EUROPEAN UNION AGENCY FOR RAILWAYS

Moving Europe towards a sustainable and safe railway system without frontiers.

Electronic reservation of seats/berths and electronic production of travel documents - exchange of messages - Annex B1 - OSDM

Telematics TSI - Technical Document - B5 - Annex B1 - OSDM

Version 4.0

Contents

A. Document management

A.1 Document properties

- File name: ERA_TD_B5_B1.docx
- Subject and document type: Telematics TSI - Technical Document - B5 - Annex B1 - OSDM
- Author: European Railway Agency
- Version: 4.0

A.2 Change management

Updates to this technical document shall be subject to Change Control Management procedure managed by the Agency pursuant:

- the applicable requirements in the reference TSI
- Art. 23(2) of the Agency Regulation

If necessary, working groups are created in line with Art. 5 of the Agency Regulation.

A.3 Configuration management

A new version of the document will be created if new changes are considered following the Change Control Management Process led by ERA.

More specifically:

- if there is a change in the requirements which influences the implementation
- if information is added to or deleted from the technical document
- adding test cases to the field checking in messages or databases.

Modifications will have to be highlighted, so they can be easily identified.

A.4 Availability

The version in force of this document is available on Agency's Gitlab repository. Any printed copy is uncontrolled.

A.5 Application and actors in the scope

Date of entry into force of reference TSI.

This document applies to all the actors in the scope of the reference TSI.

A.6 Document history

Table 1 - Document history

Version	Date	Comments
4.0	10.06.2025	Initial version for Telematics TSI

B. Acronyms, definitions and external references

B.1 Acronyms

Please refer to Chapter 3.

B.2 Definitions

Terms contained in this document are defined in the ERA Ontology. A specific set of terms is defined also in chapter 2

B.3 External references

The referenced documents listed in Table 2 are indispensable for the application of this document:

- For dated references, only the edition cited applies;
- For undated references, if any, the latest edition of the referenced document (including any amendments) applies.

Table 2 Reference documents

ID	Title	Doc ID, Edition	Date	Author/ Publisher
[1]	None			

1. Foreword

The main goal of this Appendix to TAP-TSI B.5 is to specify the messages of the OSDM distribution model that are needed to be used as alternative to the TAP-TSI B.5 XML API.

A mapping of the XML/SOAP services and the required JSON/REST services per use case is included in the chapter Mapping of APIs (Appendix A to Appendix B).

1.1. Objectives

The main objectives guiding this specification were:

- To support the essential functions of the
- To provide an alternative to the outdated XML/SOAP API on appendix A.1 and A.2 with a subset of the Open Sales and Distribution Model API.

1.2. Summary

Regarding the sales process, the API includes the following steps:

1. Getting offers
2. Booking an offer
3. Confirmation of the booking

Analogously, the after-sale process is modelled in the following steps:

1. Getting a refund/exchange offers
2. Booking a refund/exchange offer

2. Terms and Definitions

Term	Definition
Fulfillment	A fulfillment is a document (either for paper printing or electronically) provided to the passenger to prove his travel right, facilitate access to trains and stations (e.g. via gates), provide further information on the travel and provide access to services either directly or via exchange (voucher).
Individual ticketing	A separate ticket is created per passenger.
Individual contracts	A separate ticket is created per passenger and these tickets can be treated as individual contracts of carriage. After sales transactions can be applied independently per passenger and ticket.
Integrated Reservation Ticket (IRT)	Ticket for a specific train on a travel day usually including the seats. All tickets for a train are managed in one central system of the distributor. The ticket is valid on that train on a certain day only.
Non-integrated Reservation Ticket (NRT)	A ticket not including an integrated reservation. Multiple distributors can create tickets for the same route independently. The distributor of the ticket is usually the same company that issues the ticket. The ticket might be applicable to a route with many trains or a zone or a list of trains or combinations of these. The validity might be more than one day. Some conditions allow a partial refund on unused parts of the ticket route. Refund can be done via the retailer. These conditions depend on the fare providers and the distributors (i.e. providing the option of reducing the number of passengers or to interrupt the journey). NRTs not linked to a train might be reused in case the use is not tracked.
Offer Part	An abstraction of things that can be offered. Can be of type Admission, Reservation or Ancillary. Corresponding to offer parts the booking contains booking parts (= booked offer parts).

3. Acronyms

Acronyms	Acronym Description
IRT	Integrated Reservation Tariff: Tariff used for Integrated-Reservation-Tickets.
IRT	Integrated Reservation Ticket: Ticket including mandatory reservation.
JWT	JSON Web Token: Specification to transport authentication information used by the OAUTH2 authorization protocol. JSON Web Token - RFC 7519
NRT	Non-Integrated Reservation Tariff: Tariff used for Non-Integrated-Reservation-Tickets.
NRT	Non-Integrated Reservation Ticket: Ticket not including an integrated reservation.
REST	Representational State Transfer (REST): REST is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations.
UUID	Universally Unique Identifier: Standard to create a unique id. The specification is published as ISO/IEC 9834-8:2005.

4. Mapping of APIs (Appendix A to Appendix B)

Whereas the XML Api (Appendix A2) differs from the Binary Message API (Appendix A1) in the format of the data send the JSON API is a REST API and differs by different design of the messaging. Therefore, there is not a one-to-one mapping between messages. A converter needs to map between different designs to implement similar use cases.

Basic use Cases:

Use Case	XML/SOAP Implementation	JSON/REST Implementation
get reservations offers / fare list	`AvailabilityRequest`	`POST /offers(searchCriteria)` or `POST /offers(tripSpecification)` depending on the XML version used
get reservation offer	`BookingRequest` with `informationOnly=true`	`POST /offers(searchCriteria)` or `POST /offers(tripSpecification)` depending on the XML version used

Seat selection for Reservations	`BookingRequest near-by`
	
	`BookingRequest / BookingRequest with informationOnly=true`
	
	`using preference parameter`
	
	`using specific seat numbers`
	
	`GET CoachLayouts`
		To get the possible seat parameters:
		
		`POST /availabilities/place-map`
		`GET /availabilities/nearby`
		`GET /availabilities/preferences`
		`GET /coach-layouts`
		
		To set the seat parameters:
		
		`PATCH /bookings/{bookingId}`

book offer	`BookingRequest`	`POST /bookings`
		
		followed by
		
		`POST /booking/{bookingId}/fulfillments`

Use Case	XML/SOAP Implementation	JSON/REST Implementation

		In case of technical problems a booking can be retrieved or deleted:
		
		`GET /bookings/{bookingId}`
		`DELETE /bookings/{bookingId}`

cancel booking	`CancelRequest`
	
	with `reason code`
		`POST /bookings/{bookingId}/refundOffers`
		
		followed by
		
		`PATCH /bookings/{bookingId}/refund-offers/{refundOfferId}`
		
		to confirm the offer and
		
		`DELETE /bookings/{bookingId}/refund-offers/{refundOfferId}`
		
		`GET /bookings/{bookingId}`
		
		in case of technical problems.

refund booking	`CancelRequest`	`POST /bookings/{booking}/refundOffers`
		
		followed by
		
		`PATCH /bookings/{bookingId}/refund-offers/{refundOfferId}`
		
		to confirm the offer and
		
		`DELETE /bookings/{bookingId}/refund-offers/{refundOfferId}`
		
		in case of technocal problems

technical rollback	`synchro request`	`DELETE /bookings/{bookingId}`

Advanced use cases:

Use Case	Hermes Hosa	OSDM
booking with personal data	`BookingRequest` including personal data	`PATCH /bookings/{bookingId}/passengers/{passengerId}`

partial cancellation / reducing the number of passengers	`ParcialCancellation-Request`	`POST /bookings/{booking}/refundOffers`
		
		followed by
		
		`PATCH /bookings/{bookingId}/refund-offers/{refundOfferId}`
		
		to confirm the offer and
		
		`DELETE /bookings/{bookingId}/refund-offers/{refundOfferId}`
		
		in case of technocal problems.

exchange booking	`BookingRequest`
	
	including an exchange reference to the original booking
		`POST /bookings/{bookingId}/exchange-offers`
		
		followed by
		
		`POST /bookings/{bookingId}/exchange-operations`
		
		to select the exchange offer and
		
		`PATCH /bookings/{bookingId}/exchange-operations/{exchangeOperationId}`
		
		to confirm it.
		
		`DELETE /bookings/{bookingId}/exchange-operations/{exchangeOperationId}`
		
		is needed for cleanup in case of technical problems.

5. Use Cases covered

5.1. Get Offers

The offer is requested by the Distributor and the response is provided by the Attributor.

A list of offers is requested. The XML/SOAP API requires two services, one to request a list of available tariffs and a second one to request the concrete price for a combination of travelers with selected tariffs. Both services need a dedicated train in the request.

