Specification

# Specification

The [OSDM specification](./) is structured into three parts sharing common data structures:

## Common Data Structures

The [Common Data Structures](./common-data-structures/) are shared between the Offline Model and the Online API.

## Part I: An Offline Model

The [Offline Model](./offline-model/) defines how the offline data can be exchanged.

## Part II: An Online API

The [Online API](https://app.swaggerhub.com/apis-docs/schlpbch/uic-90918_10_osdm/1.0.0) is explained in this documents:

* [Technical Principles](./technical-principles/)
* [Processes](./processes/)
* [Models](./models/)

## Catalog of Code Lists

A [Catalog of Code Lists](./catalog-of-code-lists/) defines attributes and provides references to standards used.

## Compliance to the Specification

In order for an implementation to be compliant, depending on its role, a set of service need to be implemented as lined out in the [Compliance](./compliance/) section.

## Availability as UIC Leaflet

The specification is also available as [IRS-90918-10 Leaflet](../docs/IRS-90918-10-V2020.pdf).

## Common Data Structures in Offline and Online Mode

The following chapters contain the detailed description of data structures used to describe fares.

The data structure definitions are used in the bulk data exchange and the online services. The requirements listed in chapter “Requirements” reference the data structures that implement the requirement.

### General

The following general data types shall be used:

* DateTime Formats: Date time values must be encoded according to RFC 3339, section 5.6.
* Station Codes: Station codes must be taken from the MERITS code list.
* Station Names: Station names should not include ”/”,”\*”. These characters are used to define routes and alternative routes in route descriptions.

### Versioning

The specification (open api specification and schema files for offline data) are published as mayor versions in case they are not interoperable. Minor versions will include interoperable changes on the data structure which also includes providing additional data elements that are optional. Implementers must be able to ignore additional elements.

Minor minor versions will include additional documentation only.

### Indication of personal data

Within the online part the required personal data are indicated. The general grammar to indicate required data is used.

### Indication of required data

Required data are indicated in a structured way using the following language: Data elements are indicated by their path to the resource separated by dots:

* passenger.gender
* passenger.email
* passenger.phoneNumber

The required data elements can be combined using the logical operators:

* AND
* OR

Brackets ( and ) can be used in the standard way as for logical expressions, e.g.: passenger.gender AND (passenger.email OR passenger.phoneNumber)

#### Detailed data structures

The data structures to be used are defined in the schema and open api specification files. This section serves as additional documentation only.

### AfterSalesRules

After sales conditions define fees to be taken in case of an after sales transaction on behalf of a customer. The after sales transactions considered are:

* Cancellation (= Refund)
* Exchange with a new fare of the same carrier
* Exchange with a new fare of another carrier
* Upgrade

See code list: TransactionType

The after sales rules might include rules for a delayed payment to avoid fraud. This might depend in the type of fulfillment. (e.g. no cash refund on electronically payed tickets, no refund unless ticket control data have been received, …).

The refund fee can be claimed by the carrier.

The after sales rules bundle a set of after sales conditions under an id that can be referenced by a fare.

An after sales condition applies for a set of after sales transactions and specified:

* the fee to be applied
* the time when the fee needs to be applied
* whether the fee needs to be given to the carrier or can be kept by the allocator
* The data include the amount to be refunded. The amount is given to avoid any calculations with complex rules (percentage + minimum / maximum value) at the allocator side.:
* The value and currency to be applied
* A percentage for customer information. Due to rounding errors a calculated percentage could result in strange numbers (e.g. 9.99% instead of 10%)
* The unit on which the value is calculated (travellers or bookings) The time when the fee needs to be applied is defined by:
* The time unit (hours, minutes, …)
* The time difference value
* The time reference (before departure…)

See code lists: TimeReference, TimeUnit

An after sales fee is applied from a time before departure, after sale,..)

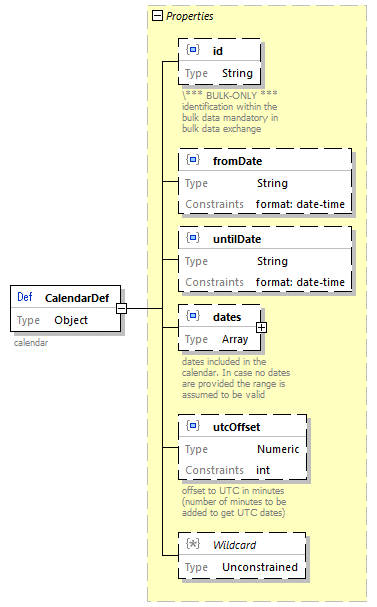
In case multiple rules apply to the same after sales transaction the rule with the closest time in the future must be applied.

#### Data Constraints on AfterSaleRule

|  |  |
| --- | --- |
| Code | Description |
| fee/feeRef | In online services a fee is included directly, in bulk data exchange a fee must be included in the list of prices and referenced by an id. The fee provided must include the currency EUR if not agreed bilaterally otherwise. |
| applicationTime / applicationTimeStamp | An application time stamp can be used in online services only. If an application time stamp is provided the allocation Time as relative time must not be included. |

### Calendar

A Calendar is referenced by a unique id which can be referenced from other data structures linked to the fare. A Calendar defines a list of days between two dates. If the dates are not provided in UTC the offset to UTC must be provided additionally.



Calendar

#### Data Constraints on Calendar

|  |  |
| --- | --- |
| Code | Description |
| fromDate, untilDate | fromDate and untilDate must be provided and fromDate <= untilDate |
| dates | fromDate <= date <= untilDate |

### CarrierConstraint

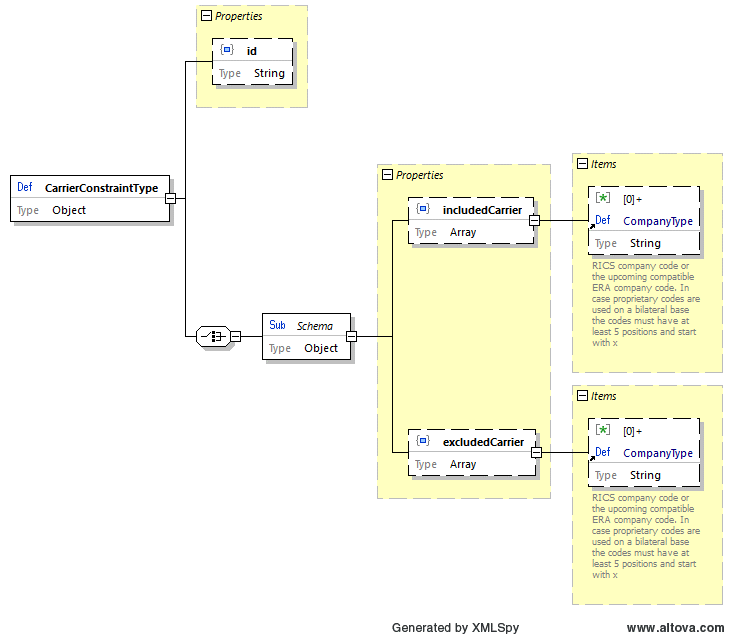
The carrier constraint can be referenced by a fare via the id.

Carrier constraint limits an open fare - not linked to a train - to some carriers. The carriers can be specified either as exclusion list or alternatively as inclusion list.

Carriers are specified by their Company code (RICS code).

The included / excluded carriers are also part of the FCB barcode (*IRS 90918-4*) content and the ticket control data (*IRS 90918-9*).

The offline data structure includes an additional id to reference the constraint within a fare data delivery.



Carrier Constraint Type

#### Data Constraints on CarrierConstraint

|  |  |
| --- | --- |
| Code | Description |
| includedCarriers/excludedCarriers | Either a list of included or a list of excluded carriers must be provided. It is not allowed to provide both lists. |

### ConnectionPoint

A connection point defines a point where two regional validities of different carriers can be connected. A connection point is implemented as the list of stations which hit connects.

In case a route ends at a real station the connection point includes the real station.

In case the combination is not at a real station an indication is needed to define the allowed combinations. This could be done by listing the next stations of other carriers which would allow a combination. Combinations would be allowed if the combination points of two routes share two common stations.

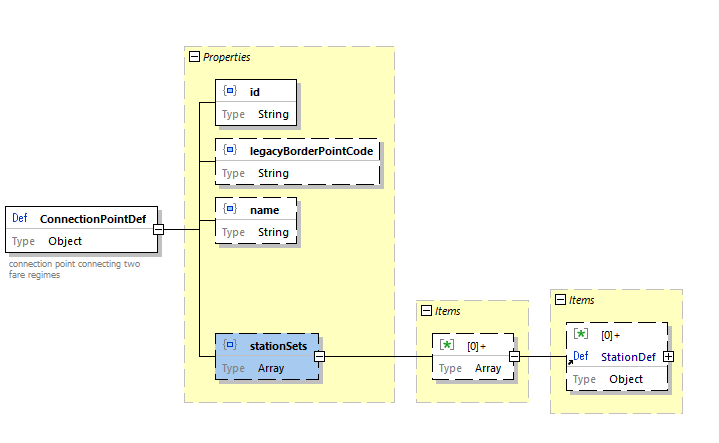
This would also work with multiple stations.

Connection points will include a border point code to support existing implementations where the border point code is compared with the timetable data. As in principle every station can become a connection point (e.g. all stops from Aachen to Brussels are connection points from DB to SNCB) implementations based on border point codes cannot cover all connections.

As on both sides of a connection multiple small stations could be connected and not all of them might be in the timetable of a train the connections point should allow to connect sets of stations.

1. Two fares can be connected in case their connection points share a common station in the provided station sets if only one set is provided by a connection point.
2. Two fares can be connected in case their connection points share a common station in two if the provided station sets of each connection point.

The online data structure does not include the id and the legacy code.



Fare Connection Point

#### Data Constraints on ConnectionPoint

|  |  |
| --- | --- |
| Code | Description |
| stationSets | At least one set with one station must be provided in case the fare border is a real station. Two station sets must be provided in case the fare border is between two real stations. |
| legacyBorderPointCode | The legacy border point code must be provided for the time being. New implementations should not use the border point code. |

### Fare

An elementary fare to create an offer linking all constraints to one price.

|  |  |
| --- | --- |
| Data elements | Description |
| fareType | **NRT**, **IRT**, **Ancillaries** , **Reservations** |
| name | Name of the fare |
| fareDetailDescription | Additional explanation on the fare (e.g. on included fees like Diabolo or Venice fee). |
| price | Price with currency € must be provided if not otherwise agreed bilaterally. |
| regionalConstraint | Definition of the regional validity of the fare and the geographical combination rules (connection points). |
| serviceConstraint | Restrictions of the service allowed to be used. |
| carrierConstraint | Restriction on the carriers that can be used with the fare. |
| serviceClass | Class the passenger can use. |
| serviceLevel | Mode detailed category of places the passenger can use. |
| passengerConstraint | Rules and restrictions on the passenger types allowed to use the fare and rules on combining passengers. |
| afterSalesRules | After sales rules for the fare. In case the allocator is responsible for the after sales rules this is almost empty. |
| combinationConstraint | Rules on the model of combination of this fare with fares of other carriers. |
| fulfillmentConstraint | Restrictions and requirements on the fulfillment and security to be applied by the allocator. |
| reductionConstraint | Rules on reduction cards necessary to apply the fare. |
| reservationParameter | Information on parameters for reservation via the *IRS 90918-1* interface and reservation options. |
| regulatoryConditions | Legal regimes to be applied to the fate (e.g. COTIV, SMPS regulations). |
| personalDataConstraint | Rules on the personal data to be provided in a booking. |
| legacyAccountingIdentifier | Data to be included in the current *IRS 30301* accounting data format. |
| salesAvailabilityConstraint | Rules on the allowed sates dates for the fare. |
| travelValidityConstraint | Rules on the validity for travel of this fare. |
| legacyConversion | Defines whether this fare is allowed to be converted to the old 108.1 data structure and used according to the old rules (YES, NO, ONLY (this fare is provided for conversion only)). |

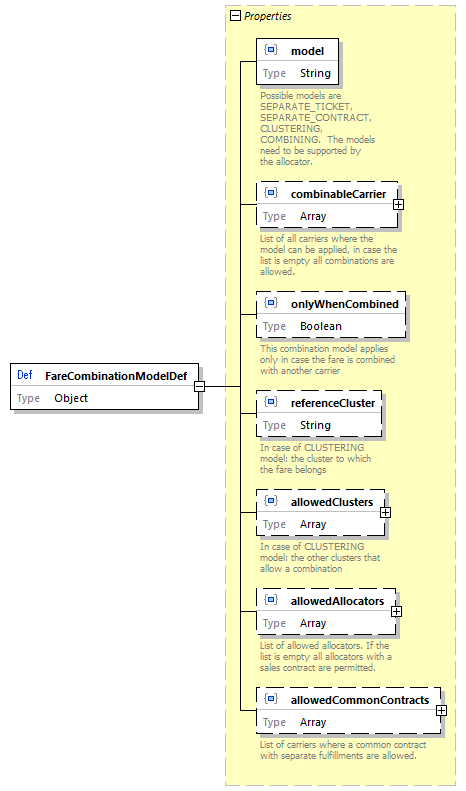
#### Data Constraints on Fare

|  |  |
| --- | --- |
| Code | Description |
| price | A price must be provided for all offline fares including those where the price is zero. |
| legacyAccountingIdentifier | In case *IRS 30301* in the current version is used to accounting these data must be provided for offline fares |
| serviceClass | Must be provided for offline fares |
| combinationConstraint | Must be provided for offline fares |
| travelValidityConstraint | Must be provided for offline fares |
| salesAvailabilityConstraint | Must be provided for offline fares |

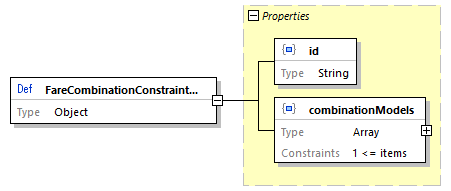
### FareCombinationConstraint

The fare combination constraint defines the rules of combining fares from different carriers. It provides a list of combination models the allocator can choose of.

|  |  |
| --- | --- |
| Content | Description |
| model | Code of the combination model applied |
| combinableCarriers | List of carriers that can be combined with this fare. If empty, there is no restriction in combining different carriers. Carriers are listed by their RICS company codes. |
| onlyWhenCombined | Indicates that this fare can be used only if it is combined with another fare of another carrier. |
| referenceCluster | Cluster within the clustering model to which this fare belongs |
| allowedClusters | List of clusters with which this fare can be combined |
| allowedAllocators | List of allocators which can combine this fare. . . If empty, there is no restriction in combining different carriers. Carriers are listed by their RICS company codes. Allowed allocators is not present in the online data. |
| allowedCommonContracts | List of Carriers with which the allocator can for a common contract. If empty, there is no restriction in indicating common contracts to the passenger except for the SEPARATE\_CONTRACT model. Carriers are listed by their RICS company codes. |



Fare Combination Model



Fare Combination Constraint

#### Combination Model

##### SEPARATE\_CONTRACT Model

This SEPARATE\_CONTRACT model is the model for not combining the fares in one ticket and not allowing the integration in one contract. The rules applied for this ticket are exactly the rules defined by the carrier in the fare data.

The allocator must ensure that it is clear for the customer that no common contract was established.

##### SEPARATE\_TICKET Model

This SEPARATE\_TICKET model is the model for not combining the fares in one ticket, so the rules applied for this ticket are exactly the rules defined by the carrier in the fare data. The allocator can form a common contract for the whole journey.

##### CLUSTERING Model

The CLUSTERING model tries to simplify conditions and fares for the customer but sacrifices a part of the control of the carrier on his fares.

