field to a representation
* Addition of a new link to the **\_links** array of a representation
* Addition of a new endpoint to an API
* Additional support of a new media type (e.g. Accept: application/pdf)

The following changes are **NOT** deemed backwards compatible:

* Removal of fields from representations
* Changes of data types on fields (e.g. **string** to **boolean**)
* Changing semantic definitions
* Removal of endpoints or functions
* Removal of media type support

|  |
| --- |
| [R 37|1] |
| API clients and subscribers SHOULD be robust:   * Be conservative with API requests and data passed as input. * Be tolerant with unknown fields in the payload, but do not eliminate them from payload if needed for subsequent **PUT** requests. |

#### Deprecation and End of Life Policy

When designing new APIs one of the most important dates to consider is when the API will be retired. APIs are not intended to last forever. Some APIs are retired after a short time as they may be proving a use-case; others may be removed when better options are available for users.

新しい API を設計する際に考慮すべき最も重要な日付の 1 つは、API がいつ廃止されるかです。 API は永遠に続くものではありません。 一部の API は、ユースケースを証明している可能性があるため、短期間で廃止されます。 ユーザーがより良いオプションを利用できるようになった場合、他のものは削除される可能性があります。

The End-of-Life (EOL) policy determines the process that APIs go through to move through their workflow from **ACTIVE** to the **RETIRED** state. The EOL policy is designed to ensure a consistent and reasonable transition period for API customers who need to migrate from the old API version to the new API version while enabling a healthy process to retire technical debt.

End-of-Life (EOL) ポリシーは、API がワークフローを通じて ACTIVE から RETIRED 状態に移行するプロセスを決定します。 EOL ポリシーは、古い API バージョンから新しい API バージョンに移行する必要がある API のお客様に対して、一貫した合理的な移行期間を確保すると同時に、技術的負債を解消するための健全なプロセスを可能にするように設計されています。

##### Major API Version EOL

Major API versions **MAY** be backwards compatible with preceding major versions. The following rules apply when retiring a major API version.

メジャー API バージョンは、以前のメジャー バージョンと下位互換性がある場合があります。 メジャー API バージョンを廃止する場合、次のルールが適用されます。

|  |
| --- |
| [R 38|1] |
| An API SHALL NOT be set to **DEPRECATED** until a replacement service is running with  status **ACTIVE**. |

|  |
| --- |
| The root service of the API SHALL provide the **Deprecation Header Field**[13](#_bookmark51) and the  **Sunset HTTP Response Header Field**[**14**](#_bookmark52).  A **Link header** SHALL be added in combination with the **Deprecation header**. It SHALL provide a link to the documentation. A second **Link header** SHALL be added linking to the replacement version of the API. |

Additionally, the following thoughts should be considered:

1. A minimum transition period of 60 days should be planned to give users adequate notice to migrate.
2. Deprecation of API versions with external users should be considered on a case-by- case basis and may require additional deprecation time and/or constraints to minimise impact to users.
3. If a versioned API is **ACTIVE** or **DEPRECATED** state has no registered users, it may move to the **RETIRED** state immediately.

|  |
| --- |
| [R 39|1] |
| Deprecated endpoints SHALL be documented in the OpenAPI specification using the  **DEPRECATED** property introduces since OpenAPI 3.0.0.  Deprecated endpoints SHOULD provide the Deprecation Header Field and the Sunset HTTP Response Header Field.  A Link header SHALL be added in combination with the Deprecation header. It SHALL provide a link to the documentation.  Where possible, communication SHOULD be sent to consumers of deprecated endpoints. |

|  |
| --- |
| [R 40|1] |
| The introduction of a major version SHOULD be avoided, whenever possible. This MAY be achieved as follows:   * Create a new service endpoint, if the process is changed.   Duplicate and Deprecate: add a **Deprecation Header** to the old service including a **Link Header** to documentation and to the new service. Eventually add a **Sunset Header**.   * Create a new resource (a variant of the old) in addition to the old. |

##### Minor API Version EOL

Due to the specified URL versioning the URL does not change if the minor version of an API changes. Minor API versions are backwards compatible with preceding minor versions within the same major version.

13 https://tools.ietf.org/html/draft-dalal-deprecation-header-02

14 https://tools.ietf.org/html/rfc8594#section-3

Therefore, the status before, during or after a minor API version update does not change. The change should have no impact on existing subscribers so there is no need to transition through a **DEPRECATED** state to facilitate client migration.

|  |
| --- |
| [R 41|2] |
| New resources or service endpoints can be added during a minor release. In order to support  the implementation of those new services a sandbox environment SHOULD be provided to the interested or affected consumers. |

|  |
| --- |
| [R 42|1] |
| It is RECOMMENDED that no more than 3 parallel MAJOR versions are available.  Implementers of the API SHALL NOT be more than 1 major version behind the latest version. |
| Example  Version 1 is **RETIRED** Version 2 is **DEPRECATED** Version 3 is **ACTIVE** |

### Hypermedia

#### Hypermedia - Linked Data

An API becomes RESTful by meeting the requirements of the REST principles. A key principle is the discoverability of the API. Ideally, this is achieved by an API being completely self-describing. According to the inventor of REST, Roy Fielding[15](#_bookmark56), the use of hypermedia is a prerequisite for designing a RESTful API.