The corresponding JSON/REST API provides one service to request a list of offers for a dedicated number of travelers. The offer request is provided with two options, one including a trip specification and one trip search criteria. The trip specification version will provide the dedicated train for the offer.

- `POST /offers(searchCriteria)`
- `POST /offers(tripSpecification)`

5.2. Create a booking from the offer

The Create booking is requested by the Distributor and the response is provided by the Attributor.

The XML/SOAP API requires to resend the request parameters to create the booking. As the XML/SOAP API does not provide a separate commit of the booking the error handling must be implemented on the consumer side und special services for rollback.

The corresponding JSON/REST API provides a service to create a booking based on the offer received in prebooked status. A separate commit is required to finalize the booking, thus there is no need for a transaction handling on consumer side to create a booking.

- `POST /bookings`

In case of technical problems a booking can be retrieved or deleted:

- `GET /bookings/{bookingId}`
- `DELETE /bookings/{bookingId}`

5.3. Set and change seat parameters

The Set and change seat parameters is requested by the Distributor and the response is provided by the Attributor.

The XML/SOAP API requires to send the seat parameter with the booking request. The client has to know the possible attributes.

The corresponding JSON/REST API provides a service to set or change the seat attributes after an offer was prebooked.

To get the possible seat parameters:

- `POST /availabilities/place-map`
- `GET  /availabilities/nearby`
- `GET  /availabilities/preferences`

- `GET /coach-layouts`

To set the seat parameters:

- `PATCH /bookings/{bookingId}`

5.4. Cancel a Booking

The Cancel a Booking is requested by the Distributor and the response is provided by the Attributor.

The XML/SOAP API provides a service to cancel a single booking. To get a refund offer this service is used in information-only mode. If accepted the request is repeated in real mode.

The corresponding JSON/REST API provides a service to get an offer for refund for a booking or some fulfillments within a booking.

- `POST /bookings/{booking}/refundOffers`

To select and confirm the refund offer:

- `PATCH /bookings/{bookingId}/refund-offers/{refundOfferId}`

A refund offer can explicitly be deleted in case of errors, either technical or by the client.

- `DELETE /bookings/{bookingId}/refund-offers/{refundOfferId}`

5.5. Partial Refund of a Booking

The Partial Refund of a Booking is requested by the Distributor and the response is provided by the Attributor.

The XML/SOAP API provides a service to cancel some of the travelers in a booking.

The corresponding JSON/REST API provides a service to request a refund offer and a refund based on a refund offer. The same request allows the selection of travelers.

- `POST /bookings/{booking}/refundOffers`

To select and confirm the refund offer:

- `PATCH /bookings/{bookingId}/refund-offers/{refundOfferId}`

A refund offer can explicitly be deleted in case of errors, either technical or by the client.

- `DELETE /bookings/{bookingId}/refund-offers/{refundOfferId}`

5.6. Change and add Personal Data

The Change and add Personal Data is requested by the Distributor and the response is provided by the Attributor. This request is needed in case of personlised tickets, issued on the name of the traveller.

The XML/SOAP API allows to add personal data in the booking request. It also provides a service to change personal data.

The corresponding JSON/REST API provides a service to change change personal data.

- `PATCH /bookings/{bookingId}/passengers/{passengerId}`

5.7. Exchange a Booking

The Exchange a Booking is requested by the Distributor and the response is provided by the Attributor.

The XML/SOAP API allows to reference a booking in a new booking request for exchange. A service to request a list of offers for exchange is not defined. The client needs to request a list of new offers and then use the information-only mode to request the dedicated exchange offer prices one by one.

The corresponding JSON/REST API provides a service to request an exchange offer for an existing booking. An exchange offer needs to be selected.

- `POST /bookings/{bookingId}/exchange-offers`

To select the exchange offer and create an exchange operation:

- `POST /bookings/{bookingId}/exchange-operations/{exchangeOperationId}`

To confirm the exchange operation and receive fulfillments:

- `PATCH /bookings/{bookingId}/exchange-operations/{exchangeOperationId}`

To in case of technical problems:

- `DELETE /bookings/{bookingId}/exchange-operations/{exchangeOperationId}`

6. Technical Principles

6.1. 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.
- Use semantic versioning.
- 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
- 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.
- Creation/ modification calls return the created/modified resource (not just an ok code)

6.2. Authentication

The general requirement of the OSDM standard to use OAuth2 for authentication and authorization by means of JW tokens (JWTs) should be implemented in one of two consistent methods. Either by:

- Using JWTs for Client Authentication according to RFC-7523 Section 2.2, (recommended) or by
- Using Client Credentials for Access Token Request according to RFC.6749 Section 4.4.2

The following RFC documents apply:

- RFC-7519 which explains what a JWT token is;
- RFC-6749 Section 3.2 which defines OAuth2 and the token endpoint involved in the creation of tokens;
- RFC-6749 Section 4.4.2 which defines the use of client credentials to obtain an access token;
- RFC-7521 laying the groundwork for cryptographic client assertions;
- RFC-7523 Section 2.2 which describes how to properly secure the token endpoint with modern cryptography, thus not relying on static secrets;
- RFC-8725 which gives guidance on securely validating and using JWTs.

This document defines the two variants of flows to be used. It also defines the parameters used, which must be agreed and exchanged bilaterally between the parties involved.

6.2.1. *Using JWTs for client authentication*

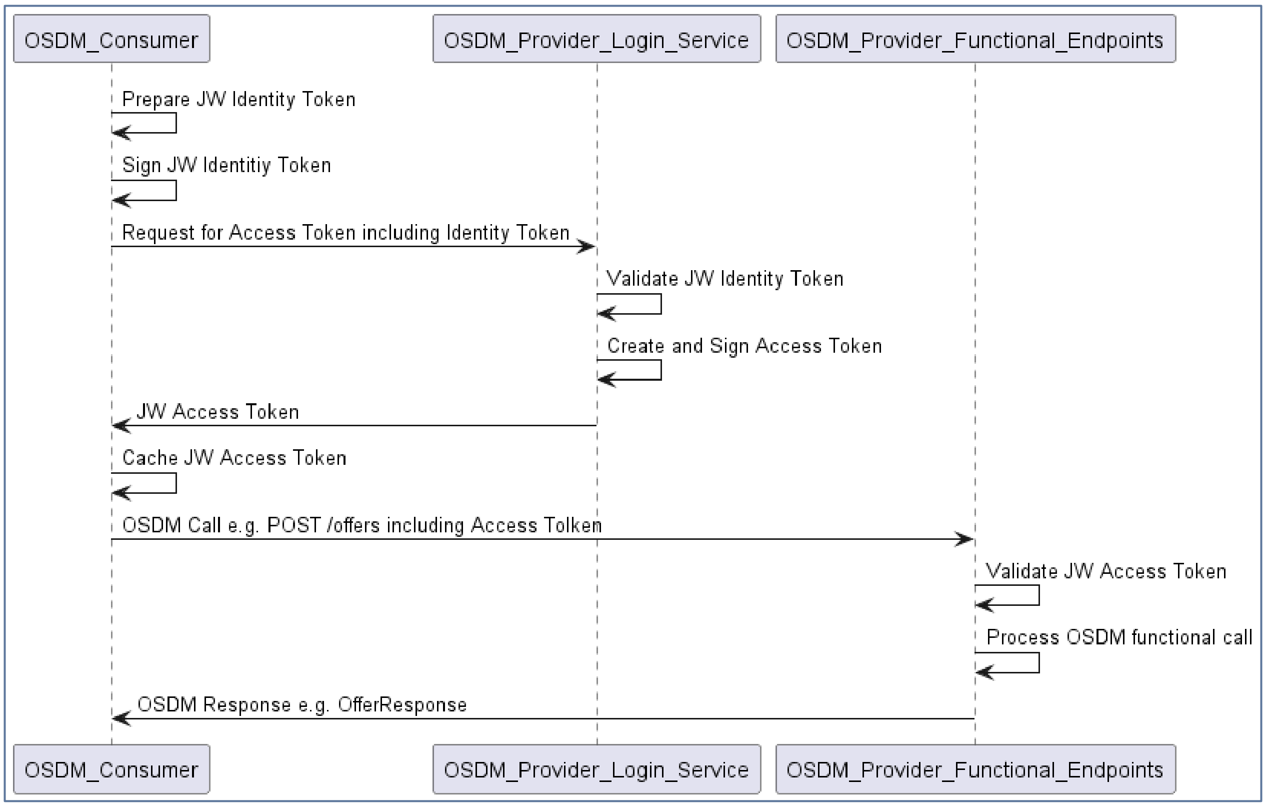
This flow uses a client authentication assertion in the form of a JW identity token (private_key_jwt in terms of OpenID Connect (OIDC)), which is cryptographically signed by the client (OSDM consumer) and can be verified by the server (OSDM provider). It is the recommended

The OSDM provider then issues a JW access token which can in turn be used by the OSDM consumer to prove their access rights to the OSDM endpoints by providing the JW access token in the Authenticate header of the OSDM endpoint invocation.