Similar types of fares are defined to belong to the same cluster. The after sales conditions for a cluster are defined by the allocator. However, the after sales conditions must basic rules on after sales for that cluster.

The clusters correspond to the flexibility a passenger receives to change the booked train. This corresponds directly to the after sales conditions. Hereby the fees to be paid for such an exchange are essential for the definition of clusters and not the complexity of the process to change. Thus, a train bound ticket and an open ticket belong to the same cluster in case the fees to change to different trains / times are comparable.

The after sales fees can be demanded by the carrier.

The other conditions might either be listed per carrier or combined by rules.

The customer buying products from one allocator has a simple unique view on after sales conditions.

The basic parameters defining the price must be obeyed individually within separately on the combined fare/offer:

* route description / train link
* class of service
* passenger types

##### COMBINING Model

The COMBINING model tries to be close to the fare conditions defined by the carrier but sacrifices the simplicity of the fare towards the customer.

The after sales conditions of the different fares will be combined into one condition to reflect the conditions of all included carriers.

The after sales conditions will thus depend on the combinations of carriers.

At any time, the after sales fees defined by the carriers are applied on the price part of these carriers only. The result is a list of times with increasing fees.

###### Example

* Carrier 1: 10% 20 days before departure. Price: 100€
* Carrier 2: 90% 2 days before departure. Price: 200 €
* Result:
  + 10€ fee: 20 days before departure
  + 190€ fee: 2 days before departure

#### Additional Clustering Model Data

Fare clusters reflect the flexibility a fare provides to the customer. Flexibility is defined by the after sales conditions that apply when a passenger wants to change his ticket.

|  |  |
| --- | --- |
| Fare cluster code | description |
| BUSINESS | Refundable/Exchangeable after the departure or last day of validity |
| FULL\_FLEX | Refundable/Exchangeable before the departure or last day of validity |
| SEMI\_FLEX | Refundable/Exchangeable with fee depending on conditions of the allocator. Minimum validity applies |
| NON\_FLEX | Non refundable. Non exchangeable. Minimum validity applies |
| PROMO | Used on a bilateral basis only. Non refundable. Non exchangeable. Minimum validity applies |

Combinations of fares of different clusters is allowed with the fare clusters listed in allowedClusters. However not all combinations would be provided to the customer. A fare will be combined with a fare of the same cluster and in case his is not available with one of the higher clusters.

##### Example

* Carrier 1:
  + BUSINESS -> CombinableClusters: BUSINESS, FULL\_FLEX, SEMI\_FLEX, NON\_FLEX
  + SEMI\_FLEX -> CombinableClusters: SEMI\_FLEX, NON\_FLEX
* Carrier 2:
  + BUSINESS -> CombinableClusters: BUSINESS, FULL\_FLEX, SEMI\_FLEX, NON\_FLEX
  + FULL\_FLEX -> CombinableClusters: FULL\_FLEX, SEMI\_FLEX, NON\_FLEX

Possible combined offers are:

* BUSINESS (Carrier 1 BUSINESS + Carrier 2 BUSINESS)
* FULL\_FLEX (Carrier 1 BUSINESS + Carrier 2 FULL\_FLEX)
* SEMI\_FLEX (Carrier 1 SEMI\_FLEX + Carrier 2 FULL\_FLEX)

A NON\_FLEX would be formally allowed, but with the same price as the SEMI\_FLEX so it should not be shown to the customer:

* NON\_FLEX (Carrier 1 SEMI\_FLEX + Carrier 2 FULL\_FLEX)

Other combinations would also be formally allowed by the data but suppressed as they would only offer a higher price. These should be suppressed by the allocator. E.g.:

* FULL\_FLEX (Carrier 1 BUSINESS + Carrier 2 BUSINESS)

#### Data Constraints on FareCombinationConstraint

|  |  |
| --- | --- |
| Code | Description |
| combinationModel | At least one model must be provided |

### FareResourceLocation

Fare resource location provides data on where to find online services for fares. The fare location provides three options:

* Link a resource to a carrier – the carrier must be known from the timetable
* Link a resource to the train – the data must be updated in case of new trains
* Link a resource to stations:
  + The link can be made for stations and for connection points
  + The link is valid if start and end station (or connection points) provide the link

The online link provides information on:

* The type of resource either for a whole train of an area. In case of a train the request must be for the train route between stations (e.g. IRT), whereas for areas there might be multiple splits in-between a train run (e.g. NRT).

#### Graphics Icons

Graphic icons are used to display a coach including its facilities based on the coach layout and availability of places. The graphical items include frames and icons to display seats etc. Graphical items must be provided by the sales application of the issuer application to ensure a unique look and feel of the application.

The coach layout provides only the position of graphic items (co-ordinates) not the graphical presentation at the sales application (pictures).

A large table spans two places, whereas a small table spans only one place. A small wall spans two places and a large wall spans 3 places. A very small wall spans one place only.

#### Data constraints on FareResourceLocation

|  |  |
| --- | --- |
| Code | Description |
| System | For reservation interface 90810-1 the reservation system code is used. |

### FareReferenceStationSet

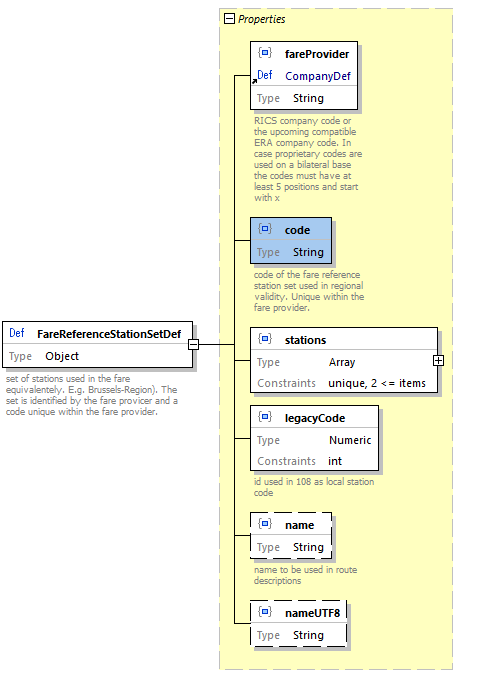
The fare reference station set defines a set of stations where the fare is valid for all included stations. This set can be used in the regionalValidity description.

The corresponding bar code ab ticket control data will only contain the code of the station set, but the allocator needs the complete list of station to link the fare to the train routes.

A name can be provided.

The station set is referenced by the company code of the fare provider and a code unique within the fare provider.

A legacyCode can be provided to include the current code in the 108.1 data.



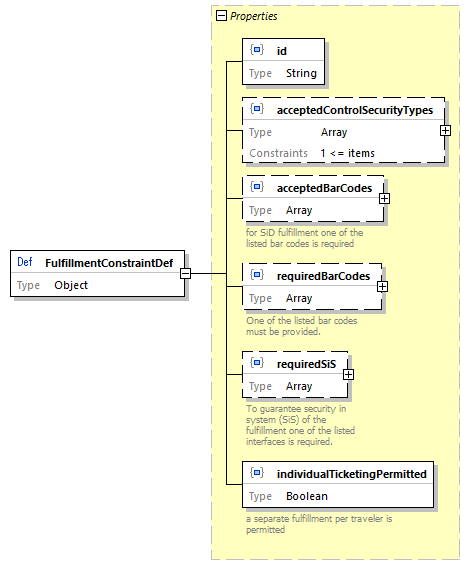
Fare Reference Station Set

#### Data Constraints on FareReferenceStationSet

|  |  |
| --- | --- |
| Code | Description |
| legacyCode | A legacyCode must be provided for the time being. New implementations should not rely on that code. |
| name | The name should not include ”/”.”\*”. |

### FulfillmentConstraint

The fulfillment constraint limits the applicable types of fulfillment and defined whether control data need to be transferred via a standard interface (*IRS 90918-4*).



Fulfillment Constraint

Code lists for required SiS: CardType

The following code list defines the card types for cards used

|  |  |
| --- | --- |
| Predefined Card-Ids | Description |
| LOYALTY\_CARD | Loyalty card |
| REDUCTION\_CARD | Card providing reduction |
| PASS | Pass for travelling |

ControlDataExchangeType:

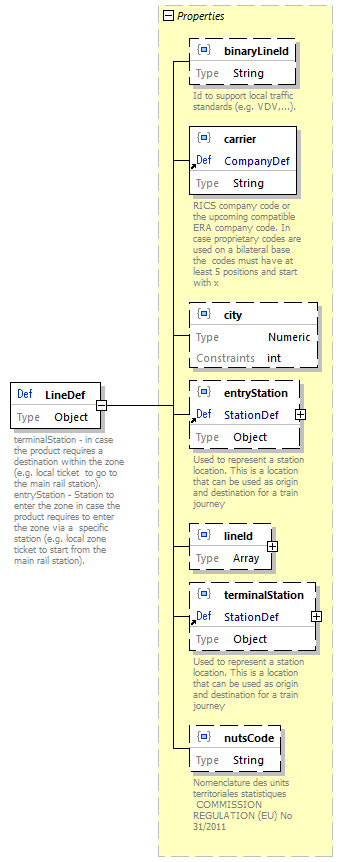
* for bar codes: BarcodeType
* for fulfillment: ControlSecurityType

#### Data Constraints on FulfillmentConstraint

|  |  |
| --- | --- |
| Code | Description |
| acceptedFulfillmentType | At least one accepted fulfillment type must be provided |

### Line

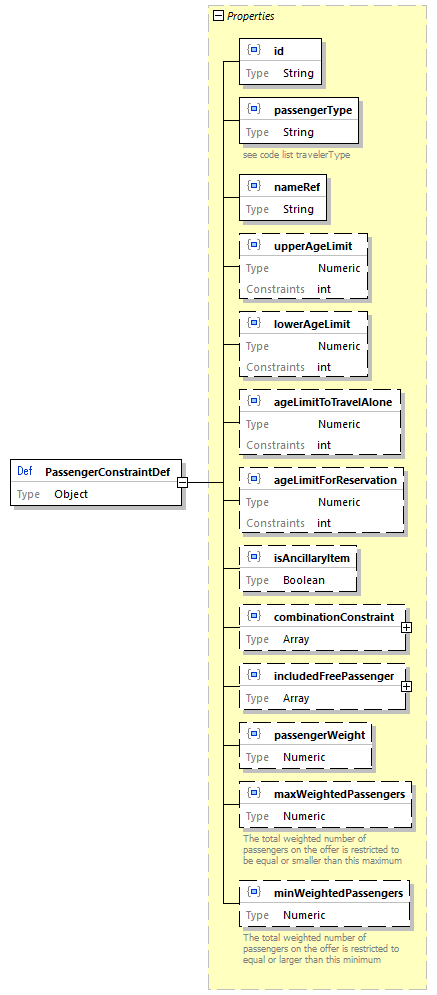
Line defines the regional validity on a specific line. It might have additional restrictions to enter or leave at specific stations or to be used within an area or city only.



Line

### PassengerConstraint

Passenger constraint defines restrictions of a fare concerning passengers. In online services the structure is reduced to constraints that need to be passed on for control to bar codes and control registries.



Passenger Constraint

#### Data Constraints on PassengerConstraint

|  |  |
| --- | --- |
| Code | Description |
| upperAgeLimit, lowerAgeLimit | upperAgeLimit >= lowerAgeLimit |

### PersonalDataConstraint

Specification of personal data to be delivered to the carrier. Personal data might be included in:

* Booking service (OSDM and/or *IRS 90918-1*)
* Control data (bar code and/or control data delivery *IRS 90918-4*)

The requirement for personal data might depend on the type of fulfillment or on specific border crossings.

|  |  |
| --- | --- |
| Code | Description |
| acceptedReason | Accepted reason to change personal data after booking confirmation. See code list: Personal data change reasons |
| transfer | The way the personal data are transferred. See code list: Personal data transfer types |
| ticketHolderOnly | Personal data are required for the ticket holder only |
| dataItem | Code of the data item required. Consists of languageCode, overruleCode (see below) and personal data items. |

#### Overrule Code

|  |  |
| --- | --- |
| Code | Description |
| STRIKE | Refund due to strike |
| SALES\_STAFF\_ERROR | Refund due to an error made by the sales staff |
| PAYMENT\_FAILURE | Refund as the payment failed |

### Price

The price data structure provides the price or a fee including the VAT details optionally in different currencies.

Scope: see code list TaxScope

#### Data Constraints on Price

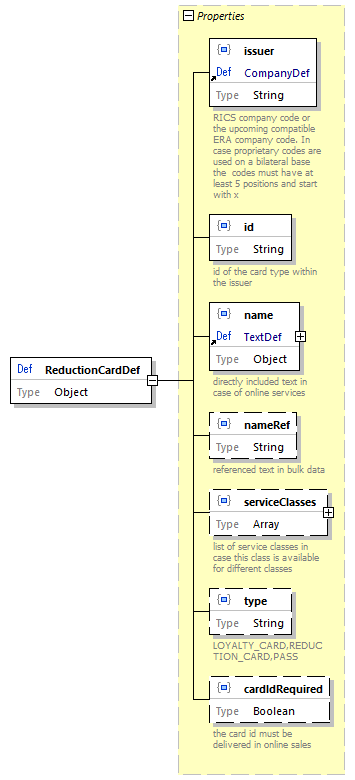
|  |  |
| --- | --- |
| Code | Description |
| amount | Amount >= sum of VAT-amounts |

### ReductionCard

The reduction cards of a carrier are listed in the bulk data.

#### List of Carrier Cards

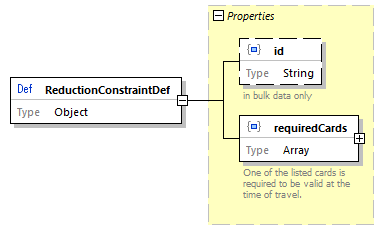
|  |  |
| --- | --- |
| Code | Description |
| id | Unique id of the card. The id must start with the RICS code of the carrier |
| name | Name and short name of the card. The name should be used for the card selection by the customer, the short name should be used for bar codes.Usually the card name is not translated, but the card name might be provided in different languages by carriers in multilingual countries. |
| serviceClass | Service class indicated for the class |
| issuer | Issuer of the card. Usually the carrier providing the fare data. |
| type | Type of the cards to separate between loyalty cards, cards that are tickets (passes), and reduction cards (LOYALTY\_CARD, REDUCTION\_CARD,PASS). |
| cardIdRequired | Indicates that the card id must be provided in the pre-booking request to validate the card. This card cannot be used without the online services for booking |



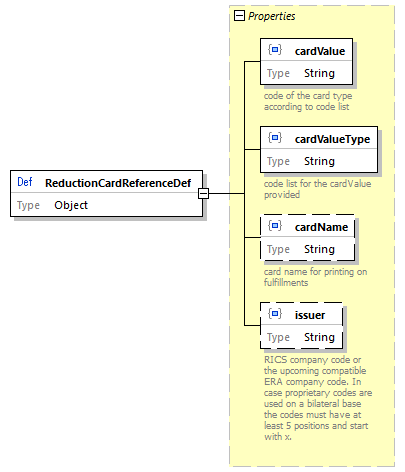
Reduction Card

### ReductionConstraint

A fare associated with this constraint requires one of the listed cards to be presented by the passenger on the trip. Card Ids can be taken from the listed cards provided within the fare data delivery or from the common code list in Reduction “cards”.