API は、REST 原則の要件を満たすことで RESTful になります。 重要な原則は、API の発見可能性です。 理想的には、これは完全に自己記述的な API によって実現されます。 REST の発明者である Roy Fielding15 によると、ハイパーメディアの使用は RESTful API を設計するための前提条件です。

Hypermedia means that links are provided together with the response payload. They inform the consumers what options are available according to their original request. Though simple in concept hypermedia links in APIs, allow consumers to locate resource without the need to have an upfront understanding of the resource and its relationship.

ハイパーメディアとは、リンクが応答ペイロードとともに提供されることを意味します。 それらは、元の要求に従って利用可能なオプションを消費者に通知します。 API のハイパーメディア リンクの概念は単純ですが、消費者は、リソースとその関係を事前に理解する必要なく、リソースを見つけることができます。

This is similar to the navigation of a web page. The user is not expected to know the structure of the web page prior to visiting. They can simply browse to the home page and the navigation lets them browse the site as required.

これは、Web ページのナビゲーションに似ています。 ユーザーは、訪問する前に Web ページの構造を知っている必要はありません。 ホームページを参照するだけで、必要に応じてナビゲーションでサイトを参照できます。

APIs that do not provide links are more difficult to use and expect the consumer to refer to the documentation.

リンクを提供しない API は使用が難しく、消費者がドキュメントを参照することを期待します。

#### HATEOAS

*Hypermedia As The Engine Of Application State* is the concept of representing allowable actions as hyperlinks associated with resource.

*Hypermedia As The Engine Of Application State（HATEOS）*は、許容可能なアクションをリソースに関連付けられたハイパーリンクとして表すという概念です。

Similar to Hypermedia Linked Data concept the links defined in the response data represents state transitions that are available from that current state to adjacent states.

Hypermedia Linked Data の概念と同様に、応答データで定義されたリンクは、現在の状態から隣接する状態への状態遷移を表します。

|  |
| --- |
| Example:  HEAD /v1/accounts/4711 HTTP/1.1 200 OK  Link: <https://api.unece.org/v1/accounts/4711>; rel="self",  <https://api.unece.org/v1/accounts/4711/deposit>; rel="deposit",  <https://api.unece.org/v1/accounts/4711/withdraw>; rel="withdraw",  <https://api.unece.org/v1/accounts/4711/transfer>; rel="transfer" |

If the same account is overdrawn, the only allowed action could be to deposit:

同じ口座が引き落とされた場合、許可される唯一のアクションは入金することです。

15 https://[www.ics.uci.edu/~fielding/pubs/dissertation/fielding\_dissertation\_2up.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf)

|  |
| --- |
| Example:  GET /v1/accounts/4711 HTTP/1.1 200 OK  Link: <https://api.unece.org/v1/accounts/4711>; rel="self",  <https://api.unece.org/v1/accounts/4711/deposit>; rel="deposit" Content-Type: application/json  Content-Length: ...  {  "accountId": "4711", "balance": {  "currency": "EUR", "value": -25  }  } |

#### Hypermedia Compliant API

In APIs, request methods such as *DELETE*, *PATCH*, *POST* and *PUT* initiate a transition in the state of a resource. A *GET* request never changes the state of the resource that is retrieved.

API では、DELETE、PATCH、POST、PUT などのリクエスト メソッドがリソースの状態遷移を開始します。 GET 要求は、取得されるリソースの状態を決して変更しません。

|  |
| --- |
| [R 43|1] |
| In order to provide a better experience for API consumers, APIs SHOULD provide a list of state transitions that are available for each resource. As possible values for link relation types the official IANA registry list[16](#_bookmark58) SHALL be used. It MAY be extended. Any extension  SHALL be documented in the API specification. |

An example of an API that exposes a set of operations to manage a user account lifecycle and implements the HATEOAS interface constraint is as follows:

ユーザー アカウントのライフサイクルを管理する一連の操作を公開し、HATEOAS インターフェイス制約を実装する API の例は次のとおりです

A client starts their interaction with a service through the URI */users*. This fixed URI supports both *GET* and *POST* operations. The client decides to do a *POST* operation to create a user in the system.

クライアントは、URI /users を介してサービスとの対話を開始します。 この固定 URI は、GET 操作と POST 操作の両方をサポートします。 クライアントは、システムにユーザーを作成するために POST 操作を実行することを決定します。

|  |
| --- |
| Request  POST https://api.unece.org/v1/v1/users  {  "firstName": "John",  "lastName" : "Smith", |

16 https://[www.iana.org/assignments/link-relations/link-relations.xhtml](http://www.iana.org/assignments/link-relations/link-relations.xhtml)

|  |
| --- |
| ...  } |

The API creates a new user from the input and returns the following links to the client in the response.

API は入力から新しいユーザーを作成し、応答で次のリンクをクライアントに返します。

* A link to the created resource in the *Location* header (to comply with the 201 response spec)
* A link to retrieve the complete representation of the user (a.k.a. *self*-link) (*GET*).
* A link to update the user (*PUT*).
* A link to partially update the user (*PATCH*).
* A link to delete the user (*DELETE*).

|  |
| --- |
| HTTP/1.1 201 CREATED  Location: https://api.unece.org/v1/users/JFWXHGUV7VI  Link: <https://api.unece.org/v1/users/JFWXHGUV7VI>, rel="self",  <https://api.unece.org/v1/users/JFWXHGUV7VI>, rel="delete",  <https://api.unece.org/v1/users/JFWXHGUV7VI>, rel="replace",  <https://api.unece.org/v1/users/JFWXHGUV7VI>, rel="edit" |

A client can store these links in its database for later use.