This method makes it unnecessary to exchange actual client secrets between the consumers and providers of the service and relies on asymmetric cryptography, i.e., the use of private/public keys for signing such requests.

6.2.2. *Flow using JWTs*

The general flow between consumer and provider looks like this:


OSDM_Consumer
OSDM_Provider_Login_Service
OSDM_Provider_Functional_Endpoints
Prepare JW Identity Token
Sign JW Identitiy Token
Request for Access Token including Identity Token
Validate JW Identity Token
Create and Sign Access Token
JW Access Token
Cache JW Access Token
OSDM Call e.g. POST /offers including Access Tolken
Validate JW Access Token
Process OSDM functional call
OSDM Response e.g. OfferResponse
OSDM_Consumer
OSDM_Provider_Login_Service
OSDM_Provider_Functional_Endpoints

In this flow, the following services are defined:

- The **OSDM Consumer** is the engine trying to issue OSDM “functional” calls, e.g., do a POST /offers

- The **OSDM Provider Login Service** is the service which controls the authentication of the OSDM Consumer by issuing JW access tokens
- The **OSDM Provider Functional Endpoints** implement the actual business logic of the OSDM Provider. Calls to these endpoints need to be authorized by providing the appropriate JW access tokens.

6.2.3. JW identity token

A JW token (JWT) consists of three parts which are separated by dots (“.”):

- The JWT header
- The JWT payload
- The JWT signature

Each part is separately encoded using Base64URL encoding. The encoded header and payload fields together are signed using the agreed-upon algorithm (which is also specified within the header), then the signature is added to the end of the token. Thus a complete JW token has the form <JWT Header>.<JWT Payload>.<Signature>, where each part is separately Base64Url encoded.

Header and Payload of the JW token are encoded as JSON structures. Their content is defined in the following sections.

6.2.4. JW identity token header

The identity token contains the following header fields. Where some fields are optional according to the relevant RFC, we still consider them mandatory for the purposes of usage within the OSDM standard.

Attribute	RFC requirement	OSDM requirement	Description	Recommended value
alg	REQUIRED	MANDATORY	Algorithm used for signing this token	to be agreed between parties, use at least RS256
kid	OPTIONAL	MANDATORY	Id of the key used for signing this token	defined by OSDM consumer. Should be provided to the OSDM provider.
typ	REQUIRED	MANDATORY	Type of the token	fixed value “JWT”

6.2.5. *JW identity token payload*

Attribute	RFC requirement	OSDM requirement	Description	Recommended value
iss	REQUIRED	MANDATORY	Issuer of the identity token	defined by the OSDM provider (client id)
sub	REQUIRED	MANDATORY	Identity of the client	defined by the OSDM provider (client id)
aud	REQUIRED	MANDATORY	URL login service endpoint	defined by the OSDM provider
exp	REQUIRED	MANDATORY	Timestamp when this request expires	current time + grace period of at least 2 minutes (120 seconds)
scope	OPTIONAL	MANDATORY	Usage of the token	defined by the OSDM provider
nbf	OPTIONAL	OPTIONAL	Timestamp when request begins to be valid	current time - grace period of at least 2 minutes (120 seconds)
iat	OPTIONAL	OPTIONAL	Timestamp of the creation of the token	current time
jti	OPTIONAL	MANDATORY	Unique ID of the token to prevent replays	newly generated UUID

Note: All timestamps are in “Unix epoch”, which is defined as the number of seconds since 1st January, 1970 UTC.

Note on the scope parameter: some IDP (Identity Provider) products need this value to be set to a specific value.

6.2.6. JW identity token signature

The signature is obtained by creating the string <JWT Header Base64URL encoded>.<JWT Payload Base64URL encoded>, and signing this string with the private key of the OSDM consumer using the algorithm specified in the JWT header field "alg". The signature is then also Base64URL encoded and added to the token.

6.2.7. Request for an access token using JWTs

To obtain the actual JW access token required to authenticate the functional OSDM requests, a token request message needs to be issued to the OSDM provider's login service. This has the following attributes:

- grant_type=client_credentials
- client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
- client_assertion=<JWT>
- scope=<scope as defined by provider>

The <JWT> means the JW identity token which has been described above.

The provider should set the **expires_in** attribute of the response, so that the consumer does not need to parse the token content.

Consumers should cache the access token in order to avoid overloading the provider's login server, using the value from the expires_in attribute to invalidate cache entries.

6.2.8. Validation of identity token

The provider's login service should take certain steps in order to validate the identity token received from the consumer before it issues the access token. The most important include:

- Check the signing algorithm against a whitelist of allowed algorithms or against a pinned value for the consumer, to mitigate these attacks:
    - A third party (attacker) sends an identity token with alg: none, disabling signature verification and allowing token forgery.
  - A third party (attacker) sends an identity token with alg set to a symmetric algorithm, but kid set to an asymmetric key. This allows token forgery if the public key is known
- Check that sub, iss, kid and the public key match (note: the provider should store this data), to mitigate this attack:
  - Consumer signs an identity token for another consumer's username with their own key
- Check the signature of the token against the public key of the requestor, to mitigate this attack:
  - A third party (attacker) tries to request an access token without knowledge of the secret key, but tries anyway
- Retire old kid/public key pairs shortly after key rollover, to mitigate this attack:
  - A third party (attacker) uses an old, leaked private key to impersonate a valid consumer

- Store the value in the jti field for at least the time specified in the exp field. Do not respond to requests with a jti field contents that match a stored value, to mitigate this attack:
  - A third party (attacker) replays a valid signing request to obtain an access token

Additionally, implementers should consult RFC 8725 for guidance on securely validating and using JWTs, both in the login service and in the functional endpoints.

6.2.9. *Configuration for identity JWTs*

Some configuration parameters need to be agreed upon bilaterally between the partners. They are listed in the following table.

Parameter	Usage	Explanation	Parameter flow
Signing algorithm	JW identity token, header field 'alg'	Algorithm used for signing the JW identity token	Mutually agreed between provider and consumer
Signing key ID	JW identity token, header field 'kid'	Key used for signing the JW identity token	Defined by consumer
Issuer of JW identity token	JW identity token, payload field 'iss'	Identity of the client within the provider's system	Defined by provider
Subject of the access token request	JW identity token, payload field 'sub'	Identity of the client within the provider's system	Defined by provider
Audience of the access token request	JW identity token, payload field 'aud'	URL of the login service	Defined by provider
Public key	Validation of signature within the provider's login service	Public key used for validating signature of identity token	Defined by consumer, part of the public/private key pair

When the signing key pair is **rotated** (which should happen on a regular basis), the consumer needs to provide the new signing key ID and the new public key to the provider, and the provider should - for a limited time - accept either pair of key_ID/public_key for validation.

6.2.10. *Examples*

Some examples are provided here.

6.2.10.1. Example JW identity token

1.1.1.1.1 JSON structure

Header:

```
{
  "alg" = "RS256",
  "typ" = "JWT",
  "kid" = "1234567890"
}
```

Payload:

```
{
  "iss" = "UIC_OSDM_1080_4",
  "sub" = "UIC_OSDM_1080_4",
  "aud" = "https://osdm-provider.eu/logon-server/public/token",
  "exp" = "1709041312",
  "scope" = "uic_osdm",
  "nbf" = "1709040292",
  "iat" = "1709040412",
  "iti" = "e57b0901-19cf-471e-81fe-61b6a7ee19b7"
}
```

1.1.1.1.2 Encoded token

```
eyJhbGciPSJSUzI1NiIsInR5cCI9IkpXVCIsImtpZCI9IjEyMzQ1Njc4OTAifQ.eyJpc3MiPSJVS
UNfT1NETV8xMDgwXzQiLCJzdWIiPSJVSUNfT1NETV8xMDgwXzQiLCJhdWQiPSJodHRwczovL29zZ
G0tcHJvdmlkZXIuZXUvbG9nb24tc2VydmVyL3B1YmxpYy90b2tlbiIsImV4cCI9IjE3MDkwNDEzM
TIiLCJzY29wZSI9InVpY19vc2RtIiwibmJmIj0iMTcwOTA0MDI5MiIsImlhdCI9IjE3MDkwNDA0M
TIiLCJpdGkiPSJlNTdiMDkwMS0xOWNmLTQ3MWUtODFmZS02MWI2YTdlZTE5YjcifQo.<signatur
e>
```

6.2.10.2. Example request

```
POST /logon-server/public/token
Host: osdm-provider.eu
Content-Type: application/x-www-form-urlencoded
grant_type=client_credentials
&client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-
bearer
&client_assertion=eyJhbGciPSJSUzI1NiIsInR5cCI9IkpXVCIsImtpZCI9IjEyMzQ1Njc4OT
AifQ.eyJpc3MiPSJVSUNf...
&scope=uic_osdm
```