Reduction Constraint



Reduction Card Reference

### RegionalConstraint

Definition of a regional validity of a fare. The regional validity constraint is defined by an entry connection point and an exit connection point to combine this regional validity with other regional validities of other carriers and the specification of the regional validity that is sued and described in *IRS 90918-4* for ticket control. The entry or exit connection point might be missing in case the fare cannot be combined or can be combined on one side only.

|  |  |
| --- | --- |
| Content | Description |
| entryConnectionPoint | Defines the connection point for connecting this fare at the start of regional validity (see ConnectionPoint) |
| exitConnectionPoint | Defines the connection point for connecting this fare at the start of regional validity (see ConnectionPoint) |
| regionalValidity | Definition of the regional validity as defined in *IRS 90918-4*. It provide data structures for zones, Lines, train links, geographical polygons and routes. |

The connection points are included for combining regions. When combining two regional validities from two carriers the connection points will disappear in the combined data structure for bar codes and ticket control and from the textual description for the passenger.

E.g.:

* Carrier 1: RegionalConstraint {Exit (A,B), RegionalValidity X – Y/Z- A}
* Carrier 2: RegionalConstraint {Entry (A,B), RegionalValidity B – C/D – E}
* *Result*: X*Y/Z*A*B*C/D\*E

The allocator might need to remove doubled stations in routes in case the connection point is a real station used in both regional validity descriptions in case it is displayed as one combined text:

* Carrier 1: RegionalConstraint {Exit (A), RegionalValidity X – Y/Z- A}
* Carrier 2: RegionalConstraint {Entry (A), RegionalValidity A – C/D – E}
* *Result*: X*Y/Z*A*A*C/D*E –> X*Y/Z*A*C/D\*E

#### Connecting Regional Validity to Trips

The regional constraint is connected to the timetable via the regional validity, the connection points are used to combine regional constraints.

To support legacy implementations the connection points can provide a border point code linked with the timetable.

The online data structure will not use the id and will directly include the entry and exit connection point, whereas the offline structure will include the id of the connection point pointing to a connection point within the same data delivery.

The regional validity contains also content that is applicable to synchronous data transfer only (e.g. train links for train bound offers).

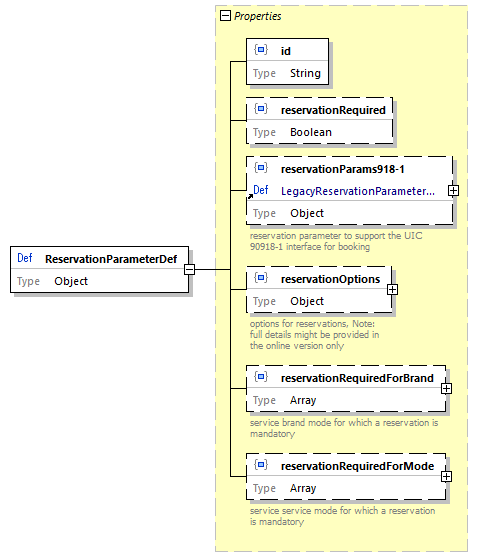
The data structure RegionalValidity is defined in *IRS 90918-4* and included by reference only. It provides a sequential list of region definitions that can be defined as zones, lines, train links (online version only) geographical areas (polygons) and route descriptions (via-stations). The route description is extended to include fare reference station sets within the route.

Extended route data structure including fare reference station sets.

### ReservationParameter

ReservationParameter provide data on how to combine reservations with NRT fares, how to book reservations via the *IRS 90918-1* interface and which options a passenger has for reservation.

|  |  |
| --- | --- |
| Code | Description |
| reservationRequired | A reservation must be made accompanying an NRT ticket. |
| reservationParameters981-1 | Parameters to request the correct reservation using the interface according to *IRS 90918-1*. |
| reservationOptions | Reservation options available that would not change the offer (same price and conditions) (e.g. Aisle or Window). The information is static and does not mean that such an option is still available. The preferences are grouped in case a selection is required (Aisle or Window). |



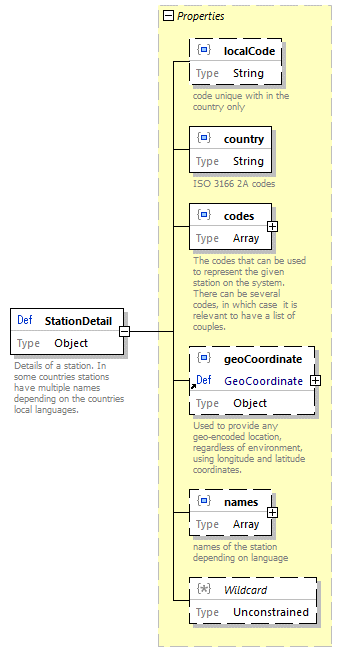
Reservation Parameter

Code Lists

* Code list Preference Groups: see Preference groups
* Code list Preferences: see Preferences of places

### StationDetail

Details on stations including codes and names. Codes must include the MERITS code in case it is defined for a station.

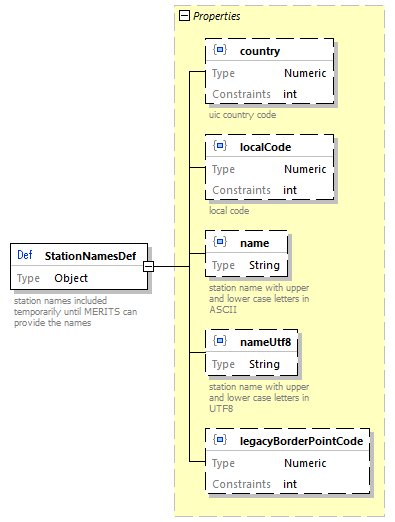


Station Detail

### StationNames

Station names provides multi language names in short and long form as currently no other data source can provide these names. Short names are used within the route descriptions whereas the long for is used for entry and exit stations.

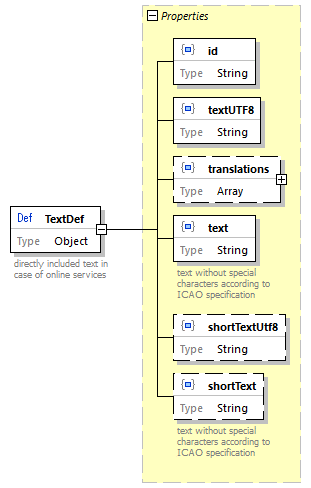
A legacy border point code can be provided during the migration to the OSDM data model.



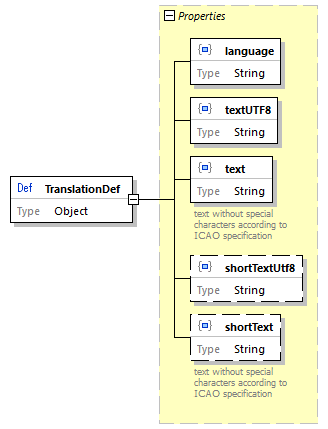
Station Names

### Text

Used for all textual descriptions where translations might be needed.



Text



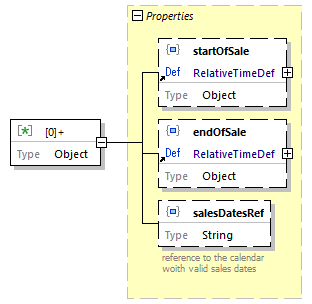
Translation

### SalesAvailability

Sales availability defines the constraints on the time when a sale of a fare can start or end. The sales availability is used in the offline data exchange only. A constraint is provides as a list of salesRestrictions that have to be applied.

Sales restrictions can define a start and end of the sale relative to the date of sale or the date of travel.

A reference to a calendar can be provided to indicate all sales dates.



Sales Availability

#### Data Constraint on SalesAvailability

|  |  |
| --- | --- |
| Code | Description |
| startOfSale, endOfSale | startOfSale < endOfSale |

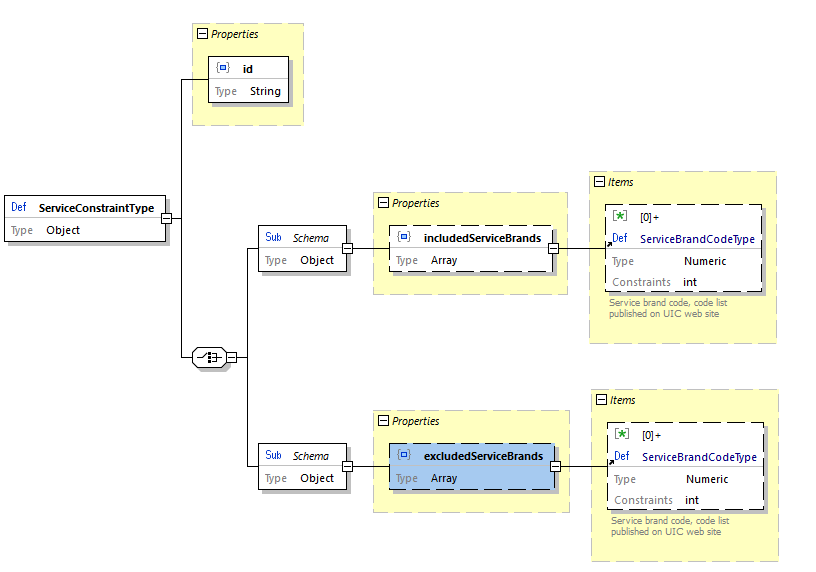
### ServiceClass

Service class provides textual descriptions for the predefined service classes.

### ServiceConstraint

The service constraint limits a fare to specific service brands (train types). The constraint can either be defined as a list of service brands included or as a list of service brands excluded for the fare.

The online data structure will not provide the id.



Service Constraint

#### Data Constraints on ServiceConstraint

|  |  |
| --- | --- |
| Code | Description |
| includedServiceBrands, excludedServiceBrands | Only one of the lists can be used. Using both lists is forbidden. |

### ServiceLevel

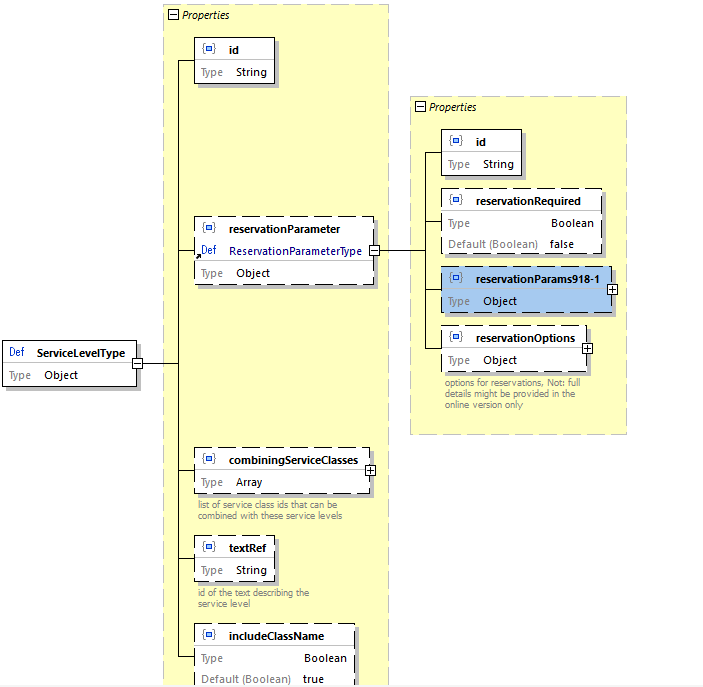
The service level data provide additional information (e.g. text) in the offline data exchange in case the reservation API of *IRS 90918-1* is used.

Description of a service level. The service level defines a specific product on a train which can have a price (e.g. Double places with shower, …). It is more specific than just the classic travel class.

The available service levels are defined in *IRS 90918-1* element 308 (Service level code). The data indicate the service class that needs to be booked in case the reservation is not an IRT and parameters needed for reservation via the *IRS 90918-1* interface.

Some service levels might require a mandatory reservation.

Additional to a service level there might be reservation options that do not affect the price. There are listed in reservation options. (e.g. Upper or lower berth in the service level for double Sleeper compartment).

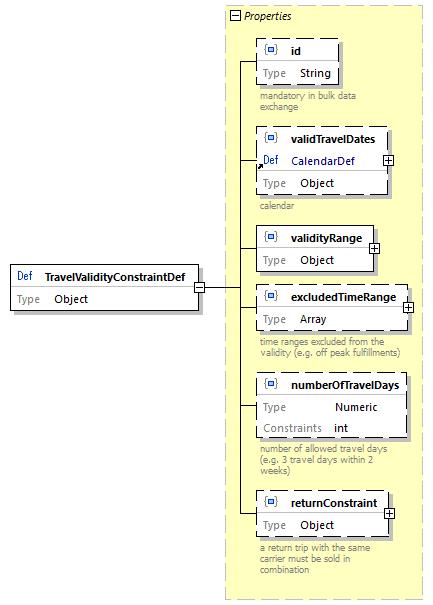


Service Level

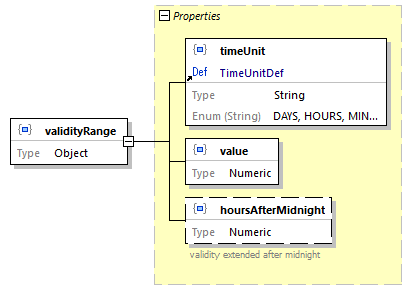
### TravelValidityConstraint

The travel validity constraint defines at which times the passenger is permitted to travel.

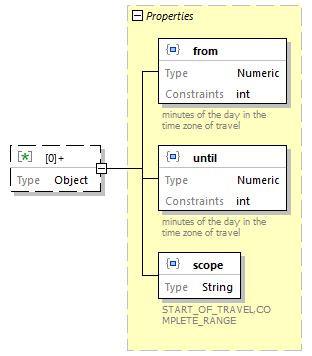
#### Data Constraints on TravelValidityConstraint



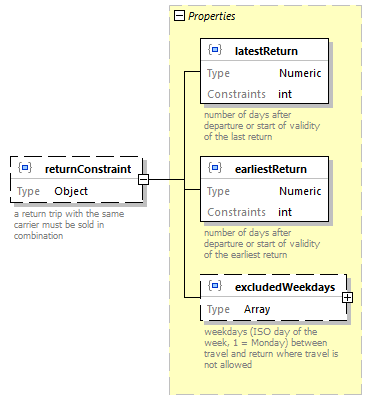
Travel Validity



Travel Validity - validity range



Travel Validity - excluded time range



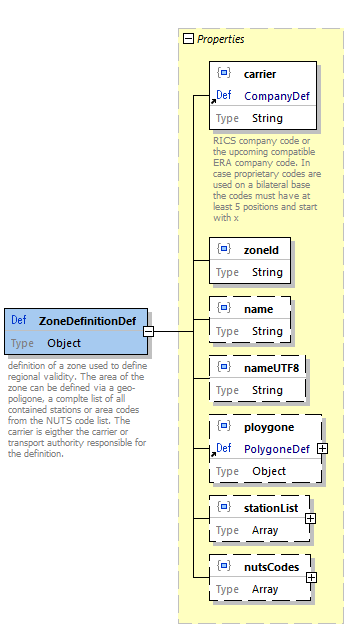
Travel Validity - return constraint

|  |  |
| --- | --- |
| Code | Description |
| excludedTimeRange | from time < until time |
| numberOfTravelDays | A duration must be provided |
| returnConstraint | earliestReturn < latestReturn |

### ZoneDefinition

Definition of zones used in regional validity.

The area of a zone can be defined by either a list of stations, geographical polygon of a list of NUTS codes. Multiple definitions are allowed in case they define the same area.



Zone Definition

## Design Guidelines

* **Do not reinvent the wheel** - Use existing concepts whenever possible (e.g. type system of OpenAPI, Problem details,…).
* Strive for a Level 3 of [REST maturity](https://martinfowler.com/articles/richardsonMaturityModel.html).
* Use [semantic versioning](https://semver.org).