クライアントは、これらのリンクを後で使用するためにデータベースに保存できます。

In summary:

要約すれば：

* There is a well-defined index or navigation entry point for every API, which a client navigates to in order to access all other resources.

• クライアントが他のすべてのリソースにアクセスするためにナビゲートするすべての API には、明確に定義されたインデックスまたはナビゲーション エントリ ポイントがあります。

* The client does not need to build the logic of composing URIs to execute different requests or code any kind of business rule by looking into the response details that may be associated with the URIs and state changes.

• クライアントは、さまざまな要求を実行するために URI を構成するロジックを構築したり、URI や状態の変化に関連付けられる可能性がある応答の詳細を調べて、あらゆる種類のビジネス ルールをコーディングしたりする必要はありません。

* The client acknowledges the fact that the process of creating URIs belongs to the server.

• クライアントは、URI を作成するプロセスがサーバーに属するという事実を認識します。

* Client treats URIs as opaque identifiers.

• クライアントは URI を不透明な識別子として扱います。

* APIs using hypermedia in representations could be extended seamlessly. As new methods are, introduced responses could be extended with relevant HATEOAS links. These way clients could take advantage of the functionality in incremental fashion. For example, if the API starts supporting a new *PATCH* operation then clients could use it to do partial updates.

• 表現にハイパーメディアを使用する API は、シームレスに拡張できます。 新しい方法と同様に、導入された応答は、関連する HATEOAS リンクで拡張できます。 このようにして、クライアントは段階的に機能を利用できます。 たとえば、API が新しい PATCH 操作のサポートを開始した場合、クライアントはそれを使用して部分的な更新を行うことができます。

The mere presence of links does not decouple a client from having to learn the data required making requests for a transition and all associated link semantics particularly for *POST*/*PUT*/*PATCH* operations.

リンクが存在するだけでは、クライアントは、特に POST/PUT/PATCH 操作の場合に、遷移の要求と関連するすべてのリンク セマンティクスを作成するために必要なデータを学習する必要がなくなるわけではありません。

# API Security

|  |
| --- |
| [R 44|1] |
| All API endpoints SHALL be secured. HTTPS SHALL be used. The OAUTH2 security scheme is RECOMMENDED. Other security schemes MAY be used.  The receivers’ endpoints of subscription callbacks MAY be designed with different security measures like those described in chapter [6.3](#_bookmark63).  The following aspects of API security are RECOMMENDED to be implemented: |

##### Rate Limiting

Rate limiting and throttling policies are introduced to prevent abuse of your API. Appropriate alerts should be implemented and respond with informative errors when thresholds are nearing or have been exceeded.

API の悪用を防ぐために、レート制限とスロットリング ポリシーが導入されています。 適切なアラートを実装し、しきい値に近づいているか、しきい値を超えたときに有益なエラーで応答する必要があります。

See [https://greenbytes.de/tech/webdav/draft-](https://greenbytes.de/tech/webdav/draft-ietf-httpapi-ratelimit-headers-latest.html) [ietf-httpapi-ratelimit-headers-latest.html](https://greenbytes.de/tech/webdav/draft-ietf-httpapi-ratelimit-headers-latest.html) for implementation details.

##### Error Handling

When your application displays error messages, it should not expose information that could be used to attack your system. You should establish the following controls when providing error messages:

アプリケーションがエラー メッセージを表示する場合、システムを攻撃するために使用される可能性のある情報を公開すべきではありません。 エラー メッセージを提供するときは、次の制御を確立する必要があります。

* Your API MUST mask any system related errors behind standard HTTP status responses and error messages e.g. do not expose system level information in your error response

• API は、標準の HTTP ステータス応答とエラー メッセージの背後にあるシステム関連のエラーをマスクする必要があります。 エラー応答でシステム レベルの情報を公開しないでください。

* Your API MUST NOT pass technical details (e.g. call stacks or other internal hints) to the client

• API は、技術的な詳細 (コール スタックやその他の内部ヒントなど) をクライアントに渡してはなりません (MUST NOT)。

##### Audit Logs

An important aspect of security is to be notified when something wrong occurs, and to be able to investigate it. It is RECOMMENDED to implement logging.

セキュリティの重要な側面は、何か問題が発生したときに通知を受け、それを調査できるようにすることです。 ロギングを実装することをお勧めします。

* Write audit logs before and after security related events which can trigger the alerts

• アラートをトリガーできるセキュリティ関連イベントの前後に監査ログを書き込む

* Sanitizing the log data to prevent log injection attacks

• ログ インジェクション攻撃を防ぐためのログ データのサニタイズ

##### Input Validation

Input validation is performed to ensure only properly formed data is received by your system, this helps to prevent malicious attacks

入力検証が実行され、適切な形式のデータのみがシステムによって受信されるようになり、悪意のある攻撃を防ぐのに役立ちます。

* Input validation should happen as early as possible, preferably as soon as the data is received from the external party

• 入力の検証はできるだけ早く、できれば外部からデータを受け取ったらすぐに行う必要があります。

* Define an appropriate request size limit and reject requests exceeding the limit

• 適切なリクエスト サイズ制限を定義し、制限を超えるリクエストを拒否する。

* Validate input: e.g. length / range / format and type

• 入力（長さ / 範囲 / フォーマットとタイプ）を検証します。

* Consider logging input validation failures. Assume that someone who is performing hundreds of failed input validations per second has a malicious intent.