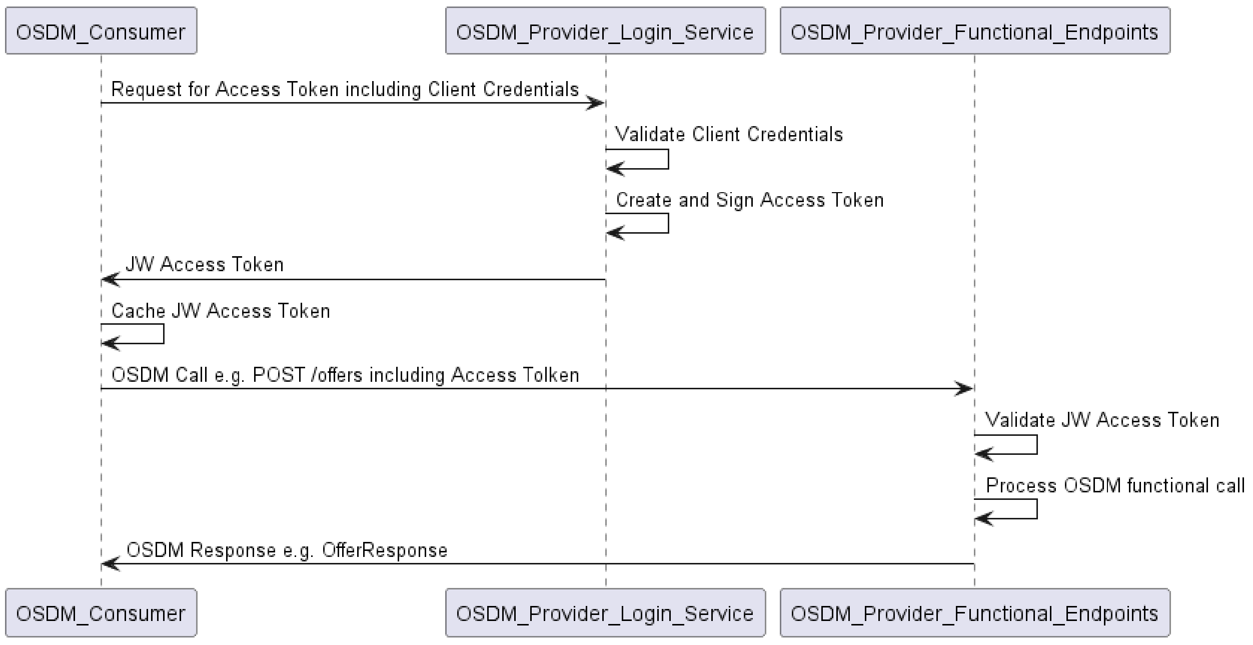
6.2.11. Using client credentials for access token request

As an alternative to the flow described above using a signed JWT as the client credential, a simpler method can - at the discretion of the OSDM provider, and subject to mutual agreement with their consumers - also be used. The resulting JWT access tokens are identical to those obtained by the other method, so there is no difference for the subsequent invocations of functional OSDM endpoints.

This method - according to RFC-6749 Section 4.4.2 - relies on the exchange of a client_id and a client_secret between the OSDM provider and the OSDM consumer.

6.2.12. Flow using client credentials

The flow to be implemented by the OSDM consumer is somewhat simpler to the other method.


OSDM_Consumer
OSDM_Provider_Login_Service
OSDM_Provider_Functional_Endpoints
Request for Access Token including Client Credentials
Validate Client Credentials
Create and Sign Access Token
JW Access Token
Cache JW Access Token
OSDM Call e.g. POST /offers including Access Tolken
Validate JW Access Token
Process OSDM functional call
OSDM Response e.g. OfferResponse
OSDM_Consumer
OSDM_Provider_Login_Service
OSDM_Provider_Functional_Endpoints

6.2.13. Request for an access token using a client secret

To obtain the actual JW access token required to authenticate the functional OSDM requests, a token request message needs to be issued to the OSDM provider’s login service. This has the following attributes:

- grant_type=client_credentials
- client_id=<client_id>
- client_secret=<client_secret>
- scope=<scope as defined by provider>

The provider should set the **expires_in** attribute of the response, so that the consumer does not need to parse the token content.

Consumers should cache the access token in order to avoid overloading the provider’s login server, using the value from the expires_in attribute to invalidate cache entries.

6.2.14. Validation of client credentials

The login service needs to validate that client_id and client_secret match. To provide additional security, it may also include other methods, like checking the IP address of the requestor against a defined range of allowed IP addresses.

6.2.15. Configuration of client credentials

Some configuration parameters need to be agreed upon bilaterally between the partners. They are listed in the following table:

Parameter	Usage	Explanation	Parameter flow
Client id	Token request, parameter 'client_id'	Identity of the client within the provider's system	Defined by provider
Client secret	Token request, parameter 'client_secret'	Secret ('password') of the client within the provider's system	Defined by provider
Scope	Token request, parameter 'scope'	Scope (usage) of the token	Defined by provider

Note on the scope parameter: some IDP (Identity Provider) products need this value to be set to a specific value.

When credentials need to be **rotated** (which should happen on a regular basis), the provider needs to provide the consumer with a **second** client_id/client_secret pair and needs, for a limited time, to accept either for validation. When the consumer has switched to the new client_id/client_secret pair, the original pair should be disabled in the provider's system.

6.2.16. *Example request using client credentials*

```
POST /logon-server/public/token
Host: osdm-provider.eu
Content-Type: application/x-www-form-urlencoded
grant_type=client_credentials
&client_id=UIC_OSDM_1080_4
&client_secret=MXCleFO22w2yAfolea75lrIE5RdqimPL
&scope=uic_osdm
```

6.2.17. *Notes on access tokens*

The access tokens provided by the login service behave according to the relevant standards, particularly:

- with the backend-to-backend flows described in this document, the login service only provides an **access token** - no refresh tokens are provided
- the expiry time of the access token is provided in the **expires_in** element of the response
- there is no need for the consumer to parse the access token
- a consumer may request multiple access tokens in parallel requests (e.g. from multiple instances of their services), in particular:
- requesting a new access token does **not** invalidate a previously issued access token for the same set of credentials

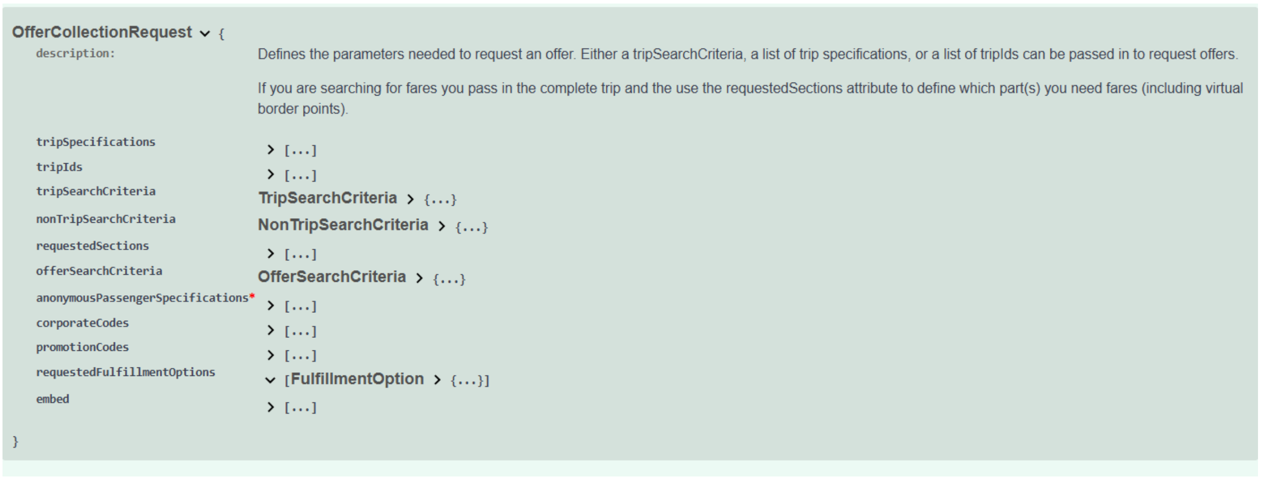
7. Services by Resource

7.1. /offers

The offer resource does not have a state. It disappears after it was used to create a booking or after the end of the offer time limit.

The offer list is created by a POST /Offers request. Either a trip specification or trip search parameters must be provided.