## Derived Guidelines

* Whenever a resource returned in a response can contain embedded resources, the request must allow specifying whether and which embedded resources should be returned in full or as references.
* Follow [Zalando RESTful API and Event Scheme guidelines](https://opensource.zalando.com/restful-api-guidelines/)
* Use of the JSON Problem element
* Standard Patch operations (not JSON PATCH)
* A resource is either represented in full or as a reference. The reference element has the name of the resource post-fixed with “Ref”. References normally only contains the URL to the referenced resource and a title element allowing to summarize the resource in one short string
* Although examples or recommendations are provided as to which information should best be represented in the title string, each implementor as the freedom to modify it to best suit his needs.
* Enumerations for very stable entities with limited set only, otherwise code lists. Stations codes are code lists.
* Where possible, existing UIC code lists should be favored.
* Creation/ modification calls return the created/modified resource (not just an ok code)

## Error Handling

In order to communicate errors to a consumer we support [RFC7807](https://tools.ietf.org/html/rfc7807).

This RFC defines a “problem detail” as a way to carry machine- readable details of errors in a HTTP response to avoid the need to define new error response formats for HTTP APIs.

A problem details object can have the following members:

* type - A URI reference [RFC3986](https://tools.ietf.org/html/rfc3986) that identifies the problem type. This specification encourages that, when dereferenced, it provide human-readable documentation for the problem type (e.g., using HTML [W3C.REC-html5-20141028]). When this member is not present, its value is assumed to be “about:blank”.
* title - A short, human-readable summary of the problem type. It SHOULD NOT change from occurrence to occurrence of the problem, except for purposes of localization (e.g., using proactive content negotiation; see [RFC7231](https://tools.ietf.org/html/rfc7231), Section 3.4).
* status - The HTTP status code ([RFC7231](https://tools.ietf.org/html/rfc7231), Section 6) generated by the origin server for this occurrence of the problem.
* detail - A human-readable explanation specific to this occurrence of the problem.
* instance - A URI reference that identifies the specific occurrence of the problem. It may or may not yield further information if dereferenced.

Consumers MUST use the type string as the primary identifier for the problem type; the title string is advisory and included only for users who are not aware of the semantics of the URI and do not have the ability to discover them (e.g., offline log analysis). Consumers SHOULD NOT automatically dereference the type URI.

## Security and Authentication

These aspects are *not* part of the OSDM specification. However each implementor of OSDM should define its own mechanisms, therefore some strongly recommended issues are described.

OSDM uses RESTful requests/responses, therefore HttpStatus are declared in the API specification.

## Authentication

There are several API-Management (APIM) solutions on the market for e.g.:

* certain cloud-providers offer necessary services already (for e.g. AWS, ..)
* open source or licencable applications to install by your devOps Team (for e.g. 3Scale)
* simple mechanisms like Basic Auth as part of the Swagger/OpenAPI specification (though these might often be not good enough for serious production)

We recommend you to satisfy the following requirements:

* A registration service allows a consumer to register the necessary data (like: Organization, tech. admin, commercial admin, support line, other non-functional requirements like throtteling limits) for approvement by the team providing access. any registered consumer will be approved or rejected if approved a unique, technical “ClientId” results and will be passed to the consumer
* A login service allows to request a valid token (for e.g. OAauth2) by the registered ClientId and related secret per configuration the validity duration of the token may be set (for e.g. 30min.), after that the token must be renewed by the consumer optionally the token might be revoked (for e.g. if your devOps realizes the consumer does not behave as expected)
* A passthrough service expects the token (typically in the header, where the bearer token contains the clientId) for each request made by the consumer
  + validates the token and if ok “redirection” to the underlying system is performed (not OK results typically in any 401 or 403 faults) the APIM should support RESTful service-contracts in relation to OSDM.
* In multi-environments (like DEV, TEST, INT, PROD) consumers might register for each environment separately
  + with different non-functional requirements
  + with different authorization for specific API (if you want to configure that at all)
* Easy testable (for e.g. Postman)

For example at SBB we use [3Scale](https://www.redhat.com/de/technologies/jboss-middleware/3scale) as APIM solution which powers the SBB official service portal [developer.sbb.ch](https://developer.sbb.ch/).

### User Lookup

From a devOps perspective it might by hard to control who uses your API (for e.g. ClientId and secret might by passed around or hacked).

The following issues should be defined outside of OSDM:

* Additional legal contracts with consumer
* IP white listing
* Mapping of related user properties to the technical ClientId

## General Scenarios

Within the offline sales model the participating companies agreed to allow sales based on the provided fare data. The receiving company is responsible to apply the rules defined within the fare data. In case the implementation does not cover some features it is not allowed to sell fares that use these features.

## Fare Exchange

Fares can be exchanged by bilateral file exchange, via a queue provided according to this specification or via some common exchange platform like the upcoming OSDM data exchange platform in case the company is a member of the platform.

Exchanged fare data deliveries can be defined to be implemented mandatory or to be optional thus allowing to continue the sales with the previous version. In case a mandatory version replaces a previous version it also replaces all previous optional version with-in the chain.

A data delivery might specify a minimal version number of the schema that needs to be supported to use the data.

## Versioning of Data Deliveries

The data delivery will contain the version number and the version number which is required to process the data. Also, a change in a minor version might restrict the usage of older version in case a carrier used a new optional feature which is mandatory to his fares.

## Automated Bulk Data Exchange

Automated asynchronous bulk data transfer is an option implemented by queues. The queues must implement the [AMQP 1.0 specification](https://www.amqp.org/about/what).

On bilateral agreement other queue technologies might be used between two systems.

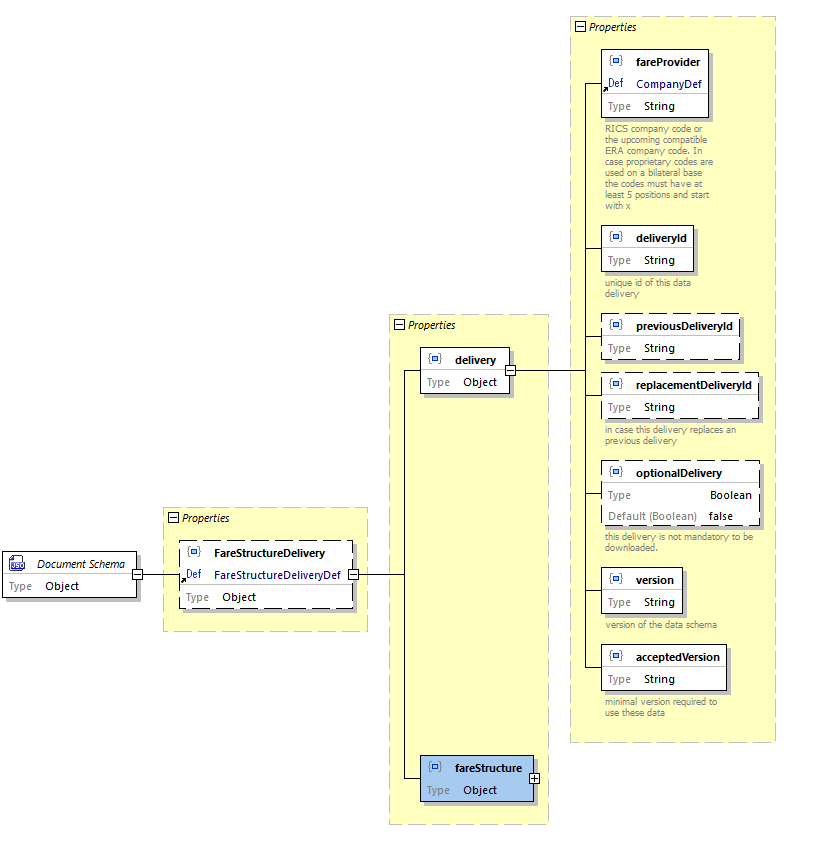
Queue authentication and encryption must use TLS version 1.2.

### AMPQ Header Parameter

|  |  |
| --- | --- |
| Parameter | Usage |
| message-id | Technical id of the data transfer, not the data delivery id in the data. |
| user-id |  |
| to |  |
| subject | „fare-data-delivery\_“<version> |
| reply-to | N/A |
| correlation-id | N/A |
| content-type | application/json |
| absolute-expiry-time | 1 year ahead |
| creation-time | Time stamp when the data are put to the queue |
| group-id |  |
| group-sequence |  |
| reply-to-group-id |  |

## Asynchronous Fare Data Delivery

The fare structure delivery is the bulk data object collecting the fare data fareStructure of a delivery and the delivery meta data delivery.



Data Structure for Bulk Data

## Introduction

This page shows a representation of the data models underlying the API specifications. It is therefore not a strict representation of the resources themselves (those are self-represented in the OpenAPI specifications.)

As such, some of the details of how the information is structured in the API are not represented or simplified in the data models. The main purpose of this data model is therefore to help a quicker understanding of the API and its underlying concepts.

### Legend

Legend

Legend

## Trips and Locations

### Trips and Locations Data Model

Trips and Locations Data Model

Trips and Locations Data Model

## Main Resources

### Locations

Locations are resources representing a specific location in a trip: departure, origin, intermediate stop or other. They can be of different types:

* **Station**: represent a train station. It is obviously the most relevant type for OSDM. Stations can be indicated as codes from different code sets. As with other code list based representations in the API, using the UIC code set is highly recommended.
* **Address**: any street address can be represented here. Is included to foresee extension towards multi modality and first/last miles solutions
* **POI**: used to represent a specific point of interest
* **GeoCoordinate**: allows providing any location on the globe using its geographical coordinates.
* **ConnectionPoint**: allows to model virtual border points by defining stations within the connection point lies.

Locations are modelled in the API as resources with a long time-to-live, which should allow efficient caching of this data, therefore removing the need of getting full location details in transactional operations.

### Trips

Trips represent the concrete realization of a trip going from departure station to destination station. A trip is composed of one or more segments.

Each segment (also sometimes called legs) represent a connection between two locations where the traveller will either step in a transport or step out of a transport (most likely a train). 3 types can be distinguished:

* **origin**: departure location of the segment
* **destination**: arrival location of the segment
* **stops**: intermediate locations encountered between origin and destination of the segment

Regardless of whether the products to travel these segments are train-bound, or based on with a validity period of any duration, segments (and by extension trips) are always train-bound and represent the realization of the travel wish using specific trains at a specific moment in time.

Trips can be retrieved with or without details of all intermediate stops on the way between departure and arrival of each segment

## Offers

### Offers Data Model

Offers Data Model

Offers Data Model

## Main Resources

### Offers

In the **Distributor Mode**, offers represent a collection of OfferParts, representing bookable elements that covers exactly one a specific trip (or the requested section of a trip in case of allocator request to an nTM Provider). Note that the offer contains a reference to the trip resource it is built for, although this reference can be redundant when the TripOffer the offer is part of is already known to the API consumer

Some of these elements can be optional (see further).

In the **Allocator Mode**, an offer will not contain offer parts but it will contain a fare element, that the allocator can use to build the final product to be distributed to travellers. There can also be hybrid situations where offers will contain both offer parts and fares if the provider offers product in the two flavors.

In some cases, API consumers will be required to provide some additional information in order to proceed with the provisional booking. In this case, the information needed will be specified in the requestedInformation element using a notation akin to regular expressions.

Offers should always contain a minimalPrice (= the price of the offer without any of the optional offer parts), a global service class, service level and flexibility. Although the calculation rules for these global values are up to the OSDM provider, the following rules are recommended:

* ServiceClass: the lowest class of a significant offer part should be the service class of the offer (1st class + 2nd class = 2nd class)
* ServiceLevel: the highest service Level should be the service level of the offer  (bed + berth in compartment of 2 + berth in compartment of 4 = berth in compartment of 2)
* Flexibility: the lowest flexibility of a significant offer part should be the service class of the offer (full flex + mid flex = mid flex)

Offers usually have a validity period, that is the period over which, from the fare conditions, the offer is likely to be proposed. It is not a guarantee that the offer remains available for that period

Offer resources and all related resources (TripOffers and all sub resources) should have a limited lifetime (recommended value 30 minutes) and be discarded when expired or at booking time.

### TripOffers

TripOffers are the resources grouping all the offers relating to one same trip. Indeed, in most cases the **Allocator** provider will propose several offers of different comfort and flexibility levels. In this resources, the trip resource representing the trip the offers are for and the passengers for the trip.

#### Offers with Partial Coverage

It is possible in OSDM to propose offers covering only a subset of the requested trip under specific conditions:

* the segments covered by a given offer are indicated through the coveredSegmentIndexes property
* all offers covering the same set of segments belong to the same offerCluster. As such, their offerCluster identifier must be filled-in and identical
* a segment can only be covered in one offerCluster within a tripOffer (no overlap)
* each segment of the trip must be covered by at least one offer in each TripOffer (no gap)

*Example with no overlap*

Clusters-no-overlap

Clusters-no-overlap

*Example with no gap*

Clusters-no-gap

Clusters-no-gap

### OfferParts

Although OfferParts are by themselves not a resources, they deserve a separate section in order to for the reader to clearly understand the data model design. The idea is that OfferParts represent an instantiation of a product that can be sold.

*Example*:

* A carrier proposes an “Early bird Holiday Fare” product for all its high-speed trains riding towards the seaside of the country, offered when sales date is at least 15 days prior to travel.
* Whenever an offer request is received and this fare can apply, an offer part is created and proposed, specifically to the date, origin and destination of that specific request, and referencing this product. As such it is therefore the offer part that for example will carry the actual price.

These offer parts can be of different type, depending on what they represent:

* **Admissions**
* **Reservations**
* **Integrated reservations**
* **Ancillaries**

However, all these different type share a significant amount of characteristics: they all apply to a defined set of passengers, have a price (calculated individually or collectively), and a few additional attributes. They also fill in the same fundamental role in the offer, which is why they are represented here as an abstract parent class.

#### Multiplicity

In OSDM, offer parts are not normalized, but will reflect the reality of the products generated. This means that one offer part will almost always equal one fulfillment in the resulting booking, should it be completely realized and confirmed.

So, two passengers travelling together happen to get exactly the same product (because their profile is identical in terms of age, reductions etc), will still get two individual offer parts (one per passenger) if the product covered has individual pricing and fulfillment, while they would be grouped in one offer part in case of collective pricing and fulfillment. (see examples at the end of the offer section)

### Offer Parts - Admissions

Admission offer parts represent a travel right, or the entitlement to travel onboard a train between the given origin and destination, following the given route, without a seat reservation. In most cases, these train products will not be train-bound either.

In some vehicles, seat reservations or an ancillary products (such as a WIFI connection or a meal onboard) can  be associated with the admission for one or more of the segments. A link will in this case point from the admission to the reservations or ancillaries, and the link will be qualified. Ancillaries can be either included or optional, while reservation can also be mandatory to travel. Finally there can be a cases where all reservations associated are optional while it is mandatory to pick at least one (it can be the case for night trains for example). In this case the reservations will all be qualified as optional, but the reservationRequired flag of the admission will be set to true.

### Offer Parts - Reservations

Reservation offer parts represent a seat (or other accommodation) reservation on the transportation. In contrast with admissions, a reservation is in essence bound to a specific train, although it normally does not include the entitlement to board the train. Travellers therefore typically need an a associated admission offer part or other entitlement (such as a pass)  in order to actually travel.

Note booking an offer will not book the reservations in the offer unless they have an included relationship with an admission of that offer. In order to add a non-included reservation to a booking, the reservation ids will have to be passed in.

Reservations have several additional attributes due to their specificities compared to admission products:

* Reservation Details provide additional information on the service Level and, once the offer will have been booked, the exact reserved places, with their properties and links to the concerned passengers
* Place selection Details: contains several elements related to the selection of places:
  + ReservationOptions show, at offer retrieval stag which options are available for this reservation.
  + SelectedOptions allows the API consumer to specify desired options.
  + SelectablePlaces and SelectedPlaces are only relevant to graphical selection of seats (seat map).