• 入力検証の失敗をログに記録することを検討してください。 1 秒間に何百回も失敗した入力検証を実行している誰かが悪意を持っていると仮定します。

* Constrain string inputs with regular expression where appropriate

• 必要に応じて正規表現を使用して文字列入力を制限します。

##### Content Type Validation

Honour the specified content-type. Reject requests containing unexpected or missing content type headers with HTTP response status *415 Unsupported Media Type*.

指定されたコンテンツ タイプを尊重します。 HTTP 応答ステータス 415 Unsupported Media Type で、予期しないコンテンツ タイプ ヘッダーまたは不足しているコンテンツ タイプ ヘッダーを含む要求を拒否します。

##### Gateway Security Features

It is RECOMMENDED to use the security policy features available in the gateway rather than to implement the policies in your back-end API.

バックエンド API にポリシーを実装するのではなく、ゲートウェイで利用可能なセキュリティ ポリシー機能を使用することをお勧めします。

# Event driven data exchange

Classic B2B data exchange scenarios reach their limits especially when it comes to processing real-time data.

従来の B2B データ交換シナリオは、特にリアルタイム データの処理に関しては限界に達します。

For example, one of the most important pieces of information in just-in-time production is the expected arrival time (ETA) at the factory.

たとえば、ジャスト イン タイム生産で最も重要な情報の 1 つは、工場への到着予定時刻 (ETA) です。

PULL scenarios are often implemented, where the consumer periodically asks the data sender for the current status of the delivery. Alternatively, the carrier sends a status message at regular but short intervals on the current status of the delivery with detailed information for each consignment item.

多くの場合、PULL シナリオが実装されます。このシナリオでは、コンシューマーが定期的にデータ送信者に配信の現在のステータスを尋ねます。 別の方法として、配送業者は定期的かつ短い間隔で、現在の配送状況に関するステータス メッセージを送信し、各貨物の詳細情報を提供します。

This leads to tremendous amounts of data, so that in practice the minimum interval of such updates is about 15 minutes. Thus, in such scenarios, real-time information is a long way off.

これにより膨大な量のデータが生成されるため、実際にはこのような更新の最小間隔は約 15 分です。 したがって、このようなシナリオでは、リアルタイムの情報は遠く離れています。

One approach to solving this problem is now to define events when they occur and exchange the data instead of constantly exchanging (less relevant) information.

この問題を解決するための 1 つのアプローチは、(関連性の低い) 情報を絶えず交換する代わりに、イベントが発生したときにイベントを定義し、データを交換することです。

This could be the case, for example, if a geo-fence is crossed, a temperature is exceeded or not reached, or a clearance takes longer than it is intended. In the consumer space, such scenarios are already familiar, for example, when the buyer of an online delivery is notified that the package is only 10 stops away from delivery.

これは、たとえば、ジオフェンスを越えた場合、温度を超えた場合、または温度に達しなかった場合、またはクリアランスに意図したよりも時間がかかった場合などに当てはまります。 消費者スペースでは、このようなシナリオはすでによく知られています。たとえば、オンライン配送の購入者が、パッケージが配送まであと 10 か所しかないことを通知された場合です。

Push-Method

Subscription

Geo-fence reached!

Feedback

Server

Client

**Figure 1: Event driven data exchange – pull versus push method**

Pull-Method

Geo-fence reached?

No

Geo-fence reached?

Yes

Geo-fence reached?

No

Server

Client

### Callbacks

In OpenAPI, you can define callbacks. Those are asynchronous requests to a consumer specified URL that are called in response to a specific event. An example is that a carrier is informed if a specific vessel approaches a port.

OpenAPI では、コールバックを定義できます。 これらは、特定のイベントに応答して呼び出される、消費者が指定した URL への非同期要求です。 一例として、特定の船舶が港に接近した場合に運送業者に通知することが挙げられます。

In order to be able to receive this information, the receiver first needs to subscribe to this event information in the API. When subscribing, he may pass filter criteria that define the conditions under which the consumer will be informed. Examples are a specific journey where the consumer wants to get informed if it approaches a specific port.

この情報を受信できるようにするために、受信側はまず API でこのイベント情報をサブスクライブする必要があります。 サブスクライブするとき、彼は、消費者に通知される条件を定義するフィルター基準を渡すことができます。 例としては、特定の港に近づいた場合に消費者が通知を受け取りたいという特定のジャーニーがあります。

The basic principle is that a consumer subscribes for an event, supplies a (callback) URL and stands by for incoming HTTP requests to that URL.

基本的な原則は、コンシューマーがイベントをサブスクライブし、(コールバック) URL を提供し、その URL への受信 HTTP 要求を待機することです。

### Webhooks

Since OpenAPI 3.1, webhooks are supported as well. The main difference between callbacks and webhooks is that webhooks are synchronous to the process flow handled by the APIs.

OpenAPI 3.1 以降、Webhook もサポートされています。 コールバックと Webhook の主な違いは、Webhook が API によって処理されるプロセス フローと同期していることです。

This means that a consumer can directly hook into the process and thus, if necessary, change the processed information before it is further processed. A webhook is used to extend the functionality of the API.