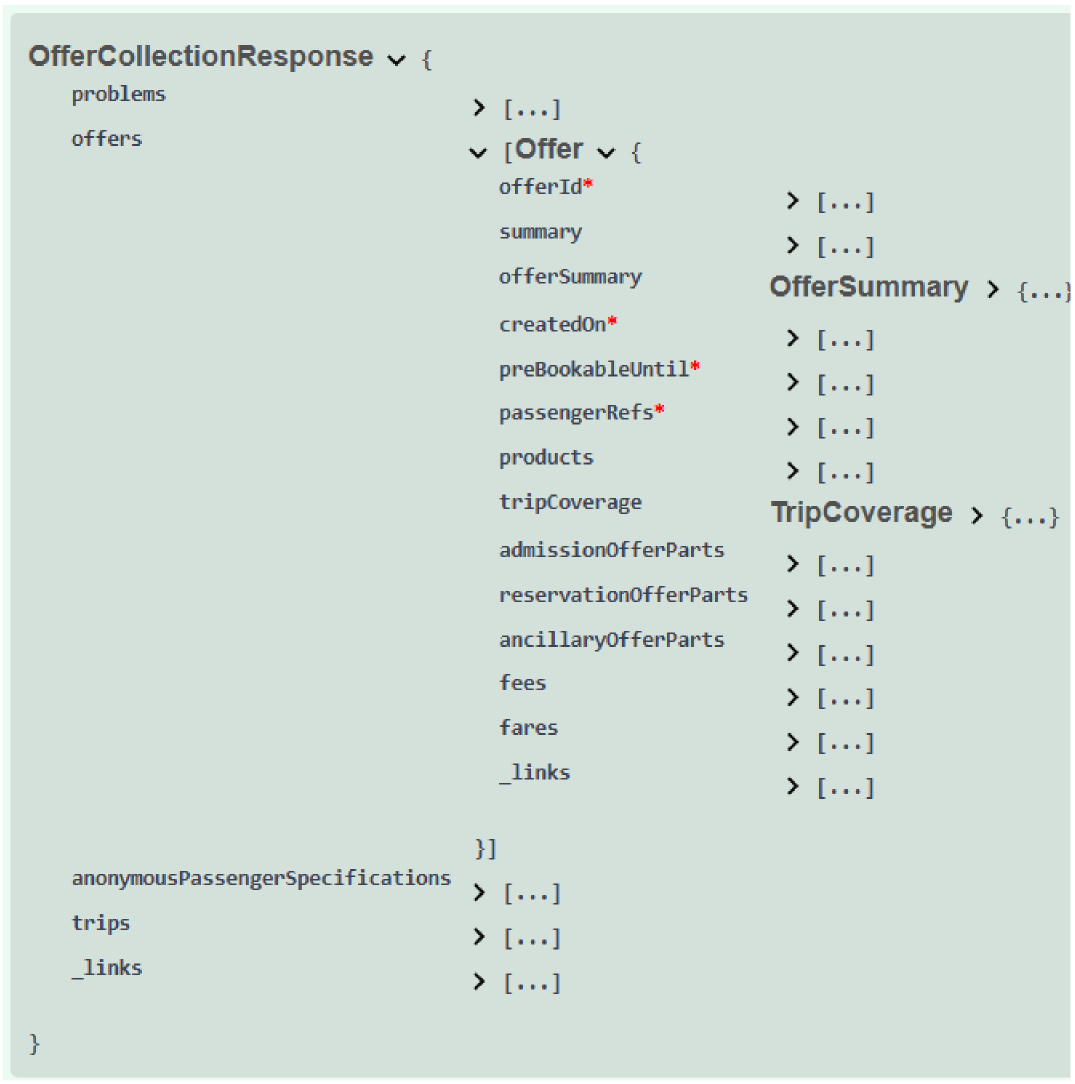
A list of anonymous passenger information is mandatory. Personal data will not be included before the booking step.



```
OfferCollectionRequest ⌄ {
    description:                    Defines the parameters needed to request an offer. Either a tripSearchCriteria, a list of trip specifications, or a list of tripIds can be passed in to request offers.

                                    If you are searching for fares you pass in the complete trip and the use the requestedSections attribute to define which part(s) you need fares (including virtual border points).

    tripSpecifications              > [...]
    tripIds                         > [...]
    tripSearchCriteria              TripSearchCriteria > {...}
    nonTripSearchCriteria           NonTripSearchCriteria > {...}
    requestedSections               > [...]
    offerSearchCriteria             OfferSearchCriteria > {...}
    anonymousPassengerSpecifications* > [...]
    corporateCodes                  > [...]
    promotionCodes                  > [...]
    requestedFulfillmentOptions     ⌄ [FulfillmentOption > {...}]
    embed                           > [...]
}
```

The response will provide a list of offers including offer parts to cover different aspects of the offer.

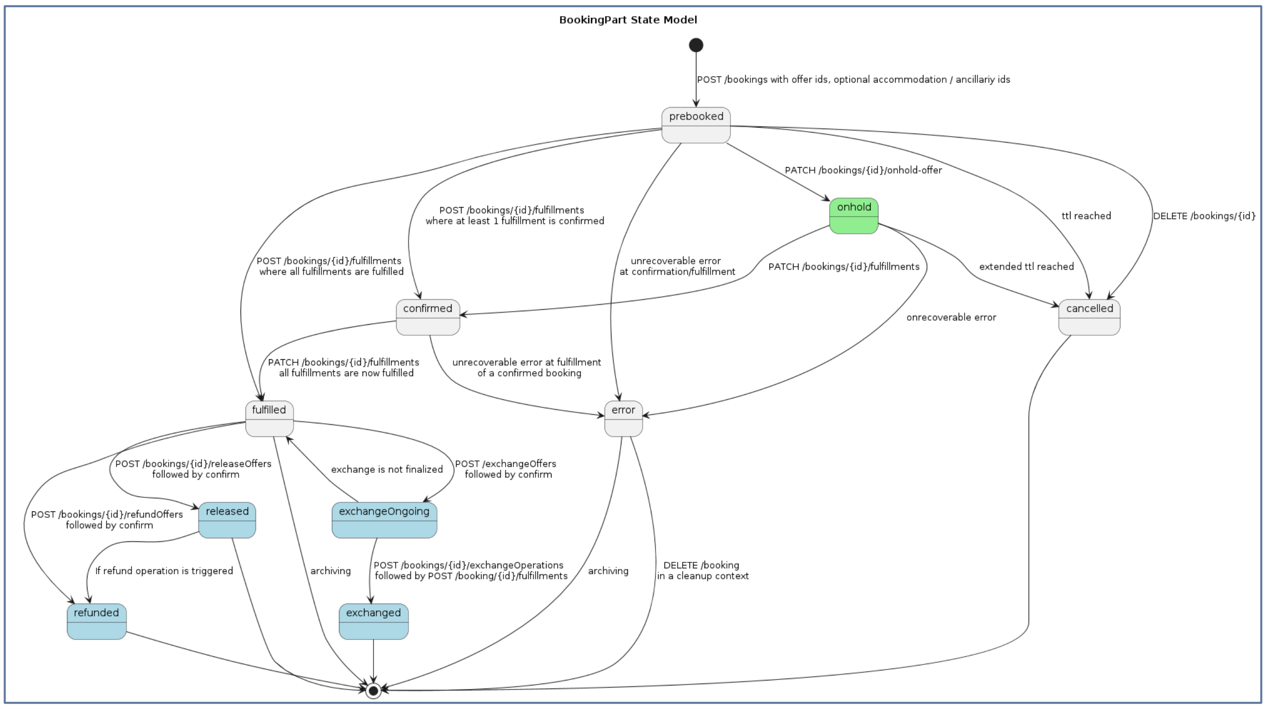
- admissionOfferParts or fares – the transport contract part (e.g. of an IRT)
- reservationOfferParts – seats
- ancillaryOfferParts – other transport parts (e.g. for bicycle transport)
- products – the list of products referenced in offer parts
- trip – the trip specification. The trip might be changed in case the requested trip is not any more available (e.g. changed stop times or changed train number)



```
OfferCollectionResponse ⌄ {
    problems                            > [...]
    offers                              ⌄ [Offer ⌄ {
                                           offerId*                  > [...]
                                           summary                   > [...]
                                           offerSummary              OfferSummary > {...}
                                           createdOn*                > [...]
                                           preBookableUntil*         > [...]
                                           passengerRefs*            > [...]
                                           products                  > [...]
                                           tripCoverage              TripCoverage > {...}
                                           admissionOfferParts       > [...]
                                           reservationOfferParts     > [...]
                                           ancillaryOfferParts       > [...]
                                           fees                      > [...]
                                           fares                     > [...]
                                           _links                    > [...]

                                        }]
    anonymousPassengerSpecifications    > [...]
    trips                               > [...]
    _links                              > [...]

}
```

