国連/CEFACT
国連
貿易円滑化・電子ビジネスセンター
（国連/CEFACT）

方法論および技術プログラム開発領域
仕様部門

OPENAPI の命名規則と設計規則

技術仕様

出典: API TechSpec プロジェクト チーム
アクション: 公開の準備ができました
日付: 2022 年 9 月 13 日
ステータス: v1.0

摘　要

この OpenAPI 命名規則および設計規則技術仕様では、ビジネス情報を一貫して表現するために、OpenAPI 仕様に基づいて API を指定、記述、実装するために必要なアーキテクチャと一連の規則を定義します。これは、OpenAPI 仕様および 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限に高めることに関心のある他の組織によって使用される予定です。

1.3　OpenAPI 命名および設計ルール プロジェクト チーム

私たちは、この国連貿易円滑化・電子ビジネスセンター（UN/CEFACT）OpenAPI 命名および設計ルール技術仕様の開発に多大な貢献をしてくださった方々に感謝の意を表します。

ATG2 議長
マレク・ラスコウスキー
プロジェクトリーダー
イェルク・ヴァルター
主任編集者

アンドレアス・ペレキエス
ゲルハルト・ヘームスケルク

1.4 謝辞

このバージョンの UN/CEFACT OpenAPI 命名および設計ルール技術仕様は、標準開発組織（SDO）間の統一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。

- デジタルコンテナ輸送協会
- GS1
- オデット

1.5 連絡先情報

ATG2 – マレク・ラスコウスキー、Marek.laskowski@gmail.com
NDR プロジェクト リード – Jörg Walther、jwalther@odette.org
編集者 – アンドレアス・ペレキエス、Andreas@pelekies.de
編集者 – ゲルハルト・ヘームスケルク、Gerhard.heemskerk@kpnmail.nl

1.6 表記法

キーワード「MUST」、「MUST NOT」、「REQUIRED」、「SHALL」、「SHALL NOT」、「SHOULD」、「SHOULD NOT」、「RECOMMENDED」、「MAY」、「OPTIONAL」は、この仕様に出現する場合、Internet Engineering Task Force (IETF) の Request For Comments (RFC) 2119 に記載されているように解釈されます。。

[Example] 定義またはルールの表現。 例は参考になります。

[Note] 説明情報です。 メモは参考になります。

[R n|c] 準拠を必要とするルールの識別。 ルールは規範的なものです。 仕様のバージョン間での連続性を確保するために、ルール番号「n」はランダムに生成されます。 削除されたルール番号は再発行されません。 追加されたルールには、以前に使用されていなかった乱数が割り当てられます。

パイプ記号 | の後の 2 番目の数字「c」 セクション 3.1 で定義されているように、指定されたルールの適合カテゴリを識別します。A[+Inf] は、有益ではあるが規範ではないルールを識別するために追加される場合があります。

`Courier` 太字の Courier フォントで表示されるすべての単語は、値、オブジェクト、またはキーワードです。 空白などの印刷不可能な文字の表現は二重引用符で囲まれます。例えば [" "]。

<<var>>　　すべてのプレースホルダーは、二重の小なり文字と大なり文字で囲まれます。 プレースホルダーの意味は本文中に説明されています。

2　はじめに

2.1　目的

この OpenAPI NDR 技術仕様文書は、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サポートすることを目的とした一連の文書の一部を形成します。

UN/CEFACT 参照データ モデルの任意のレイヤーを使用して、UN/CEFACT コア コンポーネント技術仕様バージョン 2.01 に従って準拠した OpenAPI 仕様を作成します。これには、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と同様な Buy-Ship-Pay、Accounting （会計処理）などの包括的な RDM から道路運送状（eCMR; Road Consignment Note）や原産地証明書（COO; certificate of origin）などの単一メッセージの実装までが含まれます。

2.2　要 件

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。

2.3　依存関係

この文書は以下に依存します
1. UN/CEFACT コアコンポーネント技術仕様バージョン 2.01。
2. JSON スキーマの命名および設計ルールの技術仕様。

2.4　注意事項 (caveat; 特許権保護願い) *と前提条件*

この仕様を採用した結果として作成された仕様は、普遍的に無料でアクセス可能で検索可能なライブラリ内の OpenAPI 仕様文書として公開される必要があります。 UN/CEFACT は、アクセスを希望する政府、個人、組織がそのコンテンツを自由に利用できるようにします。
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ますが、CCTS 以外の開発者も他の論理データ モデルや情報交換に使用できます。
この仕様では、スクリプトやその他の手段による変換については扱いません。XML、JSON-LD、OWL、XMI などの CCTS アーティファクトの他の表現には対応していません。
標準は相互運用性を促進します。 この電子化と設計原則の定義の作成では、いくつかの情報源が次の順序で考慮されました。

1. OpenAPI 3.1.0 仕様
2. インターネット標準化団体によって RFC として定義された標準
3. DCSA API 設計原則 1.0

4. json:api 仕様
5. 専門家の経験

2.5 指導原則

3. OpenAPI の作成
   UN/CEFACT OpenAPI 設計ルールは、手動生成だけでなく自動生成による OpenAPI 仕様の作成もサポートしています。
4. ツールの使用とサポート
   UN/CEFACT OpenAPI の設計では、作成、管理、保存、プレゼンテーションのため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。
5. 技術仕様
   UN/CEFACT OpenAPI の命名規則と設計規則は、OpenAPI 推奨ステータスと同等の技術仕様に基づいています。
6. OpenAPI 仕様
   UN/CEFACT OpenAPI の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。
7. 相互運用性
   UN/CEFACT OpenAPI 仕様で同じ情報を表現する方法の数は、できる限り 1 つに近づける必要があります。
8. メンテナンス
   UN/CEFACT OpenAPI 仕様の設計では、メンテナンスを容易にする必要があります。
9. コンテキストの敏感性
   UN/CEFACT OpenAPI 仕様の設計では、状況依存の文書タイプが排除されないようにする必要があります。
10. 実装の容易さ
    UN/CEFACT OpenAPI 仕様は、設計対象のコンテキストにおいて直観的かつ合理的に明確である必要があります。これらは、REST API（別名 RESTful API）や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。

2.6 相互運用性

何十年にもわたって、B2B および B2A プロセスの業界および国家を超えた調和が、何千人もの専門家によってセマンティックな UN/CEFACT 参照データ モデルの開発に費やされてきました。 この途方もない成果は、この範囲と深さにおいて二度と存在するものではありません。 セマンティック定義から構文への明確なパス（逆はありません）は、これらのセマンティック データ モデルが構文中立（syntax-neutral）であり、したがって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。 この目的のために、それらは NDR 仕様を介して（UN/CEFACT）構文に直接マッピングされるか、他のセクターのデータ モデルおよび構文にマッピングされます。

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。 実際には、これは今日では当てはまらないことが多く、API の設計、ドキュメントの範囲と深さ、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がまだ初期段階にあるためです。
ここでのキーワードは相互運用性です。

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。 彼らの協力により、相互運用性の次の側面が促進されます。

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、たとえば 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。
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。
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。
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、異なるコンテキスト化（個々の要件の考慮）をできるだけ少なくする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、個別に実装されるのではなく、残りの受信者によって読み取られることを意味します。
5. 送信の相互運用性: ビジネス パートナーは、統一された送信方法と、SFTP、OFTP2、または AS2 などの関連するセキュリティ対策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、この次元は従来の EDI 実装ではそれほど重要な役割を果たしません。EDI は通常、大量のデータ用に最適化されています。

WebAPI を実装する場合、原則として相互運用性に対する同じ要件が存在します。 以前の WebAPI の本質的な違いは、多数のユーザーを API に接続するアプローチです。 たとえば、地図、ルート、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。 REST の構成可能性の原則は、さまざまなサービス（おそらくはさまざまなプロバイダーからの）が、WebAPI で処理するための全体的なソリューションに組み合わ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、定員、条件、チケットが航空会社によって割り当てられ、決済サービスプロバイダーが接続され、多くの場合、国境を越えるフライトのさまざまな税体系を正確に計算する専門の請求サービスが接続されます。 多くの消費者が 1 つの API（請求サービス）を使用する必要があるだけでなく、1 人の消費者が同じプロセス（航空会社による）で多くの API を使用する必要があるという側面により、WebAPI の相互運用性要件が拡張されます。

6. 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メソッドとルールにより、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、組織横断（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。
7. サービスの相互運用性: 同じプロセス要件をマッピングする際の統一エンドポイントにより、Web API を介した B2B 通信が促進されます。

次の表は、Web API で相互運用性の 7 つの側面をどのように実現できるかを示しています。

相互運用性の次元	ガイドライン
ビジネスプロセスの相互運用性	UN/CEFACT 内では、調和されたビジネス要件仕様（BRS）を実装することにより、ビジネス プロセスの相互運用性が実現されます。
セマンティックな相互運用性	CCTS とその派生セマンティック参照データ モデル（RDM）は、UN/CEFACT ユーザーにとってのこのディメンションの基礎となります。UN/CEFACT 語彙、JSON スキーマ アーティファクト、および UN/CEFACT XML 標準は、これらのセマンティック要件をそれぞれの構文で実装します。
構文の相互運用性	ユーザー グループが統一されたデータ交換構文の使用に同意すると、この次元が達成されます。 open API 仕様を作成するときは、たとえ後の交換構文が XML 形式であっても、使用する構文は常に JSON スキーマとしてモデル化する必要があることに注意してください。 これは Open API 仕様で定義されています。
コンテキスト化の相互運用性	（特定の業界などの）実装ガイドラインでは、交換されるデータまたはメッセージ構造にコンテキスト化をどのように適用するかを定義します。
伝送の相互運用性	この寸法は実装ガイドラインにも規定されています。 特に、認可や認証などのセキュリティの側面も含まれます。
API 設計の相互運用性	この NDR 仕様は、API 設計の相互運用性を定義します。 とりわけ、フィルタリング、ページネーション、エラー処理のルールが含まれています。
サービスの相互運用性	優れた Open API 仕様は、特にサービスの相互運用性に重点を置いています。サービスが適切に設計されていれば、複数のビジネス パートナーによって実装されるように設計された API の相互運用性を促進できます。 たとえば、ユーザー グループは最小限のサブセットを持つサービスのセットに同意します。 プロバイダーが特定のサービスをサポートしていない場合でも、そのサービスは実装されていますが、対応するドキュメントへの HTTP リンク ヘッダーを含む 501 メソッドが実装されていない HTTP 応答コードで常に応答します。