つまり、コンシューマーはプロセスに直接フックできるため、必要に応じて、処理された情報をさらに処理する前に変更できます。 Webhook は、API の機能を拡張するために使用されます。

A webhook defines a clear point in the process where the consumer is enabled to react on, for example based on some external event.

Webhook は、たとえば何らかの外部イベントに基づいて、コンシューマーが反応できるプロセス内の明確なポイントを定義します。

An example is if you want to react immediately on any incoming order/payment etc. The payload itself is given with the webhook and often allows modifications. Examples are the option to link to a GitHub push event or to define a plugin for the WordPress content management system. The latter modifies for example the displayed HTML page directly by adding new functionalities like images, tables, videos or similar to the HTML page. Such modifications would not be possible with an asynchronous callback.

例としては、注文や支払いの受信などにすぐに対応したい場合があります。ペイロード自体は Webhook で指定され、多くの場合変更が可能です。 例としては、GitHub プッシュ イベントにリンクするオプションや、WordPress コンテンツ管理システムのプラグインを定義するオプションがあります。 後者は、たとえば、画像、表、ビデオなどの新しい機能を HTML ページに追加することにより、表示された HTML ページを直接変更します。 このような変更は、非同期コールバックでは不可能です。

### Security guideline for callbacks (informative)

Since webhooks work synchronously, the same security rules apply to them as to the entire API. In contrast, the call direction is reversed for asynchronous callbacks. This makes it important to ensure that the callback URL is only called from the authorized API.

Webhook は同期的に機能するため、API 全体と同じセキュリティ ルールが適用されます。 対照的に、非同期コールバックの呼び出し方向は逆になります。 これにより、コールバック URL が許可された API からのみ呼び出されるようにすることが重要になります。

The following rules are based on the current approach of the DCSA. They are in the trial phase at the time of publication of this document. As soon as sufficient practicality has been demonstrated, this specification will be updated accordingly. Against this background, the following rules are purely informative and not normative.

次の規則は、DCSA の現在のアプローチに基づいています。 このドキュメントの公開時点では、それらは試行段階にあります。 十分な実用性が実証され次第、この仕様は適宜更新されます。 このような背景に対して、以下の規則は純粋に参考情報であり、規範的ではありません。

|  |
| --- |
| [R 45|1+Inf] |
| All event subscriptions SHALL be secured via a Shared Secret that is used to sign every callback message as described in this section. The secret SHALL be provided BASE64 encoded. The provider SHALL NOT expose the **secret** in any endpoint. It is write-only. The provider SHALL assure that the secret fulfils the security requirements of the applied  algorithm. |

|  |
| --- |
| [R 46|2+Inf] |
| A sha256 signature SHALL be used computed as an HMAC-SHA246 over the request body[17](#_bookmark65). The subscriber provided Shared Secret SHALL be of at least 32-byte length. It SHOULD not be longer than 64 byte, as longer keys do not provide additional security to that algorithm.  To improve security, it is RECOMMENDED to update the **secret** (and together with it the  **callbackURL**) on a regular basis. |

|  |
| --- |
| [R 47|1+Inf] |
| The publisher API SHALL provide the following endpoints for subscriptions:   * **POST …/subscriptions** to create a new subscription * **GET …/subscriptions** to list all subscriptions the subscriber has access to * **GET …/subscriptions/{subscriptionId}** to get details about a specific subscription * **PUT …/subscriptions/{subscriptionId}** to update a specific subscription * **PUT …/subscriptions/{subscriptionId}/secret** to update the secret of a specific subscription * **DELETE …/subscriptions/{subscriptionId}** to cancel a specific subscription |

#### Subscription setup (informative)

The setup of a subscription follows the following steps:

サブスクリプションのセットアップは、次の手順に従います。

1. The subscriber defines a Shared Secret and registers with the **secret** and a **callbackURL** in the publisher's system. It is recommended to use a not-easy-to-guess[18](#_bookmark66) callback URL and to update it when the secret is changed.

1. サブスクライバーは共有シークレットを定義し、公開者のシステムでシークレットと callbackURL に登録します。 推測しにくいコールバック URL を使用し、シークレットが変更されたときに更新することをお勧めします。

1. The publisher confirms the subscription and returns the **subscriptionId** to the subscriber.

2. パブリッシャーはサブスクリプションを確認し、subscriptionId をサブスクライバーに返します。

1. The subscriber records the **subscriptionId** associated with the shared **secret**.

3. サブスクライバーは、共有シークレットに関連付けられた subscriptionId を記録します。

|  |
| --- |
| Example for a subscription setup   1. Initiating the subscription   POST https://api.unece.org/v1/events/subscribe Content-Type: application/json  Content-Length: ...  {  "callbackURL" : "https://callback.example.com/callback/Ujh4kkQ9A", "secret":  "MDEyMzQ1Njc4OWFiY2RlZjAxMjM0NTY3ODlhYmNkZWYwMTIzNDU2Nzg5YWJjZGVmMDEyMzQz NjU3ODlhYmNkZQ",  ... additional filter parameters etc. ...  }   1. a Confirmation of the publisher if the **callbackURL** is valid   Remark: As the subscription is not setup yet, not additional headers are provided.  HEAD https://callback.example.com/callback/Ujh4kkQ9A   * 1. b Response of the subscriber that the **callbackURL** is valid   HTTP/1.1 204 No Content   * 1. Response from the publisher   HTTP/1.1 201 Created  Content-Type: application/json Content-Length: ...  {  "subscriptionId": "936DA01F-9ABD-4D9D-80C7-02AF85C822A8",  "callbackURL": "https://callback.example.com/callback/Ujh4kkQ9A",  ... additional optional content ...  } |

#### Performing a subscription call (informative)

A subscription call follows the following steps:

サブスクリプション コールは、次の手順に従います。

1. The publisher SHALL perform a **POST** to the **callbackURL** of the subscriber.

1. パブリッシャーは、サブスクライバーの callbackURL に対して POST を実行する必要があります。

* + A **Subscription-ID** HTTP header containing the **subscriptionId** is added.
  + A **Notification-Signature** HTTP header containing the computed signature of the request body is added.
  + The request-body is sent using the **application/json** format.