#### Modelling Lump Sum Reservations

For some trains, especially in Germany and Austria today, a specific form of reservation booking can be found where the price for adding an optional or mandatory (but not free) reservation remains the same regardless of the number of reservations actually booked. In order to represent this type of reservation with the current model, two approaches are proposed to implementers:

* Generate two distinct offers: one with all (available) reservations proposed as included, the lump sum for the reservations being integrated in the admission price. In this approach it is assumed that a passenger will always book all available reservation, since the price is the same anyway. This approach also allows to not propose a reservation if there is none available on one of the segment, while still offering the offer for the complete trip with reservations on all segments where it is available
* Propose all reservations as optional reservations with an identical unit price equals to 0 or to the reservation lump sum, associated with specific information in the product conditions or the offer messages. At booking time, a price update (increase or decrease) is then applied so that the lump sum is counted once and only once, associated with a booking message warning that the price update took place.

### Offer Parts - Ancillaries

Ancillaries are used to represent non-transport products associated with the transportation request submitted. It could be onboard services such as a WIFI connection or a meal, or services associated with one of the stops, or origin/destination, like a parking spot or lounge access.

This offer part is significantly simpler than those instantiating transport products, and only has one additional attribute, being the category of the ancillary.

### Products

Products are the products actually offered by the OSDM provider, either directly or either as distributor if the OSDM provider itself retrieves the products  (or constitutive fares) from another provider. Products resources contain all the conditions and attributes of the product, regardless of the actual sale case. Typically this matches commercial products having the same name and recognizable common sales & after sales characteristics.

Although no manipulation is performed on products, it is nevertheless proposed as a resource mainly to allow caching of the information. Indeed, since product information usually does not change too frequently (and usually at well defined dates), those resources can then be exposed with a significantly longer time-to-live and save significant bandwidth. It also allows for a “product catalog” functionality to be built by the API consumer, should he want to do that.

### Fares

Fares should be seen as the counter part of OfferParts in case of interactions between an allocator and a fare provider. The key difference here is that where offer parts are products defined by the provider and fulfilled by it as well, the fares do not constitute a distributable product. It is up to the allocator to build the distributable product (that he could then distribute as an offer part via an OSDM API), based on one single fare or by combining fares coming from different providers. In consequence, the fulfillment of the resulting product is the responsibility of the allocator as well.

For distribution systems also able and allowed to act as allocators, encapsulating both fares and offer product in offers allows to have a similar flow of interactions regardless of the type of provider.

### Passengers

As the name suggests, passenger resources represent the passengers for whom the offers are  proposed. All offers generated are always proposed for the complete set of passengers (no partial offers covering only a part of the passengers is generated). However, it is possible that because of age, reductions or other, some passengers are allowed to travel some segments without actually needing a travel right or reservation. It is for example usually the case for infants travelling on their parents lap.

While a lot of attributes can be set for passengers, only a few are required at this stage (and even later). The key elements at offer stage are already specified in the offer request. The link between the possibly anonymous passenger profiles (in most basic form:  a unique (in the booking) reference, age and reduction entitlement if any) created in the offer request and the passenger resources in the TripOffers can be made based on the passenger reference attribute.

The passenger resources created in the context of offers have their lifetime strictly limited to the lifetime of the offer resource they are part of. The resource and all local traces of it should be discarded once the offer has been booked or reached the end of the lifetime, in order to avoid any privacy concern.

### Passengers Representation

In the railway world, several elements are used to define a passenger profile (mostly in order to define the products it is entitled to):

* the passenger’s age
* the reduction cards the passenger owns
* whether the passenger is a reduced-mobility or otherwise disabled passenger
* other specific status entitling to specific fares (military, senator, journalist…)

While in some systems, all the notions above are mixed into one “passenger type” notion, this approach is much more difficult, and cumbersome, to apply when multiple providers are involved, which is highly likely with OSDM. Indeed, different systems often have different age limits for types, and different ways to represent the other elements. For this reason, in OSDM we decide to map the elements above to two kinds of attributes:

* Passenger birth date, modelled as-is in the API. Note it could be a dummy birth date. Each implementer is then free to map this value to the age-related passenger types he is using internally
* Some attributes related to passengers disabilities (for accommodation purposes mainly)
* All other notions are modelled as reductions. Again, each implementer can map internally this clearly defined notion to the internal representation.

The presentation hereunder provides some additional examples of high-level offer modellings for pure-OSDM offers.

## Booking

### Booking Data Model

Bookings Data Model

Bookings Data Model

### Main Resources

### Booking

The booking represents the local (to the OSDM provider) booking for the offers that have been selected. It contains a set of sub resources, most of which were encountered in the offer stage. but also adds a few specific attributes and information, the most important undoubtedly being the booking status (see for the state model below). The booking will indeed evolve over time based on API consumer actions, time elapsed or other business events.

The booking also contains additional attributes that are needed to manage and control the confirmation of the booking when it is in provisional state, such as the ticket time limit or the fulfillment options.The ticket time limit is the time during which the booking is guaranteed to remain available for confirmation for the price and possible reservations assigned at provisional booking time. Basically, it is the time given to the API consumer to perform all updates needed to confirm the booking, and trigger that confirmation.

FulfillmentOptions allows the API consumer to specify the format desired for the fulfillment. Only electronic fulfillment is considered in the MVP scope.

### BookedOffers

BookedOffers are actually the same resources as the offers except that they are now booked. Most of the resource remains unchanged, except for the sections on reservation details (either in reservation or integrated reservation Offerparts, or in fares), where but the sections related to the reserved places (in reservationDetails) will now be populated with the references to the space allocated by the provider system where the transport product is hosted.

### Fulfillments

Fulfillments could once have been called tickets. But the evolutions in the industry have led this to be a limitative naming, as various kinds of ticketless onboard controls are rapidly taking over and become the norm rather than the exception. Since in OSDM only the distribution part of the process is in scope, the details of how to produce or control fulfillment are not covered. From a distribution standpoint, the only needs are

The possibility to point at a fulfillment representing an offer part (= the id) for after sales operations. The capability to link this fulfillment to that associated offer part they relate to. A business identifier that can be used in associated processes. For railways, that would be the Ticket Control Number (TCN).

Links to the documents or other security features that can be used to represent and control fulfillment status. In most case it is a PDF document and/or a barcode. These are all provided in the fulfillment sub resource.

### RefundOffers

Refund offers represent a provisional refund request that is made on all or a subset of the fulfillments contained in a booking.

### Passengers

The passengers sub-resource in the booking is actually the same as the one in the tripOffers, but it is worth mentioning it separately here as

* being a sub-resource, it will have a different path
* as mentioned in the section about offers, the passengers in the TripOffers will disappear with the booking or the time-to-live expiry of the offers, and the passengers created in the booking will have a different id.

## State Models

### Booking State Model

Booking State Model

Booking State Model

### Fulfillment State Model

Fulfillment State Model

Fulfillment State Model

### Refund State Model

Refund State Data Model

Refund State Data Model

## Introduction

This page shows a representation of the data models underlying the API specifications. It is therefore not a strict representation of the resources themselves (those are self-represented in the OpenAPI specifications.)

The main purpose of this document is therefore to help a quicker understanding of the API and its underlying concepts. As such, some of the details of how the information is structured in the API are not represented or simplified in the data models.

## Overview of Services

|  |  |
| --- | --- |
| Resources | Description |
| /locations | Resources to search for locations |
| /trips | Resources to search for trips |
| /trip-offers-collection | Resources to get bookable offers |
| /trip-offers | *dito* |
| /offers | *dito* |
| /offers/{id}/admissions | Resources to manipulate parts of an offer consisting of, e.g., admissions, .. |
| /offers/{id}/reservations | .. reservations,.. |
| /offers/{id}/ancillaries | .. ancillaries,.. |
| /offers/{id}/fares | .. or if permitted also fares. |
| /bookings | Resources to manipulate bookings |
| /offers/{id}/passengers | Resources to manipulate a passenger’s information at every stage of the flow |
| /bookings/{id}/passengers | *dito* |
| /products | Resources to retrieve products information on one or more products |
| /bookings/{id}/fulfillments | Resources to retrieve fulfillments, e.g. tickets |
| /fulfillments | *dito* |
| /bookings/{Id}/refundOffers | Resources to get and accept a refund offer |
| /bookings/{Id}/refundOffers/{refundOfferId} | *dito* |
| /bookings/{Id}/exchangeOffers | Resources to get and accept a exchange offer |
| /bookings/{}/exchangeOffers/{exchangeOfferId} | *dito* |
| /coachLayouts | Resources to get layouts of coaches |

## Process Flow

Process Flow

Process Flow

The process flow starts with getting offers which can be chosen by the customer. Once selected they can be prebooked and after the payment process (which is outside of the scope of this document) they can be booked. The fulfillment of the booking can either be on paper or paperless.

If needed bookings can either be refunded or exchanged by providing the customer with a refund or exchange offer which can the be booked by the customer.

## Trips and Locations Processes

### Looking Up Locations

Looking Up Locations

Looking Up Locations

The \locations Lookup can be used by an API provider in order to search for locations. Two typical uses cases would be

* getting a set of locations (in full or as reference) from a substring of the name
* getting full details on a location based on on of its codes

Note that the functionality is not intended to trigger a “dump” of the complete locations list or to build a full “browsing” functionality, hence the lack of pagination features here.

Given the high stability of this information, locations are given a long time to live and get responses can be cached for a long period, so these operations should not be too costly in terms of calls or bandwidth.

#### Error handling

Error handling by the **Distributor** remains basic here as a handful of cases have to be handled:

* invalid characters in the search string
* no result found for the given criteria.
* The search did not return any result
* unknown error on server side

In all cases, the error handling starts and stops with the **Distributor** returning the appropriate JSON Problem element.

In case the error can apply to multiple fields, it is recommended to provide additional details such as the incriminated field in the detail property of the Problem element.

### Getting and Browsing Trips

Getting and Browsing Trips

Getting and Browsing Trips

If the API consumer only needs a schedule, and no bookable offer, it has the possibility to create a trips collection using POST /trip-collection. If the query is successful, the initial response to this will be a set of trips matching the provided search criteria.

Please refer to the Yaml specifications for the list of search criteria available. Depending on their respective journey planner capabilities, it could be that some criteria cannot be supported by one or the other OSDM:Distributor In this case it is up to the implementing party to clearly document those limitations together with the publishing of its OSDM endpoints. In all cases, at least origin, destination and travelDateTime must be supported

Based on an initially returned trips collection, it is then possible to retrieve earlier or later trips using GET the trip-collection by specifying the appropriate scrolling-tokens. As with all cases where nested resources can be returned, individually or in list, the embed feature allows specifying whether complete trips should be returned or only a title and a link. A GET verb without any scrolling-token will simply return the last set of trips return.

It is important that once a trip has been generated, its time to live has a sufficient duration to allow the possible subsequent uses:

* A trip or a trip id could be used in subsequent offer operations (see further)
* When scrolling back and forth over time, a same trip should maintain the same id, so the API consume can, if desired, expand the set of trips in its own context and have the guarantee that one same trip (in terms of content) will remain with the same id (in terms of resource id.

#### Error Handling

Since requesting trips still does not involve any transactional operation, the error handling is also limited to returning a JSON problem element. The following cases are to be considered:

* A search criteria value contains invalid value or invalid characters
* A search criteria lies outside accepted boundaries: it could be the date in the past, or too far in the future, or value outside bounds for the max number of changes
* The origin or destination is not known
* The search did not return any result
* Unknown error on server side

In case the error can apply to multiple fields, it is recommended to provide additional details such as the incriminated field in the detail property of the Problem element.

## Offers

### Getting and Browsing Offers

Getting and Browsing Offers

Getting and Browsing Offers

Requesting and browsing offers works a lot like the trips: the API consumer submits search criteria, and a collection of “trip offers” is returned. This collection can be browsed to earlier and later trips the same way as the trips collections.

The search criteria for offers extend the search criteria available for trips with additional criteria applicable to the fares and products that can be returned such as the fare flexibility, the service class or the currency the offers should be proposed in.

Although the trip-related search criteria are present and will likely be the easiest and most used option, there is an alternative way to search offers if a set of specific trips is already known: provide the complete trip structure for one or several trips. It is actually the only way to go for a request to a fare provider working according to nTM rules. In this case, the trips provided may be larger than the part for which fares are requested. For this reason, the requested section must then be provided so that the provider knows which portion to work on.

An offer request to an **Allocator** can lead to offers with multiple offerparts, potentially coming from different sub-providers (OSDM compliant or not). However, in preparing offers with multiple offer parts for the API consumer, the **Distributors** must follow the following rules :

* The POST /trip-offers-collection only generates complete offers covering the complete trip (or complete section) requested.
* While the combination logic is left to the **Distributor**, it is recommended to only build and retain offers that are *homogeneous* (as much as possible) in terms of flexibility and comfort.
* Each offer request should create a new offer context with a dedicated /trip-offers-collection resource and dedicated sub-resources, since it is possible, and may even be required to patch offers with data that is specific to the booking dialog at hand in order to perform the booking.

The resources used at offer steps offer various *levels of embedding* and multiple granularity for the retrieval of information, so each implementing party can fine-tune the queries in order to get all the information needed for the processing at hand, and only that information.

#### Offer Messages

During the offer construction, the **Allocator** can encounter events that, while not halting the process or constituting an error, may be relevant for handling of the response by the API consumer. These events can then be passed on using the offer Message element. The following events are identified and relevant to this section

* Overbooking
* Schedule correction applied

### Round trip handling

We define a round trip as a mirrored couple of trips *(A-B B-A)*, each made of one or more segments.

The construction of a round trip in OSDM is always a two-step process, where the outward offers are requested separately from the inward offers.

#### Receiving offers with return products and fares

In order to indicate to the provider that the intention is to build a return trip, the returnSearchParameters are used:

When requesting offers for the outward travel, the API consumer has to provide a return date. The response will contain a set of offers. Each of these offers will have a offerHash. Usage of it is described further below.

To get offer for the inward travel, the API consumer will have to provide

* The id of the outward tripCollectionID (allows knowing the context in which the outward offers are made)
* Depending on the targeted fare provider, the offerHash for the selected outward offer, or the set of potential offers (as the offerHash does not have to be unique. E.g. all offers for a given date might have the same offerHash if the constraint is only on date) can or must be provided. Whether the offerHash is mandatory in the inward offer request is indicated by the “mandatory flag” that is provided in the outward offer response next to each offerHash. If the offerHash is provided in the inward offer request, the provider should then only return offers that are compatible with the indicated (set of) outward offers.  
  Note that depending on whether the offerHash is mandatory or not and whether it is unique per outward offer, it may or may not be mandatory to select the outward offer before th inward offer request can be constructed.

#### Using combinationHashes

Besides the offerHash discussed above, some offers may have one or more combinationHashes as well. As the name suggests, these can be used in order to determine how to combine offers. For round trips constructions, we are specifically interested in combination types of the type “return”.

The idea is actually fairly simple: in case no filtering is applied on the inward offers using the offerHash filter mentioned above, the returned inward offers may not all be compatible with all outward offers. Identifying compatible pairs are simply identified by the fact that they have the same (set of) return combinationHash(es). Offers with no return combinationHashes have no constraints.

Hereunder an example illustrating this concept:

##### Outward Offers

* Offer1: -
* Offer2: #123
* Offer3: #234, #123
* Offer4: -

##### Inward Offers

* Offer5: -
* Offer6: #123
* Offer7: #234
* Offer8: #123, #234

##### Valid Combinations

* Offer1 + Offer5 (no constraint on hashes)
* Offer4 + Offer5
* Offer2 + Offer6
* Offer3 + Offer8

Offer7 cannot be combined with any offer on the outward set.