7.2. /bookings

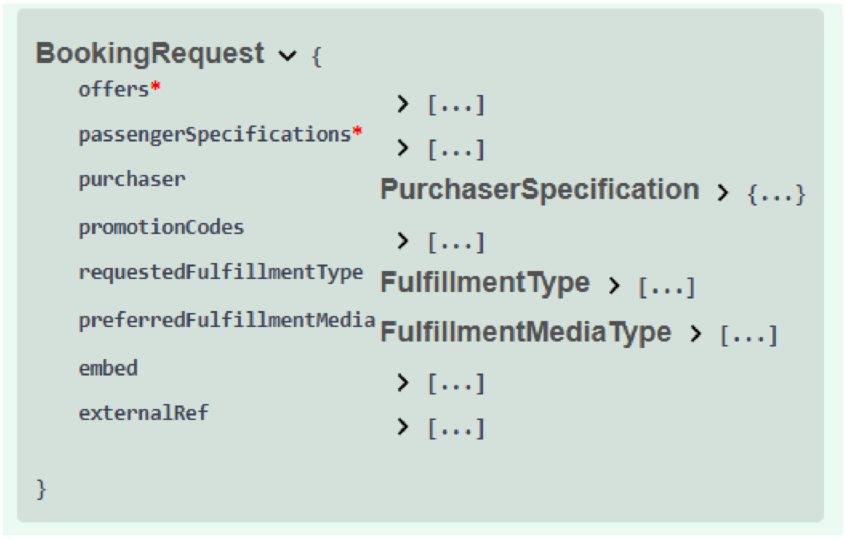
A booking is a collection of booking parts created from offer parts. The booking parts are stateful:


BookingPart State Model
POST /bookings with offer ids, optional accommodation / ancillariy ids
prebooked
PATCH /bookings/{id}/onhold-offer
onhold
ttl reached
DELETE /bookings/{id}
POST /bookings/{id}/fulfillments
where at least 1 fulfillment is confirmed
POST /bookings/{id}/fulfillments
where all fulfillments are fulfilled
unrecoverable error
at confirmation/fulfillment
PATCH /bookings/{id}/fulfillments
extended ttl reached
onrecoverable error
confirmed
cancelled
PATCH /bookings/{id}/fulfillments
all fulfillments are now fulfilled
unrecoverable error at fulfillment
of a confirmed booking
fulfilled
error
POST /bookings/{id}/releaseOffers
followed by confirm
exchange is not finalized
POST /exchangeOffers
followed by confirm
POST /bookings/{id}/refundOffers
followed by confirm
released
exchangeOngoing
If refund operation is triggered
archiving
POST /bookings/{id}/exchangeOperations
followed by POST /booking/{id}/fulfillments
archiving
DELETE /booking
in a cleanup context
refunded
exchanged

A booking is created by:

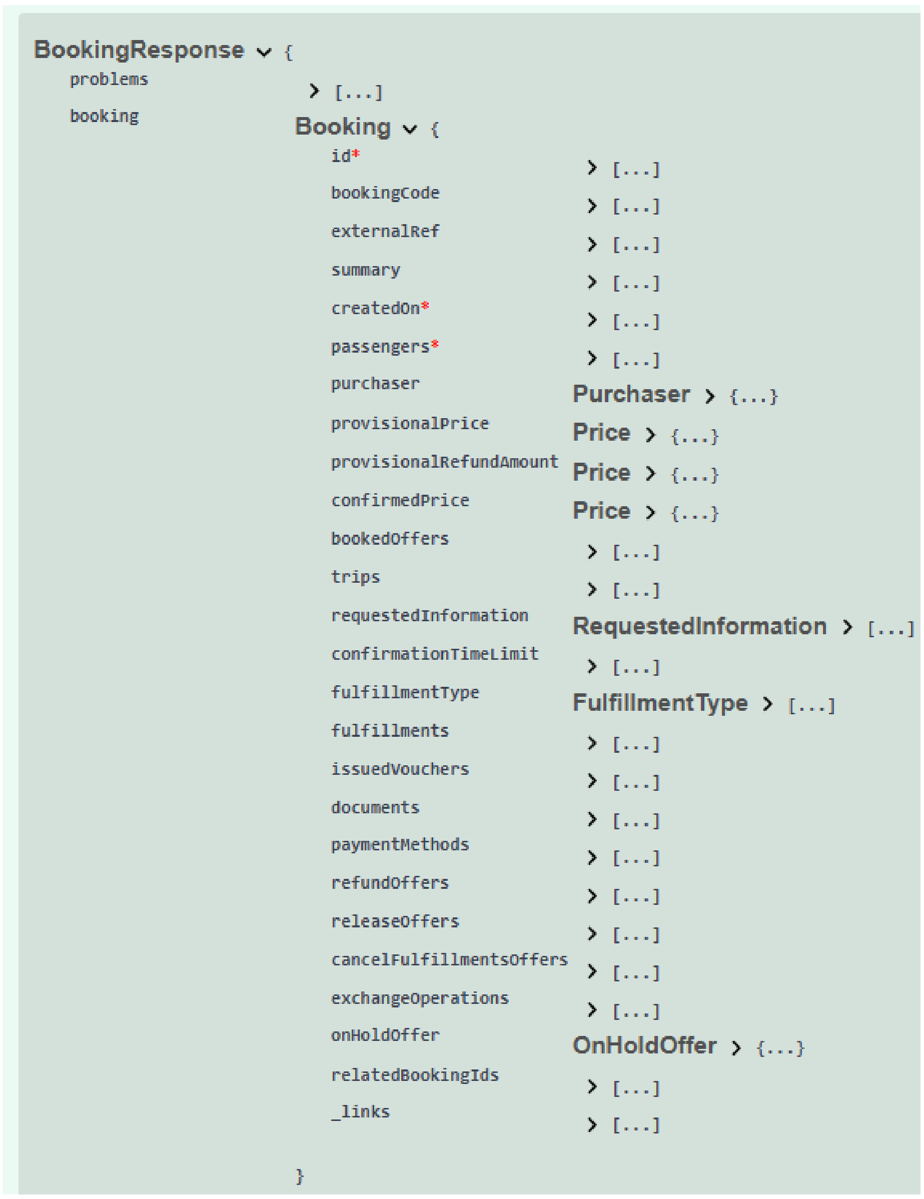
POST /Bookings

The request must reference the offers to be booked and passenger data as requested by the offer.



```
BookingRequest ⌄ {
    offers*                       › […]
    passengerSpecifications*      › […]
    purchaser                     PurchaserSpecification › {…}
    promotionCodes                › […]
    requestedFulfillmentType      FulfillmentType › […]
    preferredFulfillmentMedia     FulfillmentMediaType › […]
    embed                         › […]
    externalRef                   › […]
}
```

The reply includes the booking in prebooked state. The booked offer parts are listed in bookedOffers. The booking includes the confirmationTimeLimit, until when it must be confirmed or will be deleted.



```
BookingResponse ∨ {
    problems          > [...]
    booking           Booking ∨ {
                          id*                           > [...]
                          bookingCode                   > [...]
                          externalRef                   > [...]
                          summary                       > [...]
                          createdOn*                    > [...]
                          passengers*                   > [...]
                          purchaser                     Purchaser > {...}
                          provisionalPrice              Price > {...}
                          provisionalRefundAmount       Price > {...}
                          confirmedPrice                Price > {...}
                          bookedOffers                  > [...]
                          trips                         > [...]
                          requestedInformation          RequestedInformation > [...]
                          confirmationTimeLimit         > [...]
                          fulfillmentType               FulfillmentType > [...]
                          fulfillments                  > [...]
                          issuedVouchers                > [...]
                          documents                     > [...]
                          paymentMethods                > [...]
                          refundOffers                  > [...]
                          releaseOffers                 > [...]
                          cancelFulfillmentsOffers      > [...]
                          exchangeOperations            > [...]
                          onHoldOffer                   OnHoldOffer > {...}
                          relatedBookingIds             > [...]
                          _links                        > [...]

                      }
```

On the booking resource the following actions are possible:

GET /bookings/{bookingId}

To retrieve the booking data in case the client has lost his booking data.

PATCH /bookings/{bookingId}

To add details on the selection of seats, possible fulfilment media and payment method.

DELETE /bookings/{bookingId}

To delete a booking without confirmed booking parts. Although the booking will be deleted after the timeout it is recommended to delete it in case it is not anymore needed.