1. The subscriber SHALL validate the **POST** request. It SHOULD be done in the following order. If any of the validation steps fail, the message SHALL be rejected.

2. サブスクライバーは、POST 要求を検証する必要があります。 次の順序で実行する必要があります。 検証ステップのいずれかが失敗した場合、メッセージは拒否される必要があります。

* + It is RECOMMENDED to start message parsing only if all of the validation steps are performed without an error.
  + The **Notification-Signature** HTTP header MUST be provided.
  + The **Subscription-ID** HTTP header MUST be included. It MUST be a GUID.
  + Additional provided custom information is RECOMMENDED to be validated. (e.g. in the **callbackURL**)
  + The subscriber uses the stored Shared Secret to compute the signature of the request body. The signature SHALL equal the provided signature.
  + In case the callback was performed due to a a subscription of an event, the occurrence time of the event MUST be in the past. It MAY be a few seconds in the future to account for minor time synchronization issues.

1. A successful callback is responded by the **204 No Content** response code.

3. 成功したコールバックは、204 No Content 応答コードによって応答されます。

|  |
| --- |
| Example for a subscription call using the secret from the example above  POST https://callback.example.com/callback/Ujh4kkQ9A Subscription-ID: 936DA01F-9ABD-4D9D-80C7-02AF85C822A8  Notification-Signature: sha256=66c2912069e6c9563d66fee4674cd23dd9dd00e6c08c985e964b11f92f477e48 Content-Type: application/json  Content-Length: ...  {  "id": "84db923d-2a19-4eb0-beb5-446c1ec57d34", "occurrenceDateTime": "2022-04-16T16:40:00+01:00", "typeCode": "ARRIVAL",  "shipmentId": "123e4567-e89b-12d3-a456-426614174000"  }  Response  HTTP/1.1 204 No Content |

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

# Appendix B: Naming and Design Rules List (normative)

|  |  |
| --- | --- |
| 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.   Inf. Rules that are informative only. If a different implementation is chosen this does  not have any impact on the conformance of the implementation towards this specification. |
| [R 2|1] | All API specifications based on this OpenAPI Naming and Design Rules technical specification SHALL be compliant to the OpenAPI 3.1.x specification. |
| [R 3|1] | An API specification-claiming conformance to this specification SHALL define schema components as described in the JSON Schema Naming and Design Rules  Technical Specification. |
| [R 4|1] | Request body content and Response content used to transfer structured data information SHALL use the **application/json** media type for JavaScript Object Notation (JSON). This rule MAY only be deviated from, if the API implements a conversion service from or to JSON in another media type.  Additional media types (e.g. **text/xml**) to transfer structured data information MAY be used. If non-structured information is transferred any valid media type MAY be used. |
| [R 5|1] | Encoding SHALL be UTF-8. |
| [R 6|2] | The structure of the paths defined within APIs SHOULD be meaningful to the consumers. Paths SHOULD follow a predictable, hierarchical structure to enhance understandability and therefore usability. |
| [R 7|1] | The API URLs SHOULD follow the standard naming convention as described below:  **https://{env}.api.{dnsdomain}/v{m}/{service}/{resource}/{id}/{sub- resource}?{query}**  The components are described as follows. If a rule is mandatory for a specific component of the URL is SHALL be applied to any conformant API specification, even if the basic URL structure is different from the one described above (e.g. if **api** is not used as a prefix to the **dnsdomain**).   * https:// SHALL be used as the web protocol. * {env} indicates the environment (e.g. **test**, **sandbox** or **dev**) and is usually omitted for production environment. * {dnsdomain} is the DNS domain of the API implementer (e.g. **unece.org**) * {service} is a logical grouping of API functions that represent a business service domain (e.g. transport). The {service} component is optional. * v{m} is the major version number of the API specification. This component SHALL be stated in the URL. It MAY be provided at a different place in the URL (e.g. as a prefix to the domain). * {resource} is the plural noun representing an API resource (e.g. **consignments**) * {id} is the unique identifier for the resource defined as a path parameter. Path parameters SHALL be used to identify a resource. This component is not part of the path if an operation is performed on a collection of the resource. * {sub-resource} is an optional sub-resource. Only used when there are contained collections or actions on a main resource (e.g. **consignmentItem**). |