3 API の命名と設計ルール

3.1 適合性とコンプライアンス

政府、民間部門、および UN/CEFACT コミュニティ外部のその他の標準組織における OpenAPI 仕様の設計者は、この仕様が採用に適していると判断しました。 この広範なユーザーコミュニティ全体での再利用と相互運用性を最大限に高めるために、この仕様のルールは、これら他の組織が準拠した OpenAPI 仕様を作成できるようにしながら、全体的な相互運用性への影響が最小限に抑えられる領域での裁量または拡張性を考慮して分類されています。
したがって、アプリケーションが規範セクション、規則、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みなされます。

[R 1|1]
遵守と適合は、規範的なセクションと規則の内容の遵守を通じて決定されるものとします。 さらに、各ルールは次のように分類され、ルールの対象読者を示します。

カテゴリー	説　明
1	違反してはいけないルール。 そうしないと、コンプライアンスと相互運用性が失われます。
2	ルールは NDR 構造に準拠しながら変更される可能性があります。 カテゴリ 1 と 2 のすべてのルールに従っている場合、API は完全に準拠しています。 カテゴリ 2 のルールが変更された場合、API は準拠しなくなりますが、依然として準拠しています。
Inf	情報提供のみを目的としたルール。 別の実装が選択された場合、この仕様に対する実装の準拠性と適合性には影響しません。

表 4 - 適合カテゴリ

[R 2|1]
この OpenAPI 命名および設計ルールの技術仕様に基づくすべての API 仕様は、OpenAPI 3.1.x 仕様に準拠するものとします（SHALL）。

[R 3|1]
この仕様への準拠を主張する API 仕様は、JSON スキーマの命名および設計ルールの技術仕様 244 に記載されているようにスキーマ コンポーネントを定義するものとします（SHALL）。

3.2 設計ルール

3.2.1 構造化データ交換のメディア タイプ

[R 4|1]
構造化データ情報の転送に使用されるリクエスト本文のコンテンツとレスポンスのコンテンツは、JavaScript Object Notation (JSON) の application/json メディア タイプを使用するものとします(SHALL)。 このルールは、API が別のメディア タイプの JSON との間の変換サービスを実装する場合にのみ逸脱してもよい(MAY)。 構造化データ情報を転送するために追加のメディアタイプ(例: text/xml)を使用してもよい(MAY)。 非構造化情報が転送される場合、有効なメディアタイプを使用してもよい(MAY)。

[R 5|1]
エンコーディングは UTF-8 でなければなりません。

3.2.2 エンドポイント

[R6|2]
API 内で定義されたパスの構造は、消費者にとって意味のあるものでなければなりません(SHOULD)。 理解しやすさ、ひいては使いやすさを高めるために、パスは予測可能な階層構造に従う必要があります。

[R 7|1]
API URL は、次に説明する標準の命名規則に従う必要があります。 `https://{env}.api.{dnsdomain}/v{m}/{service}/{resource}/{id}/{sub-resource}?{query}` コンポーネントは次のように説明されます。 URL の特定のコンポーネントにルールが必須である場合、基本的な URL 構造が上記のものと異なる場合でも（例えば、 API が dns ドメインのプレフィックスとして使用されていない場合)、ルールは準拠した API 仕様に適用されるものとします(SHALL)。 ● https:// を Web プロトコルとして使用するものとします。 ● {env} は環境（テスト、サンドボックス、開発など）を示し、実稼働環境では通常省略されます。 ● {dnsdomain} は、API 実装者の DNS ドメイン（例: unece.org）です。 ● {service} は、ビジネス サービス ドメイン（トランスポートなど）を表す API 関数の論理グループです。 {service} コンポーネントはオプションです。 ● v{m} は、API 仕様のメジャー バージョン番号です。 このコンポーネントは URL に記述されるものとします（SHALL)。 URL の別の場所（ドメインのプレフィックスなど）に指定してもよい

です。

- {resource} は API リソース（委託品など）を表す複数名詞です。
- {id} は、パスパラメータとして定義されたリソースの一意の識別子です。 パスパラメータはリソースを識別するために使用されるものとします(SHALL)。 リソースのコレクションに対して操作が実行される場合、このコンポーネントはパスの一部ではありません。
- {sub-resource} はオプションのサブリソースです。 メインリソース（consignmentItem など）にコレクションまたはアクションが含まれている場合にのみ使用されます。
- {query} は、検索結果を決定するフィルターのような追加パラメータのリストです（例: consignments?loadingPort=AUSYD)。

[R 8|1]
パスとクエリを含む URL の合計文字数は、カンマ、アンダースコア、疑問符、ハイフン、プラスまたはスラッシュなどの書式設定コードを含めて 2000 文字を超えてはなりません。

[R 9|1]
エンドポイントはアクションであってはなりません。 サービスとリソースは名詞で構成されます(SHALL)。 HTTP 300 動詞はアクションに使用されるものとします（3.2.6 章を参照)。

[R 10|1]
Kebab-case2 はサービスで使用されるものとします（SHALL)。

[R 11|1]
下位の CamelCase3 は、リソース、パスパラメータ、クエリパラメータで使用されるものとします(SHALL)。

[R 12|1]
プロパティ名に関連するパスパラメータとクエリパラメータは、プロパティ名と一貫性があるものとします（SHALL)。

[R 13|1]
クエリパラメータは URL セーフである必要があります。

[R 14|1]
リソース名は複数形にする必要があります。 リソース名はスキーマと一致している必要があります(SHOULD)。スキーマが単数形で定義されている場合でも、リソースは複数形にする必要があります。リソースの複数形が標準的でない場合は、より適切な複数形の名詞を選択してもよい(MAY)。

良いエンドポイントの例:

- /employees
- /customers
- /products

3.2.3 発見可能性

REST 設計原則の 1 つはサービスの発見可能性です。 Open API 仕様では、リンクを通じ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されるものとします（SHALL)。

3.2.4 日付と時刻

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可能性があります。

[R 15|1]
クエリパラメータは、JSON スキーマ NDR 技術仕様で定義されている UNTDID 2379 json で定義されている ISO8601 準拠の日付と時刻表現を使用するものとします（SHALL)。 特定の日付、時刻、または日時を表すために、形式は日付、時刻、または日時の JSON スキーマ定義に準拠する必要があります（SHALL)。

3.2.5 UN/CEFACT セマンティクスの使用

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）が誕生しました。 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わたって存在します。

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）とサプライ チェーン（SC-RDM）に対応します。 時間が経つにつれて、何百ものビジネス文書構造が意味論的モデルレベルで調和され、標準化されました。 さまざまな構文の命名規則と設計規則により、これらのセマンティック モデルを自動的に作成し、XML などの特定の構文にマッピングできます。

Web API の世界では、ドキュメント構造の送信は時代遅れであると考えられています。 REST 原則の制限が Web API に適用される場合、ビジネス ドキュメント構造は RESTful 実装には適していません。これらの構造は、リソースの疎結合の基本原則に矛盾します。 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。リソースとは、それらを組み合わせて完全な情報（ビジネス文書など）を生成する情報ブロックです。

それにもかかわらず、B2B の情報交換には多くの場合制限があり、ドキュメント構造から完全に離れることが困難になります。 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。 組織間のコミュニケーションの基本プロセスが変わらない場合、純粋にリソースベースの情報交換への移行は、新たな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。 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）に基づいて動作する場合、純粋にリソースベースの中間送信は、多くの国で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。

一方で、プラットフォームのネットワーク（物流など）が確立されている場合でも、リソースベースの交換は特定の目的には依然として役立ちます。 たとえば、運送業者による無料の配送容量を提供および予約できる市場用のプラットフォームが存在する可能性があります。 リソースごとに分割すると、通常、アイデンティティ プロバイダーが必要になり、個々のリソースの信頼できる単一のソースの問題が明確になります。

UN/CEFACT では、セマンティック データ モデルに関する基本的な JSON ベースの出版物が 2 つあります。UN/CEFACT ボキャブラリーと UN/CEFACT JSON スキーマの出版物です。

3.2.5.1 UN/CEFACT JSON スキーマ公開の使用

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 2 つのバリアントで公開されています。

8. 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 ドキュメントとその適用されたコンテキストに関連するすべての定義が含まれています。
9. 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。 このバリアントでは、JSON スキーマでサポートされる継承および検証技術が使用されます。 基本データ構造は、参照データ モデルで一緒に必要な情報ブロックを定義します。 これに基づいて、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形成されます。

JSON スキーマは、公式 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。
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の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）することが簡単に可能です。 これにより、ユーザーは明示的に、標準化された透過的な方法で、不要なオプションの属性や補足コンポーネントに「チェックマークを付け」、コード リストを制限したり、ユーザー定義のプロパティを追加したりできるようになります。

さらに、メンテナンスも非常に簡単になります。 API を JSON スキーマ パブリケーションの新しい 389 バージ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。

あるいは、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、スキーマは公式の 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。

JSON スキーマの定義方法により、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簡単な送信が可能になります。 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基づいて作成されます。 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。 単一の ASBIE ノードごとに、JSON スキーマ パブリケーションにより、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。