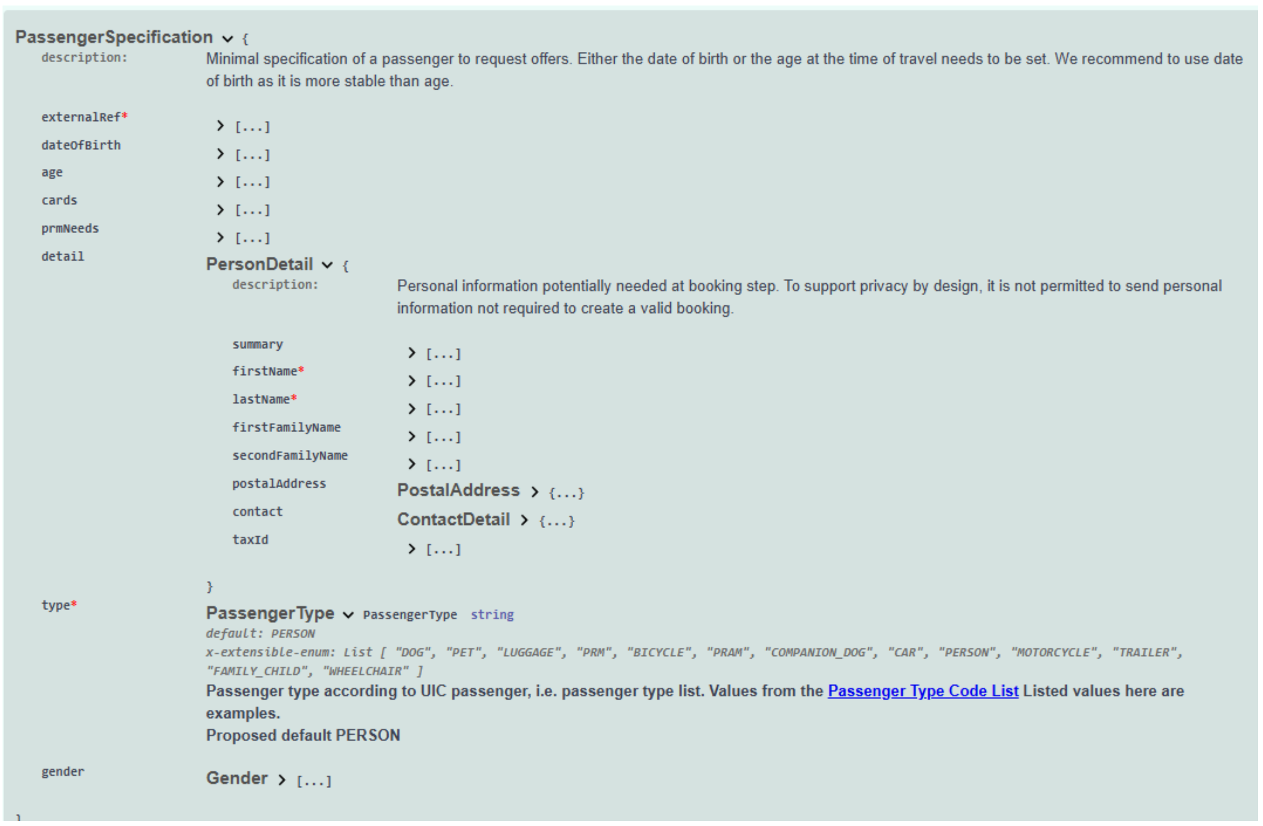
Other services are available directly on parts of a booking.

7.3. /bookings/{bookingId}/passengers

Changing passenger data per passenger:

PATCH `/bookings/{bookingId}/passengers/{passengerId}`

The PATCH provides a passenger specification:



```
PassengerSpecification ⌄ {
   description:        Minimal specification of a passenger to request offers. Either the date of birth or the age at the time of travel needs to be set. We recommend to use date
                       of birth as it is more stable than age.

   externalRef*        > [...]
   dateOfBirth         > [...]
   age                 > [...]
   cards               > [...]
   prmNeeds            > [...]
   detail              PersonDetail ⌄ {
                          description:         Personal information potentially needed at booking step. To support privacy by design, it is not permitted to send personal
                                               information not required to create a valid booking.

                          summary              > [...]
                          firstName*           > [...]
                          lastName*            > [...]
                          firstFamilyName      > [...]
                          secondFamilyName     > [...]
                          postalAddress        PostalAddress > {...}
                          contact              ContactDetail > {...}
                          taxId                > [...]

                       }
   type*               PassengerType ⌄ PassengerType  string
                       default: PERSON
                       x-extensible-enum: List [ "DOG", "PET", "LUGGAGE", "PRM", "BICYCLE", "PRAM", "COMPANION_DOG", "CAR", "PERSON", "MOTORCYCLE", "TRAILER",
                       "FAMILY_CHILD", "WHEELCHAIR" ]
                       Passenger type according to UIC passenger, i.e. passenger type list. Values from the Passenger Type Code List Listed values here are
                       examples.
                       Proposed default PERSON

   gender              Gender > [...]
}
```

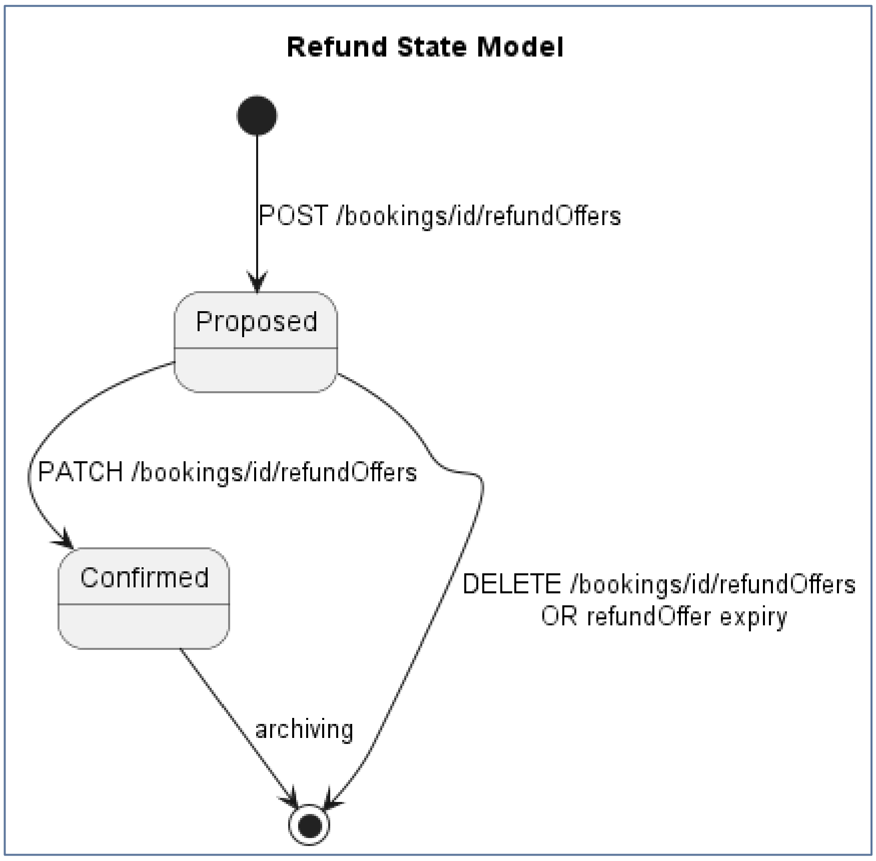
Changes on the passenger data that affect the validity of the booked offer might be rejected by the provider. In that case a new offer needs to be requested.

Viewing passenger data per passenger:

GET `/bookings/{bookingId}/passengers/{passengerId}`

7.4. /bookings/{bookingId}/refund-offers

Cancellation and partial cancellation of a booking are managed via refund offers. The refund is stateful:

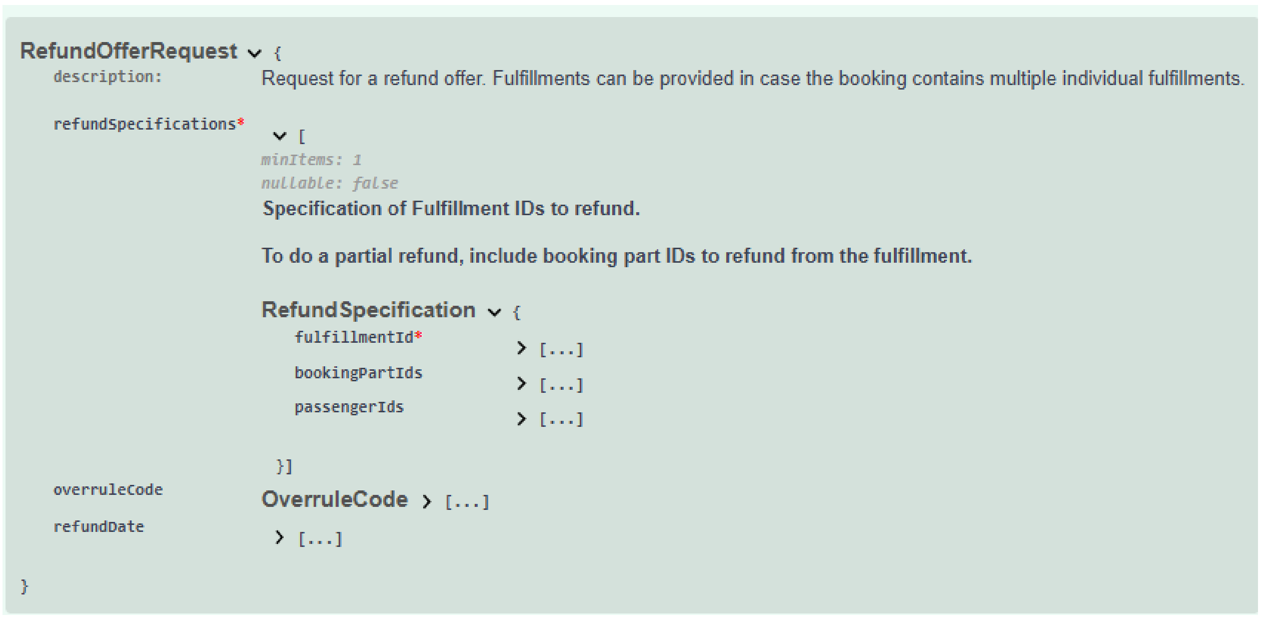

Refund State Model
POST /bookings/id/refundOffers
Proposed
PATCH /bookings/id/refundOffers
Confirmed
DELETE /bookings/id/refundOffers
OR refundOffer expiry
archiving