#### Products Covering Both Directions

While in most cases the two trips are materialized with distinct products/fares for the fare provider, there are fare providers still proposing unique products covering the outward as well as the return. In this case, the product element can be flagged as covering the mirrored segment as well. As for the offer construction process, the provider will simulate the two steps approach by using one of the following approach:

* The same product covering both outward and return is proposed in the offers for the two directions
* For one of the two directions, a dummy product is returned.

Regarding the price, it can either be placed in full on the offers in the two directions (but then the total price will be incorrect when looking at the complete return travel), or split in any way desired between the outward and the return.

Products Covering Both Directions

Products Covering Both Directions

#### Error Handling

* the referenced trip cannot be found as it might have expired
* A search criteria value contains invalid value or invalid characters
* A search criteria lies outside accepted boundaries: it could be the date in the past, or too far in the future, or value outside bounds for the max number of changes
* The origin or destination is not known
* The trip search did not return any result
* No offer could be built for any of the discovered trips
* Schedule mismatch between systems
* Unknown error on server side

#### Getting Coach Layouts

Graphical seat reservation allows a customer to conveniently choose its preferred place. Therefore two resources are added: First, GET /coachLayouts to import all coach-layouts of an allocator. This service can be used periodically as master data service . Second, GET /coachLayouts/{layoutId} returns the information for a given layoutId and can be used during the on-line offering and booking process.

## A Complex Example Mixing Offers and Fares

### Request From Front-end

I want to go from Rotterdam to Wien Stephansplatz via Antwerp.

### Request Submitted to SNCB

Proposed trip by timetable system:

|  |  |
| --- | --- |
| Origin - Destination | Train Number |
| Rotterdam → Antwerp | Thalys 9324 (integrated reservation) |
| Antwerp → Liège | IC 2345 + IR 5567 |
| Liège → Frankfurt | ICE 122 (mandatory reservation) |
| Frankfurt → Wien Hbf | RailJet RJ 23 (optional reservation) |
| Wien Hbf → Wien Stephansplatz | Metro |

### Fare Provider Resolution returns

|  |  |  |  |
| --- | --- | --- | --- |
| Origin - Destination | Train Number | Fare Provider | Consolidated |
| Rotterdam → Antwerp | Thalys 9324 (integrated reservation) | PAO | PAO |
| Antwerp → Liège | IC 2345 + IR 5567 | Fare SNCB | Fare SNCB |
| Liège → Frankfurt | ICE 122 (mandatory reservation) | GUS | GUS |
| Frankfurt → Wien Hbf | RailJet RJ 23 (optional reservation) | Frankfurt → Salzburg (Border) | Fare DB |
|  |  | Salzburg (Border) → WienHbf | Fare ÖBB |
|  |  | Frankfurt → Wien Hbf (reservation) | Fare ÖBB |
| Wien Hbf → Wien Stephansplatz | Metro | Fare ÖBB |  |

### Completing Offers for Provisional Booking

Completing Offers for Provisional Booking

Completing Offers for Provisional Booking

Once an offer has been selected some additional steps can be taken to complete the information:

* Adding passenger information
* Adding accommodation preferences regarding the accommodation (most of the times a seat, but could be sleeping accommodation for night trains), either for the type of accommodation space, or its exact location.

While selecting accommodation preferences will be optionally most of the time, some information (usually on passengers) may be mandatory in order to proceed with the booking. The RequestedInformation property will provide the details of what needs to be specified in order to book a given offer. These details are provided under the form of a boolean expression, referring to the passenger model elements using dot notation (with the TripOffer as the root). For example, if it is required that name and first name are set to proceed RequestedInformation would be :

passengerDetails.firstName AND passengerDetails.firstName

Another example, if on top of first and last names, at least one email or one phone number is needed:

(passenger[0], "passengerDetails.firstName AND passengerDetails.firstName" AND (passengerDetails.eMails[0] OR passengerDetails.phones[0])

By parsing this structure, the API consumer is able to identify the elements that need to be filled-in to proceed

The two types of updates (accommodation preferences and passenger data updates) are applied using a PATCH verb on the related resources. While the resource in its whole is presented, only property explicitly listed as updateable can be provided with a value. Attempting to modify another property must result in an error.

For accommodation preferences, the following properties can be updated:

* Reservation:
  + reservation.placeSelection.selectedOptions
  + reservation.placeSelection.selectedPlaces
* Fare
  + fare.placeSelection.selectedOptions
  + fare.placeSelection.selectedPlaces

For passengers, all properties are updateable except

* id
* reference
* type

Note however that updating a property that could have influenced the product in the offer (such as date of birth or reduction cards) will not influence the offer content anymore: the booked offer will be the offer initially generated, even though the products booked may be inconsistent with the new values of the passenger properties.

Reminder: the accommodation preferences can be found in the reservationOptions elements (offer.fare|integratedReservation|reservation.placeSelection.reservationOptions)

#### Error handling

* The requested reservation Option is not available on this transport
* An invalid value is provided for a passenger property
* Attempted to modify a read-only property

## Booking Processes

### Creating a Booking Based on Offers

Creating a Booking Based on Offers

Creating a Booking Based on Offers

Once the offer selected has been completed and all requested information provided, the API consumer can continue to the booking of that offer. Along with the offer, optional or mandatory reservations or ancillaries could be booked as well. those optional offer parts can be identified easily in the offers as they will always be linked with an admission product (in admission.reservations or admission.ancillaries). The link contains the relationType property, which indicates whether the pointed reservation is included (in which case it is not needed to explicitly add it in the booking request) , mandatory (the reservation must be added in the booking request) or optional (the reservation can be added in the booking request. Ancillaries are never mandatory (only included or optional). Adding optional or mandatory elements is simply done by adding the respective offer part in the booking request (cf YAML specifications) POST /bookings.

It is also possible to book several offers in one operation to the same booking. This is especially relevant to support return trips, where in most times it will be mandatory. If this is the case, a collection of offer ids (and associated reservations and ancillaries) is given instead of just one. However, note that in this case the passengers party for all booked offers needs to be the same. To ensure this, the passenger reference of each member of the passenger party must remain the same from one offer to the other.

If the booking succeeds, a new booking resource is created. In this booking, the booked offers can be found and should look a lot like the offers as they were in the offer responses, with the exception that for (integrated) reservations and fares, the reservedPlaces element will now be populated with the places that have actually be assigned to the passengers for this offer part.

The passengers in the booking resources are also the same type of resources as the ones manipulated in offers. However, they have to be different resources with different ids (the passengers references do remain unchanged).

Initially, a booking will have the status PREBOOKED (see also the booking status model).

At the root of the booking structure, Two balance elements are provided, in order to clarify the state of the financial exchange between an API consumer or booker and the OSDM:Distributor:

* conditional balance is the balance of the booking that is not confirmed. It is the amount that will be due to the provider if the booking is further confirmed.
* confirmed balance: is the balance of the booking that is confirmed. Unless after sales takes place on one or more fulfillments in the booking, this amount now must be paid to the provider.

At the root of the booking structure is also located the ticket time limit. This is the time for which the provider will hold a booking in pre-booked state, waiting for the confirmation while guaranteeing the booking for the given products, spaces at the announced price. Obviously, this value only has a meaning for a booking in pre-booked state. A commonly accepted value would be around 30 minutes, which is normally sufficient to allow finalizing the booking,while not monopolizing resources too long in case the booking is abandoned without properly cancelling it. However, some systems may decide a longer time. Obviously, the value for the booking ticket-time limit can never exceed the earliest ticket time limit of any of its offer parts.

#### Provisionally Booking a Return Trip

While this may not be true for all providers, most of them require that the outward and the return parts of a return trips are booked together in order to actually book a return-specific product. Therefore, when building a return travel, the API consumer should always specify the outward offer(s) and return offer(s) in the same POST /bookings operation.

#### Provisionally booking a trip with offers clusters

When booking for a trip for which several offer clusters were provided ([see offer clusters](../models#offers-with-partial-coverage))), the API consumer must be careful to always select one and only one offer from each offer cluster in the tripOffer. This ensures that even though the selection is done per offer cluster, the complete trip is covered exactly without any gap nor overlap. However, the provider implementers must verify and validate the set of offers selected is valid. if the trip being booked is also a return trip, then the rule applies for each direction.

### Handling Partial Success of Pre-Booking

Handling Partial Success of Pre-Booking

Handling Partial Success of Pre-Booking

As a **Distributor**, partial pre-booking is not expected. As a consequence all pre-booking operations are either fully successful or not executed at all. However, as an allocator, An **Distributor** system may be configured in such ways that it is able to combine offers from different sub-providers (via an OSDM-compliant API or not) and propose them in turn as one offer to its API consumers, as one undividable product or as a bundled pack.

Unfortunately, when the booking is attempted, the process may encounter errors leading to the booking failing with some of the sub-providers, while it will have succeeded for other parts of the offer, directed to other sub-providers. The result is a partially pre-booked booking. Since this situation is not compliant with the OSDM specifications, this situation needs to be rolled back. This can be done by cancelling the pre-bookings that were successful (on an OSDM sub-provider, it would be performed using the DELETE /booking/id verb). An appropriate error message is then returned in the booking response, under the form of a JSON problem element.

As it was the case with offers, during the booking process, some events may occur that are worth communicating to the API consumer, while they do not really constitute an error nor should interrupt the booking process. These events and situations can be communicated through the Warning messages:

* Price change: the booking succeeded, but the price of the offer has been modified between the offer generation and its actual booking
* Overbooking

#### Error Handling

* Referenced Offer or offer part not found (offer expired ?)
* No rights to access referenced offer
* Incompatible offer part with the offer
* Missing information
* Reservation to sub-system failed for one or more offer parts
* Insufficient availability for one of the requested products
* Requested place not available

#### Notes

* Booking an offer will not book the reservations in the offer unless they have an “included” relationship with an admission of that offer. In order to add a non-included reservation to a booking, the reservation ids will have to be passed additionally or it will not be booked.
* It is up to the OSDM API implementing party to decide whether booked offers can have the same resource ids as the offers in the shopping stage. However, it is assumed in the specifications that this is not the case, and the API Consumer should not rely on this possibility.
* In case the passengers details are different in the different offers added together in a booking, the passenger information of the first offer will be copied in the booking, and those of the following offers will be ignored.
* When the booking ???

### Completing Booking for Confirmation and Fulfillment

Completing Booking for Confirmation and Fulfillment

Completing Booking for Confirmation and Fulfillment

When the booking has been successfully created, some additional changes may be desired or even required before the booking can be confirmed.

* As with offers, some passenger information may be required. If this is the case, the mechanism used is exactly the same as for offers: the requestedInformation property at booking level will indicate which information is needed to confirm using boolean expressions and dot notation. Updating the values is done via a PATCH on passenger sub-resources of the booking (as for the offer). Even if all the required data is already present, it could still be relevant to update these values. For example a dummy date of birth might, due to the selected fulfillment type now be requested to be the exact date and require an update, even though the property is already filled-in.
* It may be needed or desired to change or set fulfillment type and options. It is however recommended to the **Distributor** implementers to set a default value for these properties (especially if only one value is possible). Note that the choice of the fulfillment type & options may impact the requestedInformation. This property should therefore be re-evaluated whenever the fulfillment type is modified (both on the provider and on the consumer side).

#### Notes

As in the offer, the modifications on the passenger’s properties will never impact the products in the offer (thus also not the price), even if this leads to an inconsistency between the offered product and the updated passenger property.

#### Error handling

* An invalid value is provided for a passenger property
* Attempted to modify a read-only property
* The booking is confirmed/refunded/cancelled and does not allow modifications

### Cancel a Not Confirmed Booking

Cancel a Not Confirmed Booking

Cancel a Not Confirmed Booking

In case a pre-booked booking is abandoned by its user, and this event is captured, it is recommended for the API consumer to properly cancel the booking on the **Distributor** side. In case this is not done the booking will be cancelled when the ticket time limit is reached, but in the meantime all related resources (seats etc) will remain unavailable for other requests. Upon receiving a DELETE /bookings for a given booking, the **Distributor** should obviously do its own cleaning as well, and if needed pass on the cancel to its sub-providers.

In case of a partial success for booking, the DELETE /bookings can also be used to clean-up the bookings on sub-providers where the pre-booking succeeded and who support the OSDM protocol.

Regardless of whether the cancel occurred through an explicit DELETE /bookings or expiry of the ticket-time-limit, the booking state will then change to CANCELLED for a short “grace” period, before being completely cleaned-up (offer parts are well cleaned-up immediately). This grace period aims at ensuring that any ongoing operation with the booking is given sufficient time to get an explicit info on the cancelled status of the booking. The choice of the duration of that grace period is left to the implementor.

#### Error handling

* the booking is already confirmed
* the booking is already cancelled
* unknown error on the server side

## Confirmation and Fulfillment Processes

### Fulfillment Process

Fulfillment Process

Fulfillment Process

The fulfillment is the final step of the booking. In most cases, the booking will be confirmed and fulfilled in one step from the API consumer standpoint:

* fulfillments elements are created with the appropriate status (see below)
* the provisional balance becomes confirmed
* the status of the booking changes to FULFILLED (for most systems) or CONFIRMED (see below)
* if relevant the documents elements in the fulfillment resources are created and linked

However, in case the **Distributor** acts as a distributor for products or fares actually hosted in sub-provider systems (OSDM compliant or not), a lot more takes place behind the scene. Indeed, the **Distributor** will have to

* confirm or fulfill the bookings towards all the sub-providers
* retrieve the fulfillment details to populate its own booking responses (and databases, most likely)
* build the fulfillments elements
* update relevant booking properties as described above.

#### Error handling

In the confirmation and fulfillment process, the following issues can arise:

* Unknown error on provider side
* Missing information in the booking
* No fulfillment type selected
* Booking already confirmed/fulfilled/cancelled

#### The Special Case of Partial Success

If a booking is composed of multiple offer parts, some of them potentially coming from sub-providers, it could be that at confirmation (or fulfillment) time, the operation only succeeds for some of the bookings. Unfortunately, a clean roll back to the previous state is not possible here for the succeeded confirmation. The middle **Distributor** (combining offers of its sub-providers on request of its API consumer) has several options to handle the situation:

In all cases, the middle **Distributor** obviously has the option of proactively retrying to confirm on OSDM sub-providers where the confirmation failed. But this may keep on failing beyond a reasonable waiting time for the API consumer. A different strategy then needs to be applied:

The first possibility is to completely clean up the booking by:

* cancelling unconfirmed content
* refunding confirmed content (with overrule if needed)
* returning an error message to the API consumer

In this case, the specific error handling remains concealed for the API consumer, who only will be informed of the final result, being the the booking has failed and been completely cancelled.

The second option is to expose the situation to the API Consumer and let it decide of the course to be taken. In this case, the resulting partial booking is returned to the API consumer with an error state

The choice of the strategy to follow here is left to the implementers. However, the implementer who would choose to expose the situation and let the API consumer handle it, also needs to implement the logic described hereunder. This may be slightly more complex than proactively cleaning up the booking in its entirety.