輸送容量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がどのように交換されるかを定義します。 多くの場合、POST と GET、または POST、サブスクライブ、および GET のシナリオを設計します。 これらのシナリオでは、最終的な受信者が誰なのか、送信者が誰なのかなどを API に伝えるために、メッセージ情報の周囲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あり、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」、「委託品目」などが含まれます。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ように計画されている場合、非常に簡単に実行できます。 それで。 まず、どのサブリソースをスタンドアロンにするかを特定する必要があります。 取引者情報のマスターデータを単一のリソースとして管理したいとします。その場合、取引当事者の種類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 のコンテキスト化されたデータ型、または対応するビジネス ドキュメント構造への参照として単純に定義できます。 次の例は、ドキュメント構造をリソースの使用状況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" }
          }
        }
      }
    }
  ]
}
```

3.2.5.2 UN/CEFACT 語彙の使用

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

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で、定義されたグラフのサブセットを使用できます。

JSON-LD を OpenAPI 仕様で直接使用したり、リンクしたり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。 さらに、JSON-LD では、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が指定されて

いません。 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ついての合理的な知識を必要とします。 一方、JSON-LD は、(公的に) 利用可能なデータが自動的にクロール、フィルタリング、評価されるあらゆる場所で絶大な威力を発揮します。 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、または特定の基準を使用したオンライン 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挙げられます。 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、定義自体には含まれていません。 定義にステート マシンを追加すると、これに役立つ可能性があります。 残念ながら、現在、この種の情報に関して広くサポートされている交換形式は存在しません。

OpenAPI 仕様には直接的なサポートが存在しないため、JSON-LD ボキャブラリを使用するには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。 API の設計を UML ツール内で実行できるようになりました。 指定された各エンドポイントでどの操作をサポートするかを定義する方法については、いくつかの仮定が行われます。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。

3.2.5.3 他の (標準化された) データ構造の使用

第 2.6 章では、WebAPI の相互運用性の 7 つの側面が定義されています。 世界的な業界横断的な観点から見ると、完全な相互運用性は、すべての側面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。 UN/CEFACT のコンテキストでは、これは、完全に準拠するには UN/CEFACT 構文だけでなく UN/CEFACT 意味定義も使用する必要があることを意味します。

ただし、この NDR 仕様は、B2B コンテキストにおける OpenAPI 仕様の設計の基本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、(非公開の) ユーザーグループ内で異なる構文または異なる意味論的仕様を使用する API 間の相互運用性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]
OpenAPI 仕様とその実装がこの NDR TS に完全に準拠するための前提条件は、UN/CEFACT セマンティクスと UN/CEFACT 構文 (UN/CEFACT XML、UN/CEFACT JSON スキーマ、UN/CEFACT 語彙など) を使用することです。 UN/CEFACT 構文または UN/CEFACT セマンティクスを使用しない OpenAPI 仕様であっても、[R 1|1] で指定されている基準を満たしていれば、この NDR TS に準拠している可能性があります。

3.2.6 操作

[R 17|1]
エンドポイントは CRUD 操作をサポートすることが推奨されます。（作成、読み取り、更新、削除)。エンドポイントがサポートを意図していない場合、例: 削除操作の場合、3.2.10 章で定義されているように HTTP 応答コードを返すものとします（SHALL)。

HTTP メソッド	説明
GET	リソースを取得/読み取ること。
POST	新しいリソースを作成するか、システムの状態を変更するリソース上で操作を実行すること、例えば、メッセージを送ること。
PUT	リソースをリクエストで指定された別のリソースと置き換えること。
PATCH	リソースに対して部分的な更新を実行すること。
DELETE	リソースを削除すること。
HEAD	リクエストに関するメタデータの取得用で、例えば、クエリはどれくらいの結果を返すでしょうか?（実際にクエリを実行することはありません)。 これは、HATEOS 実装でリンク チェーンをたどるのにも使用できます。 例は 4.3.2 章に示されています。
OPTIONS	CORS（クロスオリジン リソース共有）リクエストが実行できるかどうかを判断するために使用されます。 これは主に、フロントエンド Web アプリケーションで API が直接使用できるかどうかを判断するために使用されます。

3.2.6.1 リソースの収集

次の操作はリソースのコレクションに適用できます。

HTTP メソッド	リソース パス	操作	例
GET	*/resources*	リソースのコレクションを取得する	*GET /employees or GET /employees?status=open*
HEAD	*/resources*	リソース コレクションのヘッダーとリンク情報を取得します、例えば ページネーション用	HEAD /employees or HEAD /employees?birthday=2022-04-16

注記

同じリクエスト内で複数のリソース インスタンスを作成または更新することは標準化されていないため、避ける必要があります。 受信確認や、一連のバッチでの部分的な成功の処理方法など、ケースバイケースで考慮する必要がある要素があります。

3.2.6.2 単一リソース

次の操作は単一のリソースに適用できます。

HTTP メソッド	リソース パス	操作
GET	/resources/{id}	リソース ID に対応するインスタンスを取得します
PUT	/resources/{id}	リソース インスタンスを置き換えて更新するには –「*この新しいものを取り出し、そこに* _ **配置します** _」
DELETE	/resources/{id}	リソースに基づいてリソース インスタンスを削除します、例えば id
HEAD	/resources/{id}	リソースのヘッダーとリンク情報を取得します
PATCH	/resources/{id}	指定した属性に対して追加、更新、削除などの変更を実行します。 Is がリソースの部分的な更新を実行するためによく使用されます

3.2.6.3 冪等性

冪等 HTTP メソッドは、異なる結果を得ることなく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、二次呼び出しでは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。

たとえば、N 個の同様の DELETE リクエストを呼び出すと、最初のリクエストでリソースが削除され、応答は 200 (OK) または 204 (No Content) になります。 さらにリクエストが返されます（見つかりません)。 明らかに、応答は最初のリクエストとは異なりますが、元のリソースはすで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 メソッド	冪等であるか
GET (取得)	True
POST (登録)	False
PUT (更新)	True
*PATCH (*変更*)*	False
DELETE (削除)	True
HEAD	True
OPTIONS	True

表 5 – 演算の冪等性

[R 18|1]
API は、上記のリストで指定された操作の冪等性に従うものとします (SHALL)。

[R 19|1]
API は、POST や PATCH などの非冪等操作をフォールトトレラントにするために、冪等性のキー

（Idempotency-Key） HTTP ヘッダー フィールドと対応する実装アドバイスを実装する必要があります（SHOULD)。

3.2.7 ページネーション（ページを構成する文字、図表、画像などのレイアウトを決めること）

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。ページネーションを行わずに、大手インターネット検索エンジンの 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、データ負荷を適切な量に保つのに役立ち、同時にセキュリティ面もサポートします。

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、このアプローチではあいまいな結果が生じます。API が、目的地別に並べられた特定の運送業者のすべての計画された輸送移動のリストを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レコードのセットを要求する前に、3 つのことが起こる可能性があります。

- データバンクはまったく変更されません。 そうすれば、記録の次のページは正確です。
- レコードがデータベースに追加されます。このレコードは、クライアントがすでに受信している最初のページの結果リストに分類されます。 その場合、前のページの最後の結果が 2 番目のページ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ます。
- 逆の場合、最初のページですでにクライアントに返されている計画された輸送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 レコードが前のページに移動します。 ここでクライアントが次のページをクエリすると、このデータ レコードはまったく送信されません。

組織間のデータ交換で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 7。 さらに、カーソル ページネーションは、オフセット ページネーションよりも大規模なデータセットに対してはるかに時間効率が高くなります。

[R 20|1]
API でページネーションが使用される場合、キーセットベースのページネーション（カーソルページネーション）が使用されるものとします（SHALL)。 これは、消費者が特定のページをリクエストすることはできず、代わりにサーバーによって提供されるページリンクを選択する必要があることを意味します。 サーバーは、HTTP 応答ヘッダー内で前後のページへのリンクを提供する必要があり(SHALL)、最初と最後のページへのリンクを提供する必要があります(SHOULD)。 さらに多くのリンクが提供される場合があります。 カーソル値は文字列であり、サーバーが任意の方法を使用して作成します。 これは、特定の時点のフ

ィルターと並べ替えパラメーターを含むクエリの結果リスト内のポイントを識別します。したがって、リストはカーソルより前のリストとカーソルより後のリストに分割されます。 オプションで、カーソル「上」に位置する結果が 1 つ存在する場合があります。

カーソル ページネーションにより、特定の時点でのフィルタリング/並べ替え基準を使用したクエリの一貫したデータ セットが保証されます。 別の消費者が少し後に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]
コレクション結果に対する GET リクエストはページネーションを実装する必要があります(SHOULD)。 エンドポイントで指定されていない場合、デフォルトおよび最大ページ サイズは 100 である必要があります。 結果として生じるページの負荷が大きい場合は、これを小さくする必要があります。 デフォルトのページ サイズはエンドポイントごとに変更できます (MAY)。 消費者はデフォルトのページサイズをオーバーライドできるべきです(SHOULD)。 結果を取得するときに使用されるフィルタ、ソート、および/またはページサイズが変更された場合、ページネーションは最初のページにリセットされるものとします(SHALL)。 次の表で説明されているクエリパラメータが使用され、ルールが適用されます。

タイプ	**説　明**	**例**
ページサイズ	サーバー/仕様によって定義されたデフォルトのページ サイズをオーバーライドします。	Example for the first query: GET /transportMovements? carrier=ABC &status=PLANNED &sort=estimatedTimeOfArrival &pageSize=50
現在のページ	現在のページへのリンク。	Link: <https://api.unece.org/ transportMovements? cursor=XXX>; rel="current"
1 ページ目	最初のページへのリンクです。 最初のページの場合、リンクは省略できます。	Link: <https://api.unece.org/ transportMovements? cursor=XXX>; rel="first"
次のページ	次のページへのリンク。 最後のページの場合、次のページへのリンクは省略してもよいです。 それ以外の場合は、ヌルリンクが提供されます。	Link: <https://api.unece.org/ transportMovements? cursor=XXX>; rel="next" Link: <null>; rel="next"

前のページ	前のページへのリンクです。 最初のページの場合、前のページへのリンクは省略できます。 それ以外の場合は、ヌルリンクが提供されます。	`Link: <https://api.unece.org/transportMovements?cursor=XXX>; rel="prev"`
最後のページ	最後のページへのリンクです。 最後のページの場合、最後のページへのリンクは省略してもよい(MAY)。それ以外の場合は、ヌルリンクが提供されます。	`Link: <https://api.unece.org/transportMovements?cursor=XXX>; rel="last"`

複数のリンクを指定する場合は、カンマで区切ります。