|  |  |
| --- | --- |
|  | * {query} is a list of additional parameters like filters that determine the results of a search (e.g. **consignments?loadingPort=AUSYD**). |
| [R 8|1] | The total number of characters in the URL, including the path and the query, SHALL NOT exceed 2000 characters in length including any formatting codes such as  commas, underscores, question marks, hyphens, plus or slashes. |
| [R 9|1] | Endpoints SHALL NOT be actions. Services and resources SHALL consist of nouns. HTTP verbs SHALL be used for actions. |
| [R 10|1] | Kebab-case SHALL be used in services. |
| [R 11|1] | Lower camelCase SHALL be used in resources, path parameters and query parameters. |
| [R 12|1] | Path parameters and query parameters with a relation to property names SHALL be consistent with property names. |
| [R 13|1] | Query parameters SHALL be URL safe. |
| [R 14|1] | Resource names SHALL be pluralised. Resource names SHOULD be consistent with schemas. If a schema is defined in singular, nevertheless the resource SHALL be pluralized. If the plural of a resource is non-standard, you MAY choose a more  appropriate noun in its plural form. |
| [R 15|1] | Query parameters SHALL use ISO8601 compliant date and time representations that are defined in **UNTDID 2379 json** as defined in the JSON schema NDR technical specification. To represent a specific date, time or date-time the format SHALL  comply with the JSON schema definition for date, time or date-time. |
| [R 16|1] | A prerequisite for an OpenAPI specification and its implementation to be fully compliant with this NDR TS is the use of UN/CEFACT semantics and UN/CEFACT syntax (e.g. UN/CEFACT XML, UN/CEFACT JSON Schema, and UN/CEFACT Vocabulary).  An OpenAPI specification that does not use UN/CEFACT syntax or UN/CEFACT semantics may still be conformant to this NDR TS if it meets the criteria specified in [R 1|1]. |
| [R 17|1] | Endpoints are RECOMMENDED to support CRUD operations. (Create, Read, Update, Delete). If an endpoint is not intended to support e.g. a delete operation, it  SHALL return the HTTP response codes as defined in chapter [3.2.10.](#_bookmark35) |
| [R 18|1] | APIs SHALL adhere to the idempotency of operations specified in Table 4. |
| [R 19|1] | APIs SHOULD implement the Idempotency-Key HTTP header field and the corresponding implementation advice in order to make non-idempotent operations like POST and PATCH fault-tolerant. |
| [R 20|1] | If pagination is used in an API, keyset-based pagination (cursor-pagination) SHALL be used. This means that the consumer cannot request a specific page, instead the consumer has to select a page-link provided by the server. The server SHALL provide links in the HTTP response header to the previous and next page and SHOULD provide links to the first and last page. More links MAY be provided.  The cursor-value is a string, created by the server using whatever method it likes. It identifies a point in a list of results for a query containing filters and sorting parameters for a specific moment in time. Therefore, it divides the list into those that  fall before the cursor and those that fall after the cursor. There may optionally be one result that falls "on" the cursor. |
| [R 21|1] | GET requests on collection results SHOULD implement pagination. The default and maximum page size SHOULD be 100, if not specified on the endpoint. If SHOULD be smaller, if the resulting page load is large. The default page size MAY be changed per endpoint. A consumer SHOULD be able to override the default page size.  If the filter, sorting and/or page size used is changed when getting a result, the pagination SHALL BE reset to the first page.  The query parameters described in the following table SHALL be used, rules SHALL be applied. |

|  |  |  |  |
| --- | --- | --- | --- |
| [R 22|1] | Sorting and filtering SHALL be done using query parameters. Using a path parameter is only allowed to identify a specific resource. | | |
| [R 23|1] | As a general guide, filtering SHOULD be done with case insensitivity. Whether you choose to filter with case insensitivity or not SHALL be clearly documented. | | |
| [R 24|1] | If an application needs to support a richer search and filter capability that includes logical operators, fuzzy search, grouping, and so on, API MAY apply a query string according to lucene query syntax . In that case, the filtering and query parameters  normally are transmitted in the request body. | | |
| [R 25|1] | Sorting SHOULD be limited to specified fields. The sort direction MAY be omitted. The default sort direction is ascending. A colon : is used to separate the field name | | |
| and the sort direction. Multiple sort fields are separated by comma | , | . |
| [R 26|1] | HTTP response codes SHALL be used.  Table 6 defines HTTP response codes supported by conformant APIs. The column  Response indicates whether an additional error response payload is RECOMMENDED to be returned as described in chapter 3.2.11. | | |
| [R 27|1] | Table 7 defines which HTTP response codes SHALL be supported for a specific HTTP request method by conformant APIs. Column Use indicates how a conformant API supports the specified http response code:   * **M** the code SHALL be supported * **MA** SHALL be supported for requests where the response is handled asynchronous, for instance due to forwarding or processing time. In that case, a **Location** HTTP response header SHALL be gives that points to the respective resource. In addition, a **Retry-After** HTTP response header is RECOMMENDED to be returned. * **R** the code is recommended to be supported.   The default response code for a positive response is marked in **bold**. | | |
| [R 28|1] | An API SHALL implement an error response schema to allow a standardised error handling. The response SHALL use the following JSON Schema. The JSON Schema MAY be extended. | | |
| [R 29|1] | The following rules are RECOMMENDED:   * The definitions in a conformant OpenAPI specification SHALL be considered as technical contracts between designers and developers and between consumers and providers. * Mock APIs SHOULD be created using the API description to allow early code integration for development. * The behaviour and intent of the API SHOULD be described with as much information as possible. * Operations SHOULD provide examples for request and response bodies. * Expected response codes and error messages SHOULD be provided in full. * Known issues or limitations SHOULD be clearly documented. * Expected performance, uptime and SLA/OLA SHOULD be clearly documented. * Although YAML is a supported file format of an OpenAPI specification, the JSON format SHOULD be used as the OpenAPI specification format. | | |
| [R 30|1] | All APIs SHALL apply Semantic versioning 2.0.0 :  **MAJOR.MINOR.PATCH**  The first version of an API SHALL start with a **MAJOR** version of 1.  Pre-release version information and build metadata version information SHALL NOT be used in API versioning. | | |
| [R 31|1] | All APIs SHALL use URI versioning. They SHALL include the MAJOR version as part of the URI in the format of 'v{MAJOR}'. Example: https://api.logistics.io/transport/v1/voyages  The minor and patch version SHALL NOT be used in the URI. | | |