If this strategy is chosen, the partial booking will then be returned with the following specific characteristics:

* the returned booking has an ERROR status
* fulfillment is available/fulfilled only for some of the offerparts
* the confirmed balance amount only totals offer parts where the confirmation actually succeeded, while the provisional balance amounts to the total of the offer parts where the error occurred (or where the confirmation was never attempted because the error came too soon)

The following options are then available to the API Consumer:

* Explicitly request a retry on the confirmation, by re-triggering a POST or PATCH / Fulfillment. The **Distributor** will then re-attempt to confirm the not-yet confirmed content in the booking, while leaving the confirmed unchanged.
* Either directly, or after a few attempts on re-confirming, the booking needs to be cleaned-up so it can have a consistent status again (meaning the totality of the content is confirmed). To do so:
  + The API consumer must start by cancelling the non-confirmed content. He can do so by sending a PATCH on the booking where the cleanupPartialBooking property set on TRUE. This will result in
    - the cancellation of all non confirmed content,
    - adaptation of the balance values (provisional balance = 0, confirm balance = sum of confirmed products)
    - a reset of the booking status to FULFILLED (or CONFIRMED, depending on the confirmed content fulfillment status)
  + If deemed relevant, the API consumer can even completely remove the booking by refunding the confirmed part, if needed using an overrule code.

### Confirm booking without fulfillment

For some providers or products, the booking confirmation and the fulfillment step are distinct steps, while for others/most, booking confirmation and fulfillment are performed together. For products where this is the case, the fulfillment item generated by the POST fulfillment will show several differences from those where the product is confirmed and fulfilled in one step:

* The most obvious difference is the status, that is set to CONFIRMED instead of FULFILLED
* No document nor fulfillment item will be provided
* The fulfillment may not have a controlNumber.

In terms of process, creating this fulfillment at this stage allows an uniform confirmation process (the totality of the booking is confirmed in one step) for bookings that would mix the two kinds of fulfillment processes. The fulfillments can later get PATCHed in order to trigger the actual fulfillment.

When a confirmation request is received by the **Distributor**, it should first ensure that the operation is indeed supported for all offer parts in the booking (whether the **Distributor** is hosting those or they are coming from sub-providers). Indeed, OSDM (in MVP phase at least) will not support partial confirmation or partial fulfillment.

If this check is successful, then the execution of the confirm can start:

* All offer parts will be confirmed (locally or via requests to sub-providers), in parallel or sequentially
* The ticket-time-limit is invalidated (set to 0)
* The state of the booking is set to CONFIRMED
* The provisional balance is set to 0
* The confirmed balance is set to the total amount of the booking
* Response is sent to the API consumer

As of that point, cancelling the order becomes impossible (except for cleaning up cases, cf below) and any subsequent change should be handled as an after sales operation. Once the booking is confirmed, it becomes also impossible to modify any element in the booking (such as fulfillment type or passenger information)

### Interlude: Requested Information per Process Step

These are the required information needed per process step for major parties

|  |  |  |  |
| --- | --- | --- | --- |
| Distributor | Pre-booking Step | Booking Step | Fulfillment Step |
| Bene |  | First name and name |  |
| DB | In general one first name and name, regardless of the number of travelers. In case of regional trains, however, all names and surnames are needed, unless printed on security paper. |  |  |
| öBB | Passenger names are needed (first name and surname) Birth date may be needed. Some reduction cards require the number to be provided at pre-booking time, in order to be pre-checked. In other cases, the cards are simply checked on-board phone number or email (once per order - as contact information) | phone number or email (once per order - as contact information) |  |
| RENFE | Per passenger: Name, first name, surname Document type and identity document (DNI, NIE or passport). A phone number or email. | Per passenger: Name, first name, surname Document type and Identity document. (DNI, NIE or passport) A phone number or email. |  |
| SBB | Per passenger: name and first name date of birth. Additional sales parameters for some, none MVP products |  | e-mail |
| SNCF | Birth date is mandatory, a fake date can be used at offer time, but the real one must be provided at pre-booking time |  |  |
| Eurostar/Thalys | first name and name | Thalys loyalty card number |  |

## After Sales Processes

### Request a Refund Offer

Request a Refund Offer

Request a Refund Offer

On a confirmed booking, and if it is allowed, after sales operations are also possible via the OSDM API. In OSDM, the refunds are taking place based on fulfillment resources. There is no partial refund of one fulfillment possible. This also means that in case of collective ticketing, all passengers will be refunded in one go.

In order to perform a refund, the API consumer first has to create a refundOffer in the booking where the fulfillments to refund are located with a POST refundOffer. If the set of fulfillments provided is a valid set for refund, the operation creates a refundOffer that contains the information that is relevant to the refund operation at the moment the refund offer was created. This includes information such as the amount that will be refunded, any potential refund fee, etc (see the model for more details).

### Cancel a Refund Offer

Cancel a Refund Offer

Cancel a Refund Offer

### Confirm a Refund Offer

Confirm a Refund Offer

Confirm a Refund Offer

## Example End-to-end Interaction

Example End to End Interaction

Example End to End Interaction

This code lists are provided for convenience only. The mandatory code lists are provided within the schema or within the IRS90918-10 specification.

## Accommodation Type

|  |  |
| --- | --- |
| Code | Description |
| SEAT |  |
| COUCHETTE | Night trains only |
| BERTH | Night trains only |
| VEHICLE |  |

## Accommodation Sub Type

RS 90918-1 (Service Level).

(to be updated if/when further changes are applied to

|  |  |
| --- | --- |
| Code | Description |
| AC | Seat near children’s play area |
| AH | Seat in historic coach |
| AM | Seat in separate Compartment |
| AR | Wheel chair seat |
| AS | Quiet Compartment (Seat) |
| AV | Seat with front-view |
| BE | Restaurant (places in a dining car) |
| BP | Private compartment seats |
| D2 | Cabin-couchette coach |
| D4 | Couchette Four-berth |
| D5 | Couchette Five-berth |
| D6 | Couchette Six-berth |
| DP | Private compartment couchettes |
| DR | Wheelchair in sleeping car |
| E | Sleeperette |
| F4 | Ladies compartment, 4-couchettes |
| F6 | Ladies compartment, 6-couchettes |
| L | Single |
| M | Special |
| N | Double |
| O3 | Vehicle parking place category 1 – 3 |
| O4 | Motorcycle |
| O5 | Motorcycle with sidecar |
| O8 | Vehicle parking place category 6 – 8 |
| OB | Bicycle |
| OT | Tandem Bicycle |
| P | T2 |
| PD | Private compartment berth deluxe |
| PS | Private compartment berth |
| Q | T3 |
| R | T4 |
| S | Single with shower & WC |
| SL | Single with shower & WC & double bed |
| T | Double with shower & WC |
| TL | Double with shower & WC & double bed |
| U | T3 with shower & WC |
| W | Double with shower |

## Carrier

Carrier codes used in OSDM are based on the [UIC RICS Company Code](https://uic.org/support-activities/it/rics?recherche=RICS%20code) standard.

## Currency

Currency codes used in OSDM are based on the [ISO 4217](https://en.wikipedia.org/wiki/ISO_4217) standard.

## Fare BerthType

Legacy reservation code defined in UIC 90918-1.

## Fare CoachType

Legacy reservation code defined in UIC 90918-1.

## Fare CompartementTypeCode

Legacy reservation code defined in UIC 90918-1.

## Language

Language codes used in OSDM are based on the [ISO 639-1](https://en.wikipedia.org/wiki/ISO_639) standard.

## Nationality

Language codes used in OSDM are based on the [ISO 639-1](https://en.wikipedia.org/wiki/ISO_639) standard.

## OverRule Code

|  |  |
| --- | --- |
| Code | Description |
| SALES\_STAFF\_ERROR |  |
| PAYMENT\_ERROR |  |
| STRIKE |  |

Refund because of breakage of a vehicle is handled in a separate process.

## Passport

|  |  |
| --- | --- |
| Code | Description |
| ID\_CARD |  |
| PASSPORT |  |
| DIPLOMATIC\_PASSPORT |  |
| REFUGEE\_TRAVEL\_DOCUMENT |  |
| DRIVING\_LICENCE |  |

## Place Property

|  |  |
| --- | --- |
| Code | Description |
| ACC\_BICYCLE | Place with bicycle |
| ACC\_PRAM | Place with space for a pram |
| ACC\_TANDEM | Place with tandem bicycle |
| AISLE |  |
| AIR-CONDITIONED |  |
| BICYCLE | Bicycle |
| BICYCLE\_INCL\_SEAT | Bicycle including seat |
| BISTRO | Places in a coach with self-service bistro |
| BUSINESS | Manager compartment/business |
| CABIN8 |  |
| CARRE | Carré (4 seats facing normally 2nd Class) |
| CLASSIC | Classic coach |
| CLUB | Club (RENFE) |
| CLUB\_2 | Club Duo (2 seats facing in a separate compartment) |
| CLUB\_4 | Club 4 (4 seats facing) |
| COMPARTMENT |  |
| COMPARTMENT\_SHOWER\_WC | Compartment with shower and WC |
| COMPARTMENT\_WC | Compartment with shower |
| CONFERENCE | Conference compartment |
| CONFERENCE\_ROOM | Conference room |
| CONNECTING\_DOOR | Compartments with connecting Door (in Sleepers) |
| DOUBLE\_BED | Sleeper with double bed |
| DUO\_F2F | Duo face to face (2 seats facing) |
| DUO\_SBS | Duo side by side (2 seats side by side) |
| EASY\_ACCESS | Place with easy access for PRMs |
| FAMILY | Places in family area |
| FEMALE | Female compartment |
| FRONT\_VIEW | Places with view to the front |
| KIOSQUE | Kiosque (special seats in edge area) |
| LOWER\_BED |  |
| LOWER\_COUCHETTE |  |
| LOWER\_DECK |  |
| MANAGER | Manager compartment / business |
| MIDDLE\_BED |  |
| MIDDLE\_COUCHETTE |  |
| NEAR\_ANIMALS | Places close to place with animals |
| NEAR\_DINING | Places near the dining car |
| NEAR\_PLAY\_AREA | Places near a child play area |
| OFFICE |  |
| OPEN\_SPACE |  |
| PANORAMA | Panorama coach |
| PHONE | Places in an area with mobile phone amplifier |
| POWER | Place with power socket |
| PRAM | Space for a pram |
| PRM | Places for passenger needing assistance / disabled |
| RESTAURANT | Places in the restaurant coach |
| SALON | Salon (6 seats facing in separate compartment) |
| SILENCE | Places in silence area |
| SLEEPERETTE | Sleeperette (reclining seat) |
| SOLO | Separate place |
| TABLE | Places at a table |
| TANDEM | Tandem bicycle |
| UPPER\_BED |  |
| UPPER\_COUCHETTE |  |
| UPPER\_DECK |  |
| VIDEO | Place with video entertainment |
| WHEELCHAIR | Wheelchair place with additional seat |
| WHEELCHAIR\_WS | Wheelchair place without additional seat |
| WIFI | Places with WiFi access point |
| WINDOW |  |
| WITH\_ANIMALS | Place with animals (animals allowed) |
| WITH\_SMALL\_CHILDREN | Places for passengers with small children |
| WITHOUT\_ANIMALS | Place in an area where animals are not allowed |

## Point of Interest (POI)

POICodeList: By default the code list is set to UIC.

Additional code lists can be defined by implementers. The code list name should then be prefixed by “X\_<3 letters code for the provider>”. Example: “X\_PAO\_POIS”

POICode: Values are depending on code list and set is too large to be reproduced

## Reduction Cards

UIC code list:

|  |  |  |  |
| --- | --- | --- | --- |
| Card-Id | Issuer | Description | Type |
| UIC\_EURAIL | Eurail | Eurail Pass | PASS |
| UIC\_INTERRAIL | Eurail | Interrail Pass | PASS |
| UIC\_FIP\_LEASURE\_RED | FIP | FIP reduction (50%) | REDUCTION\_CARD |
| UIC\_FIP\_DUTY | FIP | FIP duty | PASS |
| UIC\_FIP\_LEASURE\_FREE | FIP | FIP free personal use | PASS |
| UIC\_RAILPLUS | \* | A pure rail plus card | REDUCTION\_CARD |
| UIC\_RIT\_1 | \* | RIT reduction for RIT 1 members | REDUCTION\_CARD |
| UIC\_RIT\_2 | \* | RIT reduction for RIT 2 members | REDUCTION\_CARD |
| UIC\_RIT\_3 | \* | RIT reduction for RIT 3 members | REDUCTION\_CARD |

Other parties are allowed to accept additional reduction cards.

## Refund Overrule Codes

|  |  |
| --- | --- |
| Code | Description |
| STRIKE |  |
| SALES\_STAFF\_ERROR | Error made by sales staff |
| PAYMENT-FAILURE | Cancellation made by the allocator due to a failed payment |

## Reservation Service Code

Legacy reservation code defined in UIC 90918-1.

## Reservation Preference

|  |  |  |
| --- | --- | --- |
| Code | Group | Description |
| AISLE | PLACE\_LOCATION |  |
| WINDOW | PLACE\_LOCATION |  |
| UPPER\_BED | BED\_LOCATION |  |
| LOWER\_BED | BED\_LOCATION |  |
| MIDDLE\_BED | BED\_LOCATION |  |
| UPPER\_COUCHETTE | BED\_LOCATION |  |
| MIDDLE\_COUCHETTE | BED\_LOCATION |  |
| LOWER\_COUCHETTE | BED\_LOCATION |  |
| UPPER\_DECK | LEVEL |  |
| LOWER\_DECK | LEVEL |  |
| COMPARTMENT | PLACE\_GROUPING |  |
| OPEN\_SPACE | PLACE\_GROUPING |  |
| TABLE |  | Places at a table |
| BICYCLE | VEHICLE | Bicycle |
| TANDEM | VEHICLE | Tandem bicycle |
| PRAM | VEHICLE | Space for a pram |
| AIR-CONDITIONED |  |  |
| PANORAMA |  | Panorama coach |
| MANAGER |  | Manager compartment / business |
| VIDEO |  | Place with video entertainment |
| CABIN8 | PLACE\_GROUPING |  |
| DUO\_F2F | PLACE\_GROUPING | Duo face to face (2 seats facing) |
| DUO\_SBS | PLACE\_GROUPING | Duo side by side (2 seats side by side) |
| CLUB\_2 | PLACE\_GROUPING | Club Duo (2 seats facing in a separate compartment) |
| CLUB\_4 | PLACE\_GROUPING | Club 4 (4 seats facing) |
| CARRE | PLACE\_GROUPING | Carré (4 seats facing normally 2nd Class) |
| SALON | PLACE\_GROUPING | Salon (6 seats facing in separate compartment) |
| KIOSQUE | PLACE\_GROUPING | Kiosque (special seats in edge area) |
| SOLO | PLACE\_GROUPING | Separate place |

## Reservation Preference Group

|  |  |
| --- | --- |
| Code | Description |
| ACC\_VEHICLE | Different types of accompanying vehicles |
| BERTH\_LOCATION | Location of a berth or couchette |
| FAMILY | Different types of places for families |
| LEVEL | Upper or lower deck |
| PLACE\_GROUPING | Compartment / Open Space |
| PLACE\_LOCATION | Place location (Aisle, Window) |
| USAGE | Different usage types for different coach areas |
| VEHICLE | Different types of vehicles |