```
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"
```

3.2.8 フィルタリング

API でコレクションをフィルタリングおよび並べ替える機能を提供すると、消費者は適合 API の使用方法を選択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]
並べ替えとフィルタリングはクエリパラメータを使用して行われます(SHALL)。 パス パラメーターの使用は、特定のリソースを識別する場合にのみ許可されます。

3.2.8.1 出力の選択

消費者は、クエリ パラメータで属性を指定することで、応答ペイロードで返したい属性を指定できます。

応答の *first_name* フィールドと *last_name* フィールドのみを返す例:

?attributes=first_name,last_name

3.2.8.2 単純なフィルタリング

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。

`?last_name=Citizen` は、Citizen に一致する属性 last_name を持つリソースのコレクションをフィルターで除外します。

`?last_name=Citizen&date_of_birth=1999-12-31` は、Citizen に一致する last_name 属

性と、1999 年 12 月 31 日に一致する date_of_birth を持つリソースのコレクションをフィルターで除外します。

[R 23|1]
一般的なガイドとして、フィルタリングは大文字と小文字を区別せずに実行する必要があります。 大文字と小文字を区別しないフィルタリングに 600 を選択するかどうかは、明確に文書化されなければなりません（SHALL)。

この手法で使用する場合にサポートされる演算子は、equal = 演算子のみです。 他の演算子と条件については次のセクションを参照してください。

3.2.8.3 LHS 演算子を使用した高度なフィルタリング

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状況があります。 より複雑なフィルタリング ロジックを定義するには、予約済みキーワード フィルタを使用します。 一般的なパターンは

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

この場合の = 記号は、URL クエリ文字列と RFC 3986 との互換性を維持するためにあります。ただし、API サービスは実際の比較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。

次の演算子がサポートされています。

- [gte] 以上
- [egt] 以上
- [gt] より大きい
- [lt] 未満
- [lte] 以下
- [elt] 以下
- [ne] 等しくない

LHS 属性を使用したフィルタリングの例:

```
/path?creation_date[gt]=2020-11-30
```

3.2.8.4 Lucene 構文を使用したリッチ クエリ

[R 24|1]
アプリケーションが、論理演算子、あいまい検索、グループ化などを含む、より豊富な検索およびフィルター機能をサポートする必要がある場合、API は、lucene クエリ構文に従ってクエリ文字列を適用してもよい[MAY]。 その場合、通常、フィルタリング パラメータとクエリ パラメータはリクエスト

本文で送信されます。

3.2.8.5 GraphQL

API 実装者がクライアントに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応答データセットを柔軟に定義できるようにしたい場合、GraphQL クエリ インターフェイスを提供できます。 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わた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にどのデータ要素が必要かを正確に指定できます。 非常に複雑な RESTful クエリを構築している場合は、代替手段として GraphQL を検討する必要があります。

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。

3.2.9 並べ替え

多くの場合、特定の順序でデータを提供することがクライアント アプリケーションからの要件となるため、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。

[R 25|1]
並べ替えは指定されたフィールドに限定されるべきです。 ソート方向は省略してもよい(MAY)。 デフォルトの並べ替え方向は昇順です。 コロン `:` は、フィールド名と並べ替え方向を区切るために使用されます。 複数の並べ替えフィールドはカンマ `,` で区切られます。

クエリ	パラメータの説明
sort=name *sort=name:asc*	名前フィールドで昇順に並べ替えます。
sort=name:desc	名前フィールドを降順に並べ替えます。
sort=yearOfBirth,name:dec	生まれ年の昇順に並べ替えます。 同じ年が 2 つ存在する場合は、名前を誕生年の降順に並べ替えます。

表 6: 並べ替えの例

3.2.10 API レスポンスとエラー処理

[R 26|1]
HTTP 応答コードを使用するものとします（SHALL)。 次の表は、準拠 API でサポートされる HTTP 応答コードを定義しています。 列「応答」は、第 0 章で説明されているように、追加のエラー応答ペイロードが返されることが推奨されるかどうかを示します。

コード	状態	応答	いつ使用するか
200	OK	No	リクエストは正常に処理されました。
201	Created	No	リソースが作成されました。 Location HTTP 応答ヘッダーは、新しく作成されたリソースがアクセスできる場所を示すために返されるものとします（SHALL)。
202	Accepted	No	リクエストは受け入れられ、非同期に処理されました。
204	No content	No	サーバーはリクエストを正常に処理しましたが、コンテンツを返しません。 クライアントが別の場所に移動する必要はありません。
400	Bad Request	Yes	サーバーは(リクエスト構文の不正、サイズが大きすぎ、無効なリクエスト メッセージ フレーム、不正なリクエスト ルーティング、リクエスト内の無効な値などの)リクエストは処理できません。 機密情報の場合、代わりにコード 404 Not found が返されることがあります。
401	Unauthorised	Yes	リクエストを認証できませんでした。 機密情報の場合、代わりにコード 404 Not found が返されることがあります。
403	Forbidden	Yes	リクエストは認証されましたが、リソースへのアクセスが許可されていません。 機密情報の場合、代わりにコード 404 Not found が返されることがあります。
404	Not found	Yes	リソースが見つかりませんでした。
405	Not Allowed	Yes	このメソッドはこのリソースには実装されていません。 応答には、リソースの有効なメソッドのリストを含む Allow HTTP 応答ヘッダーが含まれてもよい(MAY)。
408	Request Timeout	No	応答を受信する前にリクエストがタイムアウトしました。 Retry-After HTTP 応答ヘッダーが返されることが推奨されます。
415	Unsupported Media Type	Yes	サポートされていないメディア タイプ
422	Unprocessable Entity		このステータス コードは、リクエストで指定されたコンテンツ タイプがサーバーでサポートされていないため、サーバーがリクエストの受け入れを拒否したことを示します。
429	Too Many Requests		（消費者からの）要望が多すぎます。 Retry-After HTTP 応答ヘッダーが返されることが推奨されます。応答コードの理由に関する情報を含む応答本文が返されてもよい(MAY)。 考えられる理由としては、日/時間/月などのリクエストの割り当てを超過したことが考えられます。
500	Internal Server error		内部サーバーエラー。 応答本文にはエラー メッセージが含まれる場合があります。応答本文は、サーバー構成情報（バージョン、パス、使用されるデータベースなど）を明らかにしてはなりませ

		ん（SHALL)。
501	Method Not Implemented	これは、リクエスト メソッドがサーバーでサポートされておらず、リクエストされたリソースを処理できないことを示します。特定のサーバーが指定されたメソッドのいずれかを（まだ）サポートしていない場合、この応答コードを実装すると、同じ仕様に基づく API 実装間の相互運用性が向上します。 Link HTTP 応答ヘッダーは、特定のドキュメントを指すようにすることが推奨されます。
503	Service unavailable	これは、(メンテナンス上の理由などにより)サービスが利用できないことを示します。 Retry-After HTTP 応答ヘッダーが返されることが推奨されます。

表 7: HTTP 応答コード

[R 27|1]

次の表は、適合 APIs によって特定の HTTP リクエスト メソッドに対してどの HTTP レスポンスコードがサポートされるべきかを定義します。 列の使用は、準拠 API が指定された http 応答コードをどのようにサポートするかを示します。

- M コードはサポートされるものとします
- MA は、転送や処理時間などの理由で応答が非同期で処理されるリクエストに対してサポートされるものとします。 その場合、それぞれのリソースを指す Location HTTP 応答ヘッダーが与えられるものとします（SHALL)。 さらに、Retry-After HTTP 応答ヘッダーが返されることが推奨されます（RECOMMENDED)。
- R コードをサポートすることをお勧めします。

肯定的な応答のデフォルトの応答コードは**太字**でマークされています。

HTT リクエストメソッド	**コード**	**状 態**	**使 用**
GET	200	**OK**	M
	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			
	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	M
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

3.2.11 エラー応答ペイロード

一部のエラーでは、HTTP ステータス コードを返すだけで応答を伝えることができます。 追加のエラー情報を応答本文に追加できます。 例えば; HTTP 400 不正なリクエストは検証エラーとしては一般的すぎると考えられるため、応答本文に詳細な情報を含める必要があります。

[R 28|1]

API は、標準化されたエラー処理を可能にするエラー応答スキーマを実装するものとします (SHALL)。 応答では次の JSON スキーマを使用するものとします (SHALL)。 JSON スキーマは拡張される場合があります。

```
{
  "$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
}
```

次の定義が適用されます:

エラー応答属性	説明
id	特定のエラーの識別子
詳細	この問題の発生に特定した、人が読める説明。
コード	アプリケーション固有のエラー コード
ソース	エラーの原因に関するコンピューター処理可能な情報を含むオブジェクト。
パラメータ	エラーが発生した（クエリ）パラメータ。
ポインタ	リクエストドキュメント内の関連エンティティへの JSON ポインタ [RFC6901] [例: プライマリ データ オブジェクトの場合は「/data」、特定の属性の場合は「/data/attributes/title」]。

表 8: エラー応答の属性

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"
    }
  }
  ]
}
```

3.2.12 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=AUBN747
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}
```

4 十分に文書化された API

4.1 一般的な考慮事項

[R 29|1]
次のルールが推奨されます: - 準拠した OpenAPI 仕様の定義は、設計者と開発者の間、および消費者とプロバイダーの間の技術契約とみなされなければならない（SHALL)。 - 開発のための早期コード統合を可能にするために、API 記述を使用してモック API を作成する必要があります（SHOULD)。 - API の動作と意図は、できるだけ多くの情報で説明される必要があります。 - オペレーションでは、リクエスト本体とレスポンス本体の例を提供する必要があります。 - 予期される応答コードとエラー メッセージを完全に提供する必要があります（SHOULD)。 - 既知の問題または制限は明確に文書化されるべきです。 - 期待されるパフォーマンス、稼働時間、SLA/OLA を明確に文書化する必要があります。 - YAML は OpenAPI 仕様でサポートされているファイル形式ですが、JSON 形式を OpenAPI 仕様形式として使用する必要があります（SHOULD)。

4.2 API のバージョン管理

4.2.1 バージョン管理スキーム

[R 30|1]
すべての API はセマンティック バージョン管理 2.0.0 を適用するものとします: `MAJOR.MINOR.PATCH` API の最初のバージョンは、メジャー バージョン 1 で始まるものとします（SHALL)。 プレリリース バージョン情報とビルド メタデータバージョン情報は、API のバージョン管理で使用

してはなりません（SHALL NOT）。

API バージョン番号を増やすときは、次のガイドラインに従ってください。

- 下位互換性を損なう API 変更を行った場合のメジャー バージョン、
- 下位互換性のある方法で機能を追加する場合のマイナー バージョン、および
- 下位互換性のあるバグ修正を行う場合の PATCH バージョン。 PATCH には新しい機能は含まれません。

4.2.2 URI のバージョン管理

[R 31|1]
すべての API は URI バージョン管理を使用する**ものとします**（SHALL）。 URI の一部として MAJOR バージョンを「`v{MAJOR}`」の形式で含めるものとします（SHALL） 例: https://api.logistics.io/transport/v1/voyages マイナー バージョンとパッチ バージョンは URI で使用してはなりません（SHALL NOT）。

4.2.3 バージョン情報の提供

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。 これらは HATEOS（第 4.3.2 章を参照）のサポートを義務付けます。 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、バージョン情報を含む、呼び出される API に関する基本的なメタデータを提供すると、RESTful でないシナリオでも役立ちます。
[R 32|1]

[R 32|1]
API-Version という名前のカスタムヘッダーは、API の応答に追加されるものとします（SHALL）。これは URI バージョンと一致するものとし、3 つのレベルすべてを示すものとします: `API バージョン: 1.21.5`

[R 33|1]
API-Version カスタムヘッダーをリクエストに追加してもよい(MAY)。追加する場合は、メジャー バージョンのみを含めるものとします。 `API バージョン: 1`

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、準拠した API から次の情報を取得できます。

[R 34|1]

API は、API のベース URI への GET リクエストに対する応答を実装するものとします (SHALL)。応答では次の JSON スキーマを使用する必要があります:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
     "title": { "type": "string" },
     "version": {
       "type": "string",
       "pattern": "^¥¥d+(-.+)?¥¥.¥¥d+(-.+)?¥¥.¥¥d+(-.+)?$"
     },
     "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
```

```
}
```