|  |  |
| --- | --- |
| [R 32|1] | A custom header named API-Version SHALL be added to any response of the API. It SHALL be aligned with the URI version and SHALL state all three levels:  API-Version: 1.21.5 |
| [R 33|1] | An API-Version custom header MAY be added to a request. If added, it SHALL only contain the MAJOR version.  API-Version: 1 |
| [R 34|1] | An API SHALL implement a response to a GET request to the base URI of the API. The response SHALL use the JSON Schema defined in R 33. |
| [R 35|2] | APIs that are still in a **DRAFT** status SHOULD be placed in a sandbox environment. This could be done by changing the basis URL accordingly.  Example for a productive base URL:  https://api.uncefact.unece.org/v1/  Examples for a development base URL:  https://sandbox.api.uncefact.unece.org/v1/  https://staging.api.uncefact.unece.org/v1/ |
| [R 36|1] | Within a major release backward compatibility SHALL NOT be broken. |
| [R 37|1] | API clients and subscribers SHOULD be robust:   * Be conservative with API requests and data passed as input. * Be tolerant with unknown fields in the payload, but do not eliminate them from payload if needed for subsequent PUT requests. |
| [R 38|1] | An API SHALL NOT be set to **DEPRECATED** until a replacement service is running with status **ACTIVE**. The root service of the API SHALL provide the **Deprecation Header Field** and the **Sunset HTTP Response Header Field** .  A **Link header** SHALL be added in combination with the **Deprecation header**. It SHALL provide a link to the documentation. A second **Link header** SHALL be  added linking to the replacement version of the API. |
| [R 39|1] | Deprecated endpoints SHALL be documented in the OpenAPI specification using the **DEPRECATED** property introduces since OpenAPI 3.0.0.  Deprecated endpoints SHOULD provide the Deprecation Header Field and the Sunset HTTP Response Header Field.  A Link header SHALL be added in combination with the Deprecation header. It SHALL provide a link to the documentation.  Where possible, communication SHOULD be sent to consumers of deprecated endpoints. |
| [R 40|1] | The introduction of a major version SHOULD be avoided, whenever possible. This MAY be achieved as follows:   * Create a new service endpoint, if the process is changed. * Duplicate and Deprecate: add a **Deprecation Header** to the old service including a **Link Header** to documentation and to the new service. Eventually add a **Sunset Header**. * Create a new resource (a variant of the old) in addition to the old. |
| [R 41|2] | New resources or service endpoints can be added during a minor release. In order to support the implementation of those new services a sandbox environment SHOULD  be provided to the interested or affected consumers. |
| [R 42|1] | It is RECOMMENDED that no more than 3 parallel MAJOR versions are available. Implementers of the API SHALL NOT be more than 1 major version behind the latest version. |
| [R 43|1] | In order to provide a better experience for API consumers, APIs SHOULD provide a list of state transitions that are available for each resource. As possible values for link relation types the official IANA registry list SHALL be used. It MAY be extended.  Any extension SHALL be documented in the API specification. |
| [R 44|1] | All API endpoints SHALL be secured. HTTPS SHALL be used. The OAUTH2 security scheme is RECOMMENDED. Other security schemes MAY be used. |

|  |  |
| --- | --- |
|  | The receivers endpoints of subscription callbacks MAY be designed with different security measures like those described in chapter 6.3.  The aspects described after rule 32 of API security are RECOMMENDED to be implemented. |

# Appendix C: 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 – e.g.  “consignee” as a relationship between “consignment” and “party” |
| B2B | Business to Business |
| BBIE | Basic Business Information Entity – a term from CCTS that describes a property of a class such as party.name |
| BRS | Business requirement specification |
| CamelCase | CamelCase is a naming rule for a technical representation of identifiers consisting of several words. White spaces are removed and every new word begins with a capital letter. Example: **this identifier** is written  as **thisIdentifier** in camelCase. |
| CCL | Core Component Library |
| 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” |
| HATEOS | Hypermedia as the Engine of Application State |
| 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. |
| Kebab-case | Kebab-case is a naming rule for a technical representation of identifiers consisting of several words. Hyphens are used to connect words. Example: **this identifier** is written as **this-identifier** in  kebap-case. |
| 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 |
| 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 |
| UN/CEFACT | United Nations Centre for Trade Facilitation and Electronic Business |
| UNECE | United Nations Economic Commission for Europe |

|  |  |
| --- | --- |
| **Term** | **Definition** |
| 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 Markup Language |
| XMI | Xml Metadata Interchange - a well-established OMG standard for exchange of UML models between different tools. |

**Table 9 - Glossary**