## Service Brands

https://uic.org/spip.php?action=telecharger&arg=3007

|  |  |  |  |
| --- | --- | --- | --- |
| Service Brand Code | Abbreviation | Reservation Ticket Text | MERITS Description |
| 33 |  | Ferry | Ship |
| 37 |  | normal train | Train |
| 46 | TAJ | TAJ | Day car train |
| 47 | TAC | TAC | Car sleeper train, motor rail (CST) |
| 48 | SAE | SAE | Unaccompanied car service, motor rail |
| 49 | EIC | ExpressIC | Fast and Comfortable Interregional trains |
| 50 | EC | EuroCity | EuroCity |
| 51 | ICE | ICE | ICE |
| 52 | AVE | AVE | AVE |
| 53 | EIL | EUROSTAR | Eurostar |
| 54 |  |  | Talgo |
| 55 | OTU | Oresundstog | Oresundstog |
| 56 | TGV | TGV | TGV Bruxelles – Lille / Province |
| 58 | TRN | Intercités | Intercités |
| 59 | AE | ALLEGRO | Allegro |
| 60 | ECB | EuroCityBrenner | EuroCityBrenner |
| 62 |  |  | Suburban service |
| 63 | IC | Intercity | Intercity |
| 64 |  |  | Hotel Train |
| 65 |  | Ferry | hydrofoil |
| 66 | IC | Intercity | Inter City Lyn |
| 67 |  |  | TRN |
| 68 |  |  | International |
| 69 |  |  | Express |
| 70 | EN | EuroNight | Euro Night |
| 71 | HST | High-speed train | High-speed train |
| 72 | TRN | TRAIN | Train SNCF |
| 73 | TGV | TGV | TGV Sud-Est |
| 74 | TGV | TGV | TGV Atlantique |
| 75 | TGV | TGV | TGV Nord |
| 76 | TGV | TGV | TGV Lyria |
| 77 | TGV | TGV | TGV Duplex |
| 79 | TGV | TGV | TGV Est |
| 80 | TGV | TGV | TGV Interconnexion |
| 82 |  | THALYS | Thalys |
| 83 |  | Ferry | Hovercraft |
| 84 | RE | regional train | Regional |
| 85 | GPE | Gotthard Panorama Express | Gotthard Panorama Express |
| 87 |  | PENDOLINO | Pendolino |
| 88 |  |  | Suburban |
| 89 | ALV | Alvia | Alvia |
| 90 | AVN | Avant | Avant |
| 91 | TER | TRAIN | Regional TER |
| 92 | REG | Regiontog | Regiontog |
| 93 | FB | FRECCIABIANCA | FRECCIABIANCA |
| 94 | SC | SuperCity | Supercity |
| 95 | CNL | City Night Line (D) | DB Nachtzug |
| 96 | INI | InterCityNotte Italia | InterCityNotte |
| 97 | GB | ATOC MEMBER OPERATED SERVICE | ATOC MEMBER OPERATED SERVICE |
| 98 | ESI | ES Italia | Eurostar Italia |
| 99 |  |  | Funicular |
| 100 |  |  | Airport train |
| 101 |  |  | Night train |
| 102 |  |  | Touristic train |
| 107 |  |  | Historical train, steam engine train |
| 108 | IRE | IRE | Interregio-Express |
| 109 | RB | RB | Regionalbahn |
| 110 | RE | RE | Regional-Express |
| 111 | RT | RT | RegioTram |
| 112 |  |  | Shinkansen |
| 113 | THT | TrainHotel Talgo | Train hotel talgo |
| 114 | EUR | Euromed | Euromed |
| 115 | ALR | Alaris | Alaris |
| 116 | ALT | Altaria | Altaria |
| 117 | ARC | Arco | Arco |
| 119 |  |  | S-Bahn |
| 121 |  | Night Train | Night Train |
| 122 | IR | Interregional | Interregional |
| 123 | IRN | Interregional Night Train | Interregional Night Train |
| 124 | NLT | TOLSTOI | Tolstoi |
| 126 |  |  | ARZ |
| 128 | AVE | RENFE SNCF EN COOPERATION | Renfe SNCF |
| 129 | TGV | TGV INOUI | Renfe SNCF |
| 130 | BUS | IC Bus | Bus |
| 131 | BUS | IC Bus International | Bus |
| 153 |  | special train | Sonderzug |
| 154 |  |  | InterCityRapid |
| 155 |  |  | InterPici |
| 157 |  |  | Fast train |
| 158 |  |  | Euregio |
| 159 |  | Bus | IC Ersatzbus |
| 160 |  | Bus | IP Ersatzbus |
| 162 |  | Bus | Replacement Bus |
| 163 | TGV | TGV | TGV Duplex Lyria |
| 166 | TGV | TGV INOUI | TGV Duplex France Allemagne |
| 170 | YHT | YHT | High speed train in Turkey |
| 171 | FA | FRECCIARGENTO | FRECCIARGENTO |
| 172 | FR | FRECCIAROSSA | FRECCIAROSSA |
| 173 | AP | Albula Panorama | Albula Panorama (Panoramic Car) |
| 174 | BEX | Bernina Express | Bernina Express (Panorama Train) |
| 175 | GEX | Glacier Express | Glacier Express (Panorama Train) |
| 176 | GP | Golden Pass | Golden Pass (Panorama Train) |
| 177 | BNI | Bernina Panorama | Bernina Panorama (Panoramic Car) |
| 178 | zb | zb Zentralbahn AG | Luzern-Interlaken Express (Panorama Train) |
| 179 | BXB | Bernina Express Bus | Bernina Express (Panorama Bus) |
| 200 | GGB | Gornergrat Bahn | Mountain train |
| 202 | ICE | ICE-Allemagne France | ICE Allemagne-France |
| 203 |  | ÖBB-NIGHTLINE | ÖBB Night Line |
| 205 | ICP | Intercity Plus | Intercity Plus |
| 206 | RID | Riviera Day | Riviera day |
| 207 | RIN | Riviera Night | Riviera night |
| 209 | RJ | R A I L JET | Rail Jet |
| 213 | AZ | DB Autozug | DB Autozug |
| 214 |  | Berlin-Warszawa-Expresas | Berlin-Warszawa-Express |
| 215 |  | Railpromo Austria Express/Treski | Austria Express/Treski |
| 216 | PRECIOS | MERCADO | Precios Mercado |
| 219 | TGV | TGV | TGV |
| 223 | FB | FB | FernBus |
| 224 | ICB | Intercitybus | ÖBB-Intercitybus |
| 225 | TLK | TLK train | Yours Rail Lines |
| 226 | A | RailBus | RailBus |
| 227 | BUS | Replacement bus for Regional Train | Replacement bus for Regional Train |
| 228 | IR | InterREGIO train | InterREGIO train |
| 229 | IRB | Replacement bus for InterRegio train | Replacement bus for InterRegio train |
| 230 | MP | Fast International Train | Fast International Train |
| 231 | MR | musicREGIO train | musicREGIO train |
| 232 | OS | Stopping Train | Stopping Train |
| 233 | P | Fast Train | Fast Train |
| 234 | R | REGIO train | REGIO train |
| 235 | RE | REGIOekspres train | REGIOekspres train |
| 236 | VR | viaREGIO train | viaREGIO train |
| 237 | TK | TurKol | TurKol |
| 238 | EIP | EIC Premium | High-speed train |
| 239 | SKM | PKP SKM w Trojmiescie | PKP SKM w Trojmiescie |
| 240 | SA | SAPSAN | High speed train |
| 242 | STR | STRIZH | Strizh night train |
| 243 | STR | STRIZH | Strizh interregional |
| 244 | NJ | NJ | NJ Night Jet |
| 245 | CAR | AUTOCAR | French regional buses (not sold via Hermes) |
| 246 | RJX | RJX | RJX railjet xpress |
| 247 | CJX | CJX | CJX cityjet xpress |
| 248 |  | Night train BC | Night train BC |
| 249 | TGV | TGV INOUI | TGV INOUI |
| 250 | TGV | TGV INOUI | TGV INOUI DUPLEX (double decker TGV) |
| 251 | ALI | Aare Linth | Aare Linth (Panorama Train) |
| 252 | TGO | Treno Gottardo | Treno Gottardo (Panorama Train) |
| 253 | VAE | Voralpen-Express | Voralpen-Express (Panorama Train) |
| 254 | LK | FRECCIALINK | FRECCIALINK |

## Service Class

|  |  |
| --- | --- |
| Code | Description |
| BEST |  |
| HIGH |  |
| STANDARD |  |
| BASIC |  |

## Stations

codeList = UIC, ERA, HAFAS

Additional codelists can be defined by implementers. The codelist name should then be prefixed by “X\_<3 letters code for the provider>”. Example: “X\_PAO\_STATIONS”

*StationCode*: Values are depending on codelist and set is too large to be reproduced.

## Transfer Type

|  |  |
| --- | --- |
| Code | Description |
| WALK | A walk |
| OTHER | Other types of transfer (e.g. taxi, local city transport not included in the offer,…) |

## Transport Mode

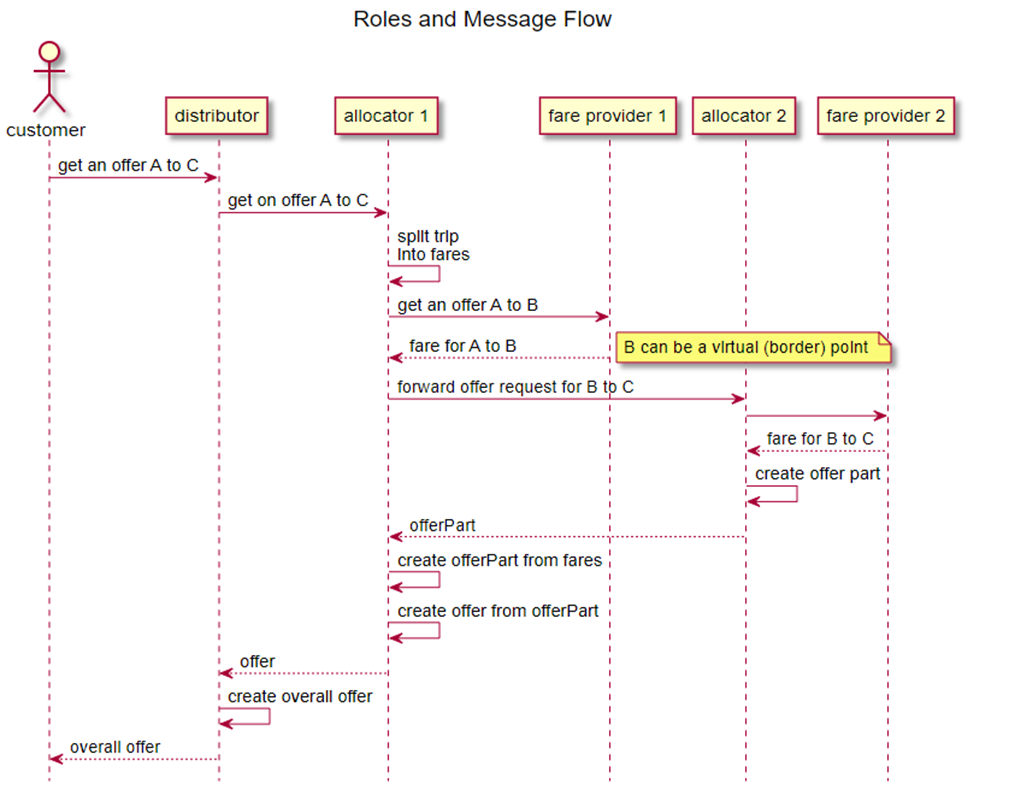
??? Need Merits input

## TravelerType

|  |  |
| --- | --- |
| Code | Description |
| DOG |  |
| ACCOMP\_DOG |  |
| LUGGAGE |  |
| BICYCLE |  |
| CAR |  |
| MOTORCYCLE |  |
| TRAILER |  |
| PRM | Person with reduced mobility |

## Compliance

To be compliant with the OSDM specification in total a party must be compliant with the offline as well as the online part of the specification. However, a party can decide based on their business need to implement the offline or the online part online based on the role they want to play in the distribution and sales process.



Compliance

## Compliance with the Offline Part

An implementation of the bulk data exchange specification is compliant with the specification if

* A feature specified in the data structure is implemented

Or

* A fare providing the feature in its data is excluded from sale

A system receiving data for a fare must be able to understand all features and rules of the fare defined in the data and obey these features and rules or must not sell such a fare.

## Compliance with the Online Part

### Compliance as an Allocator

The following services/features are mandatory/optional to implement:

|  |  |  |
| --- | --- | --- |
| Resources | Description | Need to Support |
| /locations and /trips | Resources to search for trip and locations | **Mandatory** for train stations UIC code need to be supported. |
| /trip-offers-collection and /trip-offers and /offers | Resources to get bookable offers | **Mandatory** is to provide at least admission offers. |
| /offers/{id}/admissions and /offers/{id}/reservations and /offers/{id}/ancillaries and /offers/{id}/fares | Resources to manipulate parts of an offer consisting of, e.g., admissions, reservations or ancillaries; | **Mandatory** |
| /offers/{offerId}/fares/{fareId} and /offers/{offerId}/reservations/{reservationId} | Resources to manipulate seat assignment | *Conditional*; mandatory to be supported in case seat assignment is provided. |
| /bookings | Resources to manipulate bookings | **Mandatory** |
| /offers/{id}/passengers and /bookings/{id}/passengers | Resources to manipulate passenger information at every stage of the flow | **Mandatory** |
| /products | Resources to retrieve products information on one or more products | **Mandatory** |
| /bookings/{id}/fulfillments and /fulfillments | Resources to retrieve fulfillments, e.g. tickets | **Mandatory** is to support A4 PDF tickets. |
| /bookings/{id}/refundOffers | Resources to get and accept a refund offer | **Mandatory** is to support full refund. |
| /bookings/{id}/exchangeOffers | Resources to get and accept an exchange offer | *Optional* |
| /coachLayouts | Resources to get layouts of coaches | *Optional* |

All non-functional requirements defined in the services must be fulfilled.

Especially all implementations most support the Tolerant Reader pattern. This integration pattern helps creating robust communication systems. The idea is to be as tolerant as possible when reading data from another service. This way, when the communication schema changes, the readers must not break.

### Compliance as a Fare Provider

The following services/features are mandatory/optional to implement:

|  |  |  |
| --- | --- | --- |
| Resources | Description | Need to Support |
| /locations and /trips | Resources to search for trip and locations | *Optional* |
| /trip-offers-collection and /trip-offers and /offers | Resources to get bookable offers | **Mandatory** is to provide fare offers on a /trip-offers-collection. *Optional*: /offers for non trip based offers. |
| /offers/{id}/admissionsand /offers/{id}/reservations and /offers/{id}/ancillaries and /offers/{id}/fares | Resources to manipulate parts of an offer consisting of, e.g., admissions, reservations or ancillaries; if permitted, also fares are offered. | **Mandatory** is to support pre-booking and booking of fares. |
| /bookings | Resources to manipulate bookings | **Mandatory** is to support bookings consisting of fares except in the special case of direct sale fare offers |
| /offers/{id}/passengers and /bookings/{id}/passengers | Resources to manipulate passenger information at every stage of the flow | *Conditional*, mandatory to be supported in case personal data are required by the fare provider |
| /products | Resources to retrieve products information on one or more products | **Mandatory** |
| /offers/{offerId}/fares/{fareId} and /offers/{offerId}/reservations/{reservationId} | Resources to manipulate seat assignment | *Conditional*, to be supported in case seat assignment is provided. |
| /bookings/{id}/fulfillments and fulfillments | Resources to retrieve fulfillments, e.g. tickets | *Conditional*, mandatory to be supported in case fulfillment items need to be provided |
| /bookings/{id}/refundOffers | Resources to get and accept a refund offer | **Mandatory** is to support for full refund, partial refund is optional. Not required in the special case of direct sale offers |
| /bookings/{id}/exchangeOffers | Resources to get and accept an exchange offer | *Optional* |
| /coachLayouts | Resources to get layouts of coaches | *Optional* |

All non-functional requirements defined in the services must be fulfilled.

On a technical level the implementation must support the Tolerant Reader pattern. This integration pattern helps creating robust communication systems. The idea is to be as tolerant as possible when reading data from another service. This way, when the communication schema changes, the readers must not break.

An implementation of specification is compliant with the specification if for a given version

* A feature specified in the data structure is implemented

Or

* A fare providing the feature in its data is excluded from sale

A system receiving data for a fare must be able to understand all features and rules of the fare defined in the data and obey these features and rules or must not sell such a fare.