次の定義が適用されます。

- title: API の名前。 OpenAPI 仕様で定義されている API タイトルと同一であるものとします。
- version: API バージョン
- status: API の動作ステータス。 次の値が使用されます。
  - ACTIVE: API は生産段階にあります。 特定のサービスのメンテナンスまたは非推奨は、サービス レベルで示されるものとします(SHALL)。 有効とは、API が生産段階に入ってからの過去の瞬間を定義します。
  - DEPRECATED 非推奨: 完全な API はサポート終了フェーズに入ります。 有効とは、将来、API が RETIRED に切り替わる瞬間を定義します。 非推奨の規則 (4.2.5 章を参照) が追加で適用されます。
  - RETIRED: 完全な API はサポート終了段階にあります。 有効とは、API が RETIRED に設定された過去の瞬間を定義します。 非推奨の規則 (4.2.5 章を参照) が追加で適用されます。
- 有効: ステータスに対応する瞬間。
- 仕様: 現在の API の OpenAPI 仕様に対する有効な URI。 このようにして、利用可能なサービスとデータ型は、その基本構造から自己記述的になります。 OpenAPI 仕様は可能な限り公開し、必要とする人が簡単にアクセスできるようにする必要があります (SHOULD)。

必要に応じて、追加のメタデータを応答に追加できます。

```
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"
}
```

ドラフトの期間中、API サンドボックス環境の開発、またはテスト段階が、意図した機能を検証するために使用されます。 開発中のこの種の API には、DRAFT のような追加の状態は提供されません。

[R 35|2]
まだドラフト段階にある API は、サンドボックス環境に配置する必要があります。これは、基本 URL をそれに応じて変更することで実行できます。 **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/

4.2.4 堅牢性

消費者向けの下位互換性を確保するには、疎結合を念頭に置いて API を開発することが重要です。

[R 36|1]
メジャー リリース内では、下位互換性が壊れてはなりません。

次の変更は下位互換性があると見なされます。

- 表現への新しいオプションのフィールドの追加
- 表現の `_links` 配列への新しいリンクの追加
- API への新しいエンドポイントの追加
- 新しいメディアタイプの追加サポート (例: `Accept: application/pdf`)

次の変更は下位互換性が**ない**と見なされます。

- 表現からのフィールドの削除
- フィールドのデータ型の変更 (例: 文字列からブール値へ)
- セマンティック定義の変更
- エンドポイントまたは機能の削除
- メディアタイプのサポートの削除

[R 37|1]
API クライアントとサブスクライバーは堅牢である必要があります。 - API リクエストと入力として渡されるデータは慎重に扱ってください。 - ペイロード内の不明なフィールドを許容しますが、後続の PUT リクエストで必要な場合はペイロードからそれらのフィールドを削除しないでください。

4.2.5 非推奨およびサポート終了ポリシー

新しい API を設計するとき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 は永久に存続することを意図した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。 他のものは、ユーザーにとってより良いオプションが利用できるようになったときに削除される可能性があります。

End-of-Life（EOL）ポリシーは、API がワークフローを ACTIVE 状態から RETIRED 状態に移行するために通過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 顧客に対して、一貫した合理的な移行期間を確保しながら、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。

メジャー API バージョン EOL

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（MAY)。メジャー API バージョンを廃止する場合は、次のルールが適用されます。

[R 38|1]
代替サービスがステータス ACTIVE で実行されるまで、API を DEPRECATED に設定してはなりません。 API のルート サービスは、Deprecation ヘッダー フィールド 13 と Sunset HTTP レスポンス ヘッダー フィールド 14 を提供するものとします。 Link ヘッダーは Deprecation ヘッダーと組み合わせて追加されるものとします（SHALL)。 ドキュメントへのリンクを提供するものとします(SHALL)。 API の代替バージョンにリンクする 2 番目の Link ヘッダーを追加する必要があります（SHALL)。

さらに、次の点を考慮する必要があります。

1. ユーザーに移行の適切な通知を与えるために、最低 60 日間の移行期間を計画する必要があります。
2. 外部ユーザーに対する API バージョンの非推奨はケースバイケースで検討する必要があり、ユーザーへの影響を最小限に抑えるために追加の非推奨時間や制約が必要になる場合があります。
3. バージョン管理された API が ACTIVE であるか、登録ユーザーがいない DEPRECATED 状態の場合、ただちに RETIRED 状態に移行する可能性があります。

[R 39|1]
非推奨のエンドポイントは、OpenAPI 3.0.0 以降に導入された DEPRECATED プロパティを使用して、OpenAPI 仕様に文書化するものとします（SHALL)。 非推奨のエンドポイントは、非推奨ヘッダー フィールドと Sunset HTTP 応答ヘッダー フィールドを提供すべきです（SHOULD)。

Link ヘッダーは Deprecation ヘッダーと組み合わせて追加されるものとします（SHALL)。 ドキュメントへのリンクを提供するものとします(SHALL)。
可能であれば、通信は非推奨のエンドポイントの消費者に送信されるべきです(SHOULD)。

[R 40|1]

メジャー バージョンの導入は可能な限り避けるべきです。 これは次のようにして達成できます(MAY)。

- プロセスが変更された場合は、新しいサービス エンドポイントを作成します。 複製と非推奨: ドキュメントと新しいサービスへのリンク ヘッダーを含む非推奨ヘッダーを古いサービスに追加します。 最終的には Sunset ヘッダーを追加します。
- 古いリソースに加えて、新しいリソース（古いリソースのバリアント）を作成します。

マイナー API バージョン EOL

指定された URL バージョン管理により、API のマイナー バージョンが変更されても URL は変更されません。 API のマイナー バージョンには、同じメジャー バージョン内の以前のマイナー バージョンとの下位互換性があります。

したがって、API バージョンのマイナー更新前、更新中、更新後のステータスは変わりません。 この変更は既存のサブスクライバには影響しないため、クライアントの移行を容易にするために非推奨状態に移行する必要はありません。

[R 41|2]

新しいリソースまたはサービス エンドポイントは、マイナー リリース中に追加できます。 これらの新しいサービスの実装をサポートするために、関心のある消費者または影響を受ける消費者にサンドボックス環境を提供する必要があります（SHOULD)。

[R 42|1]

利用できる並列メジャー バージョンは 3 つまでにすることが推奨されます。 API の実装者は、最新バージョンよりもメジャー バージョンが 1 つ以上遅れてはなりません（SHALL)。

Example

Version 1 is **RETIRED**
Version 2 is **DEPRECATED**
Version 3 is **ACTIVE**

4.3 ハイパーメディア

4.3.1 ハイパーメディア - リンクされたデータ

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 によれば、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。

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。 彼らは消費者に、元の要求に従ってどのようなオプションが利用可能であるかを知らせます。 API 内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していなくても、リソースを見つけることができます。

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、Web ページにアクセスする前に Web ページの構造を知る必要はありません。 ホーム ページを参照するだけでよく、必要に応じてナビゲーションを使用してサイトを参照できます。

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が期待されます。

4.3.2 HATEOAS

アプリケーション状態のエンジンとしてのハイパーメディア（*Hypermedia As The Engine Of Application State*）は、許可される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概念です。 ハイパーメディア リンク データの概念と同様に、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利用可能な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"
```

同じアカウントがオーバードローされた場合、許可される唯一のアクションは deposit です。

```
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
  }
}
```

4.3.3 ハイパーメディア準拠 API

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の遷移を開始します。 *GET* リクエストは、取得されたリソースの状態を変更することはありません。

[R 43|1]
API コンシューマにより良いエクスペリエンスを提供するために、API は各リソースで利用可能な状態遷移のリストを提供すべきです（SHOULD)。 リンク関係タイプの可能な値として、公式 IANA レジストリ リスト 16 を使用するものとします（SHALL)。 延長される場合があります。 拡張はすべて API 仕様に文書化するものとします（SHALL)。

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。

クライアントは、URI /users を通じ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、システム内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"
  ,...
}
```

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。

- Location ヘッダー内の作成されたリソースへのリンク（201 応答仕様に準拠するため）
- ユーザーの完全な表現を取得するためのリンク（別名セルフリンク）（*GET*）。
- ユーザーを更新するためのリンク（*PUT*）。
- ユーザーを部分的に更新するためのリンク（*PATCH*）。
- ユーザーを削除するためのリンク（*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"
```

クライアントは、後で使用できるようにこれらのリンクをデータベースに保存できます。

要約すれば：

- 各 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、クライアントは他のすべてのリソースにアクセスするためにそこに移動します。
- クライアントは、URI や状態変化に関連付けられている可能性のある応答の詳細を調べることで、さまざまなリクエストを実行したり、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ために URI を作成するロジックを構築する必要がありません。
- クライアントは、URI の作成プロセスがサーバーに属するという事実を認識します。
- クライアントは URI を不透明な識別子として扱います。
- 表現で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るようになります。 たとえば、API が新しい PATCH 操作のサポートを開始すると、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。

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、移行のリクエストを行うのに必要なデータと、関連するすべてのリンク セマンティクスを学習する必要がなくなります。

5 APIセキュリティ

[R 44|1]
すべての API エンドポイントはセキュリティで保護されなければなりません (SHALL)。HTTPS を使用するものとします。 ・動物でみる動物の心理占い無料サイト結果 1077 他のセキュリティ方式を使用してもよい(MAY)。 サブスクリプション コールバックの受信側のエンドポイントは、6.3 章で説明されているような異なるセキュリティ手段を使用して設計されてもよい(MAY)。 API セキュリティの次の側面を実装することが推奨されます。

レート制限

API の悪用を防ぐために、レート制限およびスロットル ポリシーが導入されます。 適切なアラートを実装し、しきい値に近づいた場合、またはしきい値を超えた場合に有益なエラーで応答する必要があります。 実装の詳細については、https://greenbytes.de/tech/webdav/draftietf-httpapi-ratelimit-headers-latest.html を参照してください。

エラー処理

アプリケーションがエラー メッセージを表示する場合、システムの攻撃に使用される可能性のある情報を公開してはなりません。 1089 エラー メッセージを提供する場合は、次の制御を確立する必要があります。

- API は、標準の HTTP ステータス応答やエラー 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しなければなりません。 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
- 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。

監査ログ

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、それを調査できることです。 ロギングを実装することが推奨されます。

- アラートをトリガーする可能性のあ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
- ログデータをサニタイズしてログインジェクション攻撃を防止する

入力の検証

入力検証は、適切に形成されたデータのみがシステムに受信されることを保証するために実行されます。

これは、悪意のある攻撃の防止に役立ちます。

- 入力検証はできるだけ早く、できれば外部パーティからデータを受信したらすぐに行う必要があります。
- 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
- 入力を検証します：例：長さ/範囲/形式と種類
- 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 秒あたり何百回も失敗した入力検証を実行している人が悪意を持っていると仮定します。
- 必要に応じて正規表現を使用して文字列入力を制限する

コンテンツタイプの検証

指定されたコンテンツ タイプを尊重します。 予期しないコンテンツ タイプ ヘッダーまたは欠落しているコンテンツ タイプ ヘッダーを含むリクエストを、HTTP 応答ステータス *415 Unsupported Media Type* で拒否します。

ゲートウェイのセキュリティ機能

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。

6 イベント駆動型のデータ交換

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限界に達します。 たとえば、ジャストインタイム生産における最も重要な情報の 1 つは、工場への到着予定時刻（ETA）です。多くの場合、消費者がデータ送信者に配信の現在のステータスを定期的に問い合わせる PULL シナリオが実装されます。 あるいは、運送業者は、定期的かつ短い間隔で、各積荷品目の詳細情報を含む配送の現在のステータスに関するステータス メッセージを送信します。 これにより膨大な量のデータが生成されるため、実際にはそのような更新の最小間隔は約 15 分になります。 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が得られるのはかなり先のことです。

この問題を解決する 1 つのアプローチは、(関連性の低い) 情報を常に交換するのではなく、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。 これは、たとえば、ジオフェンスを越えた場合、温度が超過または到達していない場合、またはクリアランスに意図したよりも時間がかかる場合などに当てはまります。 消費者分野では、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に、荷物が配達まであと 10 駅であることが通知される場合です。

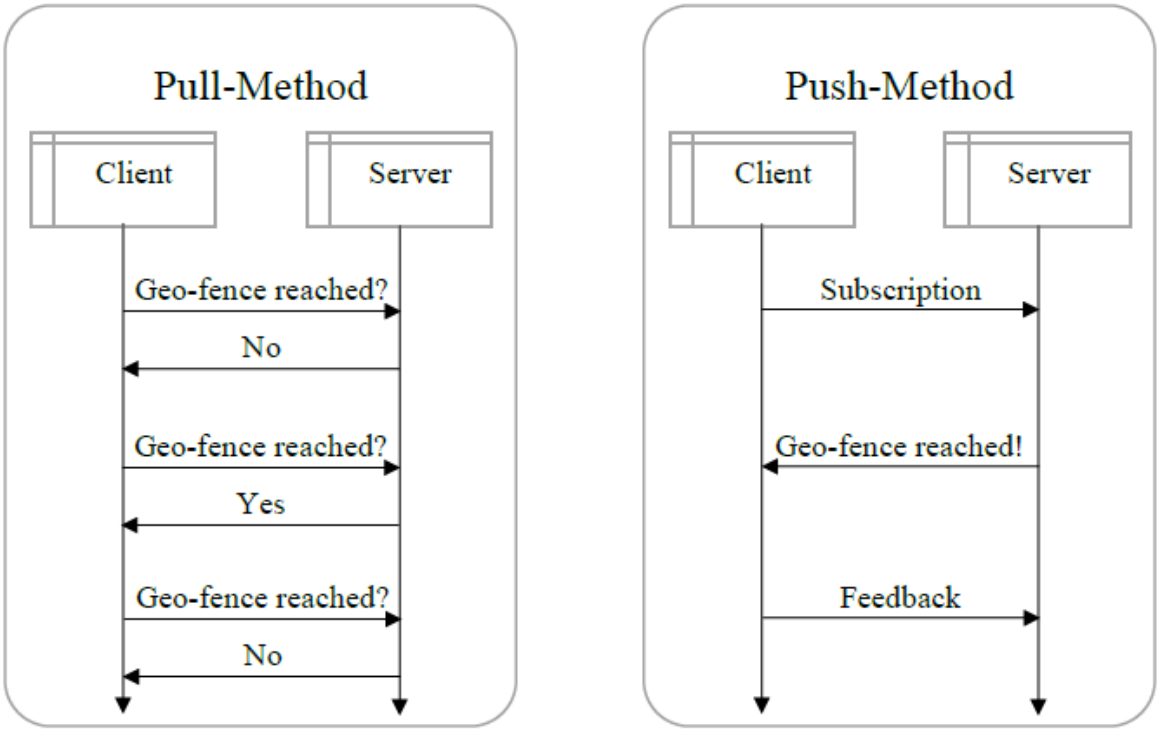

図 1: イベント駆動型のデータ交換 - プル方式とプッシュ方式

6.1 コールバック

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、コンシューマ指定の URL への非同期リクエストです。 一例として、特定の船舶が港に近づくと運送業者に通知されます。
この情報を受信できるようにするには、受信者はまず 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。 購読する際、消費者に通知す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、消費者が特定の港に近づいた場合に通知を受け取りたいという特定の旅行が挙げられます。

基本原則は、購入者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 HTTP リクエストの受信を待機することです。

6.2 Webhook

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 は 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。 これは、コンシューマがプロセスに直接接続できるため、必要に応じて、さらに処理される前に処理された情報を変更できることを意味します。 Webhook は、API の機能を拡張するために使用されます。

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。 例としては、受信した注文や支払いなどにすぐに反応したい場合です。ペイロード自体は Webhook で提供され、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などがあります。 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って、表示されている 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。

6.3 コールバックのセキュリティ ガイドライン (参考)

Webhook は同期的に動作するため、API 全体と同じセキュリティ ルールが Webhook に適用されます。対照的に、非同期コールバックでは呼び出し方向が逆になります。 このため、コールバック URL が承認された 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。
以下のルールは、DCSA の現在のアプローチに基づいています。 この文書の発行時点では試用段階にあります。 十分な実用性が実証され次第、この仕様はそれに応じて更新されます。 このような背景を踏まえ、以下のルールは純粋に参考となるものであり、規範的なものではありません。

[R 45|1+Inf]
すべてのイベント サブスクリプションは、このセクションで説明されているように、すべてのコールバック メッセージの署名に使用される共有秘密によって保護されるものとします。 シークレットは BASE64 でエンコードされて提供されるものとします (SHALL)。 プロバイダーは、いかなるエンドポイントでもシークレットを公開してはなりません (SHALL NOT)。 書き込み専用です。 プロバイダは、シークレットが適用されるアルゴリズムのセキュリティ要件を満たしていることを保証するものとします(SHALL)。

[R 46|2+Inf]
sha256 署名は、リクエスト本文に対して HMAC-SHA246 として計算されて使用されるものとします。 加入者が提供する共有秘密は、少なくとも 32 バイトの長さでなければなりません (SHALL)。長い鍵はそのアルゴリズムに追加のセキュリティを提供しないため、64 バイトを超えてはなりません (SHOULD)。 セキュリティを向上させるために、定期的にシークレット（およびコールバック URL とともに）を更新することが推奨されます。

[R 47|1+Inf]

パブリッシャー API は、サブスクリプションに対して次のエンドポイントを提供するものとします。

- POST …/subscriptions は、新しいサブスクリプションを作成します。
- GET …/subscriptions は、サブスクライバーがアクセスできるすべてのサブスクリプションをリストします。
- GET …/subscriptions/{subscriptionId} は、特定のサブスクリプションに関する詳細を取得します。
- PUT …/subscriptions/{subscriptionId} は、特定のサブスクリプションを更新します。
- PUT …/subscriptions/{subscriptionId}/secret は、特定のサブスクリプションのシークレットを更新します。
- DELETE …/subscriptions/{subscriptionId} は、特定のサブスクリプションをキャンセルします。

6.3.1 サブスクリプションの設定（参考）

サブスクリプションの設定は次の手順に従います。

1. サブスクライバは共有シークレットを定義し、そのシークレットとコールバック URL をパブリッシャーのシステムに登録します。 推測しにくい 18 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。
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。
3. サブスクライバは、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":
"MDEyMzQ1Njc4OWFiY2RlZjAxMjM0NTY3ODlhYmNkZWYwMTIzNDU2Nzg5YWJjZGVmMDEyMzQz12
08 NjU3ODlhYmNkZQ",
  ... additional filter parameters etc. ...
}

2.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

2.b Response of the subscriber that the callbackURL is valid

HTTP/1.1 204 No Content

3.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 ...
}
```

6.3.2 サブスクリプションコールの実行（参考）

サブスクリプションの呼び出しは次の手順に従います。

1. パブリッシャーはサブスクライバーのコールバック URL に対して POST を実行するものとします（SHALL)。
   - `subscriptionId` を含む `Subscription-ID HTTP` ヘッダーが追加されます。
   - リクエスト本文の計算された署名を含む、`Notification-Signature HTTP` ヘッダーが追加されます。
   - リクエストボディは、`application/json` 形式を使用して送信されます。
2. サブスクライバは POST リクエストを検証するものとします（SHALL)。 次の順序で実行する必要があります。 いずれかの検証ステップが失敗した場合、メッセージは拒否されるものとします(SHALL)。
   - すべての検証ステップがエラーなしで実行された場合にのみ、メッセージ解析を開始することをお勧めします。
   - 通知署名 `HTTP` ヘッダーを提供する必要があります。
   - `Subscription-ID HTTP` ヘッダーを含める必要があります。 `GUID` である必要があります。
   - 提供された追加のカスタム情報を検証することが推奨されます。(例: `callbackURL` 内)
   - サブスクライバは、保存された共有秘密を使用してリクエスト本文の署名を計算します。

署名は提供された署名と等しいものとします。

- イベントのサブスクリプションによってコールバックが実行された場合、イベントの発生時刻は過去でなければなりません。 マイナーな時刻同期の問題を考慮して、数秒先になる可能性があります。

3. コールバックが成功すると、204 No Content 応答コードが返されます。

```
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 1261 HTTP/1.1 204 No Content
```

7　付録 A: 例

実際の例の印刷された JSON スキーマ ファイルは、特に使用されるコード リストのせいで、非常に大きくなる可能性があります。 したがって、ここでは例を含めていません。
ただし、例は Web 上の次のアドレスで見つけることができます。
https://github.com/uncefact/spec-openAPI/examples

8 付録 B: 命名および設計ルールのリスト（規定）

ルール #	ルール
[R 1|1]	遵守と適合は、規範的なセクションと規則の内容の遵守を通じて決定されるものとします。さらに、各ルールは次のように分類され、ルールの対象読者を示します。 1. 違反してはいけないルール。 そうしないと、コンプライアンスと相互運用性が失われます。 2. ルールは NDR 構造に準拠しながら変更される可能性があります。 カテゴリ 1 と 2 のすべてのルールに従っている場合、API は完全に準拠しています。 カテゴリ 2 のルールが変更された場合、API は準拠しなくなりますが、依然として準拠しています。
[R 2|1]	この OpenAPI 命名および設計ルールの技術仕様に基づくすべての API 仕様は、OpenAPI 3.1.x 仕様に準拠するものとします（SHALL)。
[R 3|1]	この仕様への準拠を主張する API 仕様は、JSON スキーマの命名および設計ルールの技術仕様 244 に記載されているようにスキーマ コンポーネントを定義するものとします(SHALL)。
[R 4|1]	構造化データ情報の転送に使用されるリクエスト本文のコンテンツとレスポンスのコンテンツは、JavaScript Object Notation（JSON）の application/json メディア タイプを使用するものとします（SHALL)。 このルールは、API が別のメディア タイプの JSON との間の変換サービスを実装する場合にのみ逸脱してもよい(MAY)。 構造化データ情報を転送するために追加のメディアタイプ(例: text/xml)を使用してもよい(MAY)。 非構造化情報が転送される場合、有効なメディアタイプを使用してもよい(MAY)。
[R 5|1]	エンコーディングは UTF-8 でなければなりません。
[R6|2]	API 内で定義されたパスの構造は、消費者にとって意味のあるものでなければなりません(SHOULD)。 理解しやすさ、ひいては使いやすさを高めるために、パスは予測可能な階層構造に従う必要があります。
[R 7|1]	API URL は、次に説明する標準の命名規則に従う必要があります。 `https://{env}.api.{dnsdomain}/v{m}/{service}/{resource}/{id}/{sub-resource}?{query}` コンポーネントは次のように説明されます。 URL の特定のコンポーネントにルールが必須である場合、基本的な URL 構造が上記のものと異なる場合でも（例えば、API が dns ドメインのプレフィックスとして使用されていない場合)、ルールは準拠した API 仕様に適用されるものとします（SHALL)。 ● https:// を Web プロトコルとして使用するものとします。 ● {env} は環境（テスト、サンドボックス、開発など）を示し、実稼働環境では通常省略されます。 ● {dnsdomain} は、API 実装者の DNS ドメイン（例: unece.org）です。

	● {service} は、ビジネス サービス ドメイン（トランスポートなど）を表す API 関数の論理グループです。 {service} コンポーネントはオプションです。 v{m} は、API 仕様のメジャー バージョン番号です。 このコンポーネントは URL に記述されるものとします（SHALL)。 URL の別の場所（ドメインのプレフィックスなど）に指定してもよいです。 ● {resource} は API リソース（委託品など）を表す複数名詞です。 ● {id} は、パスパラメータとして定義されたリソースの一意の識別子です。 パスパラメータはリソースを識別するために使用されるものとします(SHALL)。 リソースのコレクションに対して操作が実行される場合、このコンポーネントはパスの一部ではありません。 ● {sub-resource} はオプションのサブリソースです。 メインリソース（consignmentItem など）にコレクションまたはアクションが含まれている場合にのみ使用されます。 ● {query} は、検索結果を決定するフィルターのような追加パラメータのリストです（例: consignments?loadingPort=AUSYD)。
[R 8|1]	パスとクエリを含む URL の合計文字数は、カンマ、アンダースコア、疑問符、ハイフン、プラスまたはスラッシュなどの書式設定コードを含めて 2000 文字を超えてはなりません。
[R 9|1]	エンドポイントはアクションであってはなりません。サービスとリソースは名詞で構成されます(SHALL)。HTTP 300 動詞はアクションに使用されるものとします（3.2.6 章を参照)。
[R 10|1]	Kebab-case2 はサービスで使用されるものとします（SHALL)。
[R 11|1]	下位の CamelCase3 は、リソース、パスパラメータ、クエリパラメータで使用されるものとします（SHALL)。
[R 12|1]	プロパティ名に関連するパスパラメータとクエリパラメータは、プロパティ名と一貫性があるものとします（SHALL)。
[R 13|1]	クエリパラメータは URL セーフである必要があります。
[R 14|1]	リソース名は複数形にする必要があります。リソース名はスキーマと一致している必要があります（SHOULD)。 スキーマが単数形で定義されている場合でも、リソースは複数形にする必要があります。 リソースの複数形が標準的でない場合は、より適切な複数形の名詞を選択してもよい(MAY)。
[R 15|1]	クエリパラメータは、JSON スキーマ NDR 技術仕様で定義されている UNTDID 2379 json で定義されている ISO8601 準拠の日付と時刻表現を使用するものとします(SHALL)。 特定の日付、時刻、または日時を表すために、形式は日付、時刻、または日時の JSON スキーマ定義に準拠する必要があります（SHALL)。
[R 16|1]	OpenAPI 仕様とその実装がこの NDR TS に完全に準拠するための前提条件は、UN/CEFACT セマンティクスと UN/CEFACT 構文（UN/CEFACT XML、UN/CEFACT JSON スキーマ、UN/CEFACT 語彙など）を使用することです。

	UN/CEFACT 構文または UN/CEFACT セマンティクスを使用しない OpenAPI 仕様であっても、[R 1|1] で指定されている基準を満たしていれば、この NDR TS に準拠している可能性があります。
[R 17|1]	エンドポイントは CRUD 操作をサポートすることが推奨されます。（作成、読み取り、更新、削除)。 エンドポイントがサポートを意図していない場合、例: 削除操作の場合、3.2.10 章で定義されているように HTTP 応答コードを返すものとします（SHALL)。
[R 18|1]	API は、上記のリストで指定された操作の冪等性に従うものとします（SHALL)。
[R 19|1]	API は、POST や PATCH などの非冪等操作をフォールトトレラントにするために、冪等性のキー（Idempotency-Key） HTTP ヘッダー フィールドと対応する実装アドバイスを実装する必要があります（SHOULD)。
[R 20|1]	API でページネーションが使用される場合、キーセットベースのページネーション（カーソルページネーション）が使用されるものとします（SHALL)。 これは、消費者が特定のページをリクエストすることはできず、代わりにサーバーによって提供されるページリンクを選択する必要があることを意味します。 サーバーは、HTTP 応答ヘッダー内で前後のページへのリンクを提供する必要があり(SHALL)、最初と最後のページへのリンクを提供する必要があります(SHOULD)。 さらに多くのリンクが提供される場合があります。 カーソル値は文字列であり、サーバーが任意の方法を使用して作成します。 これは、特定の時点のフィルターと並べ替えパラメーターを含むクエリの結果リスト内のポイントを識別します。 したがって、リストはカーソルより前のリストとカーソルより後のリストに分割されます。 オプションで、カーソル「上」に位置する結果が 1 つ存在する場合があります。
[R 21|1]	コレクション結果に対する GET リクエストはページネーションを実装する必要があります（SHOULD)。 エンドポイントで指定されていない場合、デフォルトおよび最大ページ サイズは 100 である必要があります。 結果として生じるページの負荷が大きい場合は、これを小さくする必要があります。 デフォルトのページ サイズはエンドポイントごとに変更できます（MAY)。 消費者はデフォルトのページサイズをオーバーライドできるべきです(SHOULD)。 結果を取得するときに使用されるフィルタ、ソート、および/またはページサイズが変更された場合、ページネーションは最初のページにリセットされるものとします(SHALL)。 次の表で説明されているクエリパラメータが使用され、ルールが適用されます。
[R 22|1]	並べ替えとフィルタリングはクエリパラメータを使用して行われます(SHALL)。 パス パラメーターの使用は、特定のリソースを識別する場合にのみ許可されます。
[R 23|1]	一般的なガイドとして、フィルタリングは大文字と小文字を区別せずに実行する必要があります。 大文字と小文字を区別しないフィルタリングに 600 を選択するかどうかは、明確に文書化されなければなりません（SHALL)。
[R 24|1]	アプリケーションが、論理演算子、あいまい検索、グループ化などを含む、より豊富な検索およびフィルター機能をサポートする必要がある場合、API は、lucene クエリ構文に従ってクエリ文字列を適用してもよい[MAY]。 その場合、通常、フィルタリング パラメー

	タとクエリ パラメータはリクエスト本文で送信されます。
[R 25|1]	並べ替えは指定されたフィールドに限定されるべきです。 ソート方向は省略してもよい(MAY)。 デフォルトの並べ替え方向は昇順です。 コロン `:` は、フィールド名と並べ替え方向を区切るために使用されます。 複数の並べ替えフィールドはカンマ `,` で区切られます。
[R 26|1]	HTTP 応答コードを使用するものとします（SHALL)。 次の表は、準拠 API でサポートされる HTTP 応答コードを定義しています。 列「応答」は、第 0 章で説明されているように、追加のエラー応答ペイロードが返されることが推奨されるかどうかを示します。
[R 27|1]	次の表は、適合 APIs によって特定の HTTP リクエスト メソッドに対してどの HTTP レスポンス コードがサポートされるべきかを定義します。 列の使用は、準拠 API が指定された http 応答コードをどのようにサポートするかを示します。 - `M` コードはサポートされるものとします - `MA` は、転送や処理時間などの理由で応答が非同期で処理されるリクエストに対してサポートされるものとします。その場合、それぞれのリソースを指す Location HTTP 応答ヘッダーが与えられるものとします（SHALL)。 さらに、Retry-After HTTP 応答ヘッダーが返されることが推奨されます（RECOMMENDED)。 - `R` コードをサポートすることをお勧めします。 肯定的な応答のデフォルトの応答コードは**太字**でマークされています。
[R 28|1]	API は、標準化されたエラー処理を可能にするエラー応答スキーマを実装するものとします（SHALL)。 応答では次の JSON スキーマを使用するものとします（SHALL)。 JSON スキーマは拡張される場合があります。
[R 29|1]	次のルールが推奨されます: - 準拠した OpenAPI 仕様の定義は、設計者と開発者の間、および消費者とプロバイダーの間の技術契約とみなされなければならない（SHALL)。 - 開発のための早期コード統合を可能にするために、API 記述を使用してモック API を作成する必要があります（SHOULD)。 - API の動作と意図は、できるだけ多くの情報で説明される必要があります。 - オペレーションでは、リクエスト本体とレスポンス本体の例を提供する必要があります。 - 予期される応答コードとエラー メッセージを完全に提供する必要があります(SHOULD)。 - 既知の問題または制限は明確に文書化されるべきです。 - 期待されるパフォーマンス、稼働時間、SLA/OLA を明確に文書化する必要があります。 - YAML は OpenAPI 仕様でサポートされているファイル形式ですが、JSON 形式を OpenAPI 仕様形式として使用する必要があります（SHOULD)。
[R 30|1]	すべての API はセマンティック バージョン管理 2.0.0 を適用するものとします:

	`MAJOR.MINOR.PATCH` API の最初のバージョンは、メジャー バージョン 1 で始まるものとします (SHALL)。プレリリース バージョン情報とビルド メタデータバージョン情報は、API のバージョン管理で使用してはなりません (SHALL NOT)。
[R 31|1]	すべての API は URI バージョン管理を使用する**ものとします** (SHALL)。 URI の一部として MAJOR バージョンを「`v{MAJOR}`」の形式で含めるものとします (SHALL)。例: https://api.logistics.io/transport/v1/voyages マイナー バージョンとパッチ バージョンは URI で使用してはなりません (SHALL NOT)。
[R 32|1]	API-Version という名前のカスタムヘッダーは、API の応答に追加されるものとします (SHALL)。 これは URI バージョンと一致するものとし、3 つのレベルすべてを示すものとします: `API バージョン: 1.21.5`
[R 33|1]	API-Version カスタムヘッダーをリクエストに追加してもよい(MAY)。 追加する場合は、メジャー バージョンのみを含めるものとします。 `API バージョン: 1`
[R 34|1]	API は、API のベース URI への GET リクエストに対する応答を実装するものとします (SHALL)。 応答では次の JSON スキーマを使用する必要があります:

[R 35|2]	まだドラフト段階にある API は、サンドボックス環境に配置する必要があります。 これは、基本 URL をそれに応じて変更することで実行できます。
	`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]	メジャー リリース内では、下位互換性が壊れてはなりません。
[R 37|1]	API クライアントとサブスクライバーは堅牢である必要があります。 - API リクエストと入力として渡されるデータは慎重に扱ってください。 - ペイロード内の不明なフィールドを許容しますが、後続の PUT リクエストで必要な場合はペイロードからそれらのフィールドを削除しないでください。
[R 38|1]	代替サービスがステータス ACTIVE で実行されるまで、API を DEPRECATED に設定してはなりません。API のルート サービスは、Deprecation ヘッダー フィールド 13 と Sunset HTTP レスポンス ヘッダー フィールド 14 を提供するものとします。Link ヘッダーは Deprecation ヘッダーと組み合わせて追加されるものとします (SHALL)。 ドキュメントへのリンクを提供するものとします(SHALL)。 API の代替バージョンにリンクする 2 番目の Link ヘッダーを追加する必要があります (SHALL)。
[R 39|1]	非推奨のエンドポイントは、OpenAPI 3.0.0 以降に導入された DEPRECATED プロパティを使用して、OpenAPI 仕様に文書化するものとします (SHALL)。

	非推奨のエンドポイントは、非推奨ヘッダー フィールドと Sunset HTTP 応答ヘッダー フィールドを提供すべきです (SHOULD)。 Link ヘッダーは Deprecation ヘッダーと組み合わせて追加されるものとします (SHALL)。 ドキュメントへのリンクを提供するものとします(SHALL)。 可能であれば、通信は非推奨のエンドポイントの消費者に送信されるべきです (SHOULD)。
[R 40|1]	メジャー バージョンの導入は可能な限り避けるべきです。 これは次のようにして達成できます (MAY)。 - プロセスが変更された場合は、新しいサービス エンドポイントを作成します。 複製と非推奨: ドキュメントと新しいサービスへのリンク ヘッダーを含む非推奨ヘッダーを古いサービスに追加します。 最終的には Sunset ヘッダーを追加します。 - 古いリソースに加えて、新しいリソース (古いリソースのバリアント) を作成します。
[R 41|2]	新しいリソースまたはサービス エンドポイントは、マイナー リリース中に追加できます。 これらの新しいサービスの実装をサポートするために、関心のある消費者または影響を受ける消費者にサンドボックス環境を提供する必要があります (SHOULD)。
[R 42|1]	利用できる並列メジャー バージョンは 3 つまでにすることが推奨されます。 API の実装者は、最新バージョンよりもメジャー バージョンが 1 つ以上遅れてはなりません (SHALL)。
[R 43|1]	API コンシューマにより良いエクスペリエンスを提供するために、API は各リソースで利用可能な状態遷移のリストを提供すべきです (SHOULD)。 リンク関係タイプの可能な値として、公式 IANA レジストリ リスト 16 を使用するものとします (SHALL)。 延長される場合があります。 拡張はすべて API 仕様に文書化するものとします (SHALL)。
[R 44|1]	すべての API エンドポイントはセキュリティで保護されなければなりません (SHALL)。 HTTPS を使用するものとします。 ・動物でみる動物の心理占い無料サイト結果 1077 他のセキュリティ方式を使用してもよい(MAY)。 サブスクリプション コールバックの受信側のエンドポイントは、6.3 章で説明されているような異なるセキュリティ手段を使用して設計されてもよい(MAY)。 API セキュリティの次の側面を実装することが推奨されます。

付録　C: 用語集

用語	定義
ABIE	集約ビジネス情報エンティティ –「委託品」などの情報クラスを表す CCTS の用語
API	アプリケーション プログラミング インターフェイス – マシン間のインターフェイスを指す用語。
ASBIE	Association Business Information Entity– ソース ABIE からターゲット ABIE への有向関係を定義する CCTS の用語 – 例: 「荷受人」と「当事者」の関係
B2B	ビジネスツービジネス
BBIE	基本的なビジネス情報エンティティ – party.name などのクラスのプロパティを説明する CCTS の用語
BRS	ビジネス要件の仕様
CamelCase	キャメルケースは、複数の単語で構成される識別子の技術的表現の命名規則です。空白は削除され、新しい単語はすべて大文字で始まります。 例: この識別子は、キャメルケースで `this Identifier` と記述されます。
CCL	コアコンポーネントライブラリ
CCTS	Core Component Technical Specification – 情報管理メタモデルを説明した UN/CEFACT 仕様文書。
CDT	Core Data Type - 「テキスト」や「コード」などの単純なタイプの BBIE の値ドメイン
HATEOS	Hypermedia as the Engine of Application State - アプリケーション状態のエンジンとしてのハイパーメディア
IETF	インターネットエンジニアリングタスクフォース
JSON	JavaScript Object Notation – API 用に Web 開発者によって一般的に使用される IETF ドキュメント構文標準。
JSON-LD	JSON-Linked Data – リンクされたデータ グラフ/セマンティック語彙の JSON 標準。
Kebab-case	Kebab-case は、複数の単語で構成される識別子の技術的表現の命名規則です。 ハイフンは単語を接続するために使用されます。 例: この識別子は、kebap-case では this-identifier として記述されます。
NDR	Naming & Design Rules – 命名と設計ルール ある表現（例: RDM）を別の表現（例: JSON-LD）にマッピングするための一連のルール
Open API	RESTful API へのオープンソース標準の、言語に依存しないインターフェイス。
OWL	Web Ontology Language　Web オントロジー言語
RDF	Resource Description Framework – リソース記述フレームワーク – W3C セマンティック Web 標準
RDM	参照データ モデル - UN/CEFACT セマンティック出力。

RESTful API	「REST API」を参照
REST API	Representation State Transfer Application Programming Interface - 表現状態転送アプリケーション プログラミング インターフェイス（別名 RESTful API）
RFC	Request for Comments - コメントの要求
SDO	Standards Development Organisation- 標準開発組織
UN/CEFACT	United Nations Centre for Trade Facilitation and Electronic Business - 国連貿易円滑化・電子ビジネスセンター
UNECE	United Nations Economic Commission for Europe - 国連欧州経済委員会
URI	Uniform Resource Identifier 統一リソース識別子 - リソースを明確に識別する名前空間修飾された文字列。 AURL は URI の一種です。
URL	Uniform Resource Locator – リソースの Web アドレス。
UNTDID	United Nations Trade Data Interchange Directory - 国連貿易データ交換ディレクトリ
XML	Extensible Markup Language - 拡張マークアップ言語
XMI	XML Metadata Interchange - 異なるツール間で UML モデルを交換するための確立された OMG 標準。