Refund offers are created via:

POST `/bookings/{bookingId}/refund-offers`

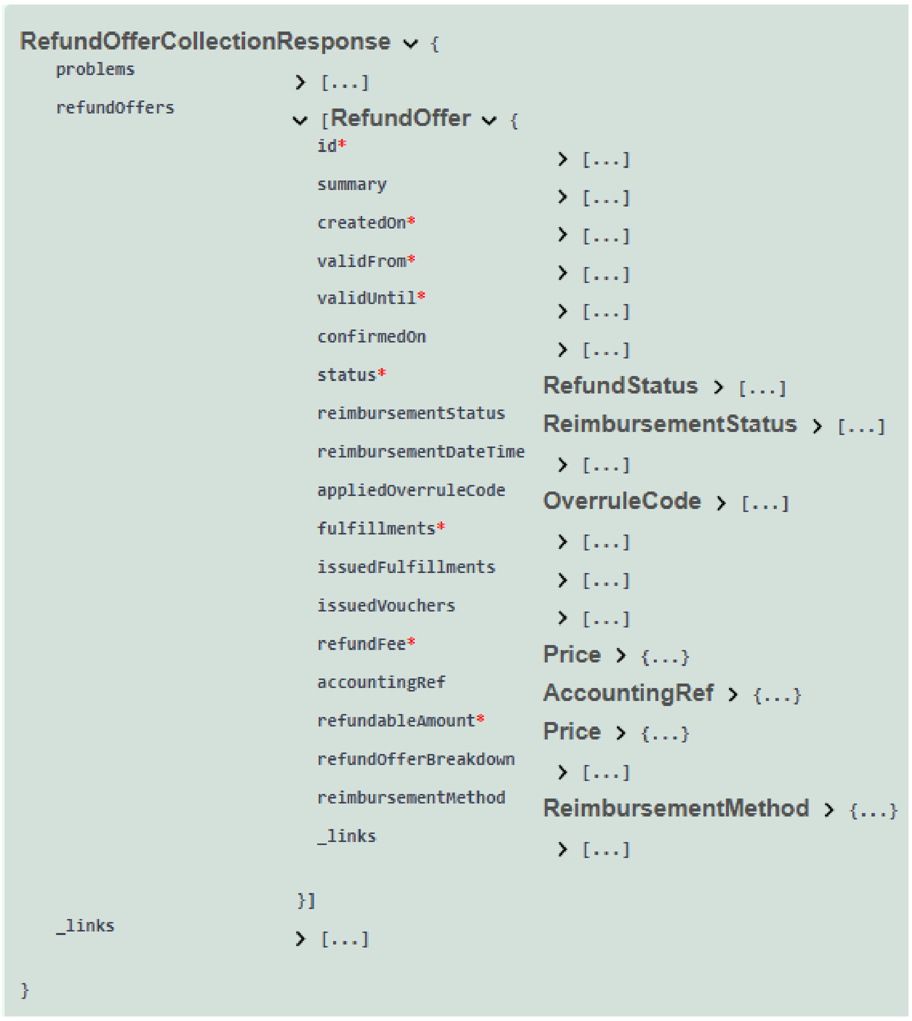
The request specifies the fulfilments to be refunded and/or the passengers to be removed. Overrule codes indicate special cases where the refund rules of the booked offer should not apply (e.g. mistakes by sales staff,..).

In case of collective ticketing of collective tariff a partial refund might be rejected as an exchange of the existing fulfilments needs to be made.



```
RefundOfferRequest ⌄ {
    description:          Request for a refund offer. Fulfillments can be provided in case the booking contains multiple individual fulfillments.

    refundSpecifications*
                           ⌄ [
                          minItems: 1
                          nullable: false
                          Specification of Fulfillment IDs to refund.

                          To do a partial refund, include booking part IDs to refund from the fulfillment.

                          RefundSpecification ⌄ {
                              fulfillmentId*        › [...]
                              bookingPartIds        › [...]
                              passengerIds          › [...]

                           }]
    overruleCode          OverruleCode › [...]
    refundDate
                           › [...]
}
```

The reply provides the refund-offers:


```
RefundOfferCollectionResponse ⌄ {
   problems                      > [...]
   refundOffers                  ⌄ [RefundOffer ⌄ {
                                    id*                          > [...]
                                    summary                      > [...]
                                    createdOn*                   > [...]
                                    validFrom*                   > [...]
                                    validUntil*                  > [...]
                                    confirmedOn                  > [...]
                                    status*                      RefundStatus > [...]
                                    reimbursementStatus          ReimbursementStatus > [...]
                                    reimbursementDateTime        > [...]
                                    appliedOverruleCode          OverruleCode > [...]
                                    fulfillments*                > [...]
                                    issuedFulfillments           > [...]
                                    issuedVouchers               > [...]
                                    refundFee*                   Price > {...}
                                    accountingRef                AccountingRef > {...}
                                    refundableAmount*            Price > {...}
                                    refundOfferBreakdown         > [...]
                                    reimbursementMethod          ReimbursementMethod > {...}
                                    _links                       > [...]

                                 }]
   _links                        > [...]

}
```

A refund offer is accepted via a PATCH on the refund offers.

PATCH /bookings/{bookingId}/refund-offers/{refundOfferId}

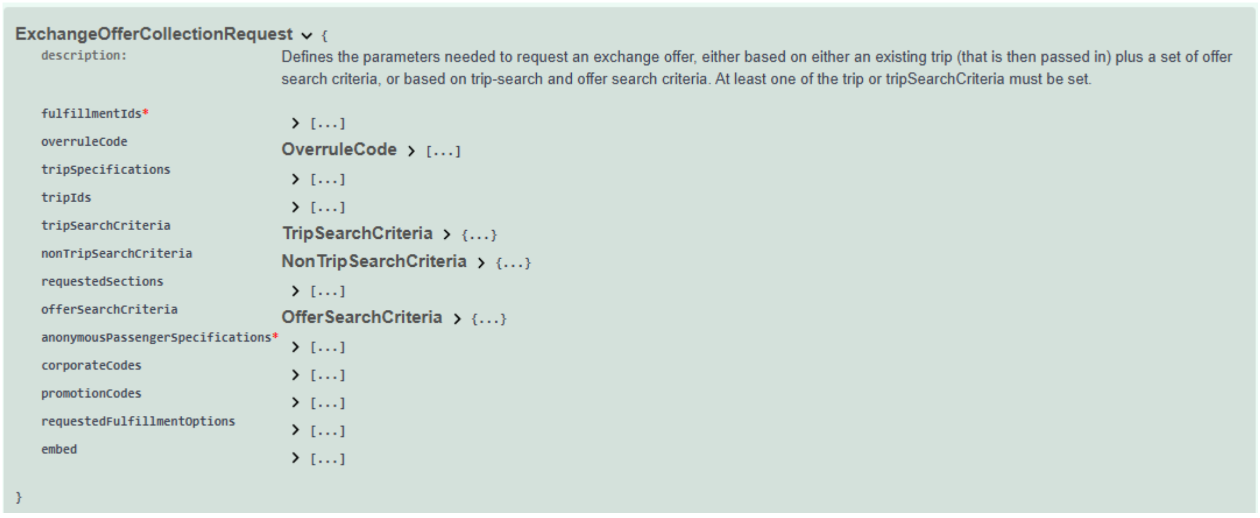
With the confirmation of the refund offer alternative unconfirmed refund offer will be removed from the booking.

7.5. /bookings/{bookingId}/exchange-offers

Exchanges are managed via exchange offers. An exchange offer is created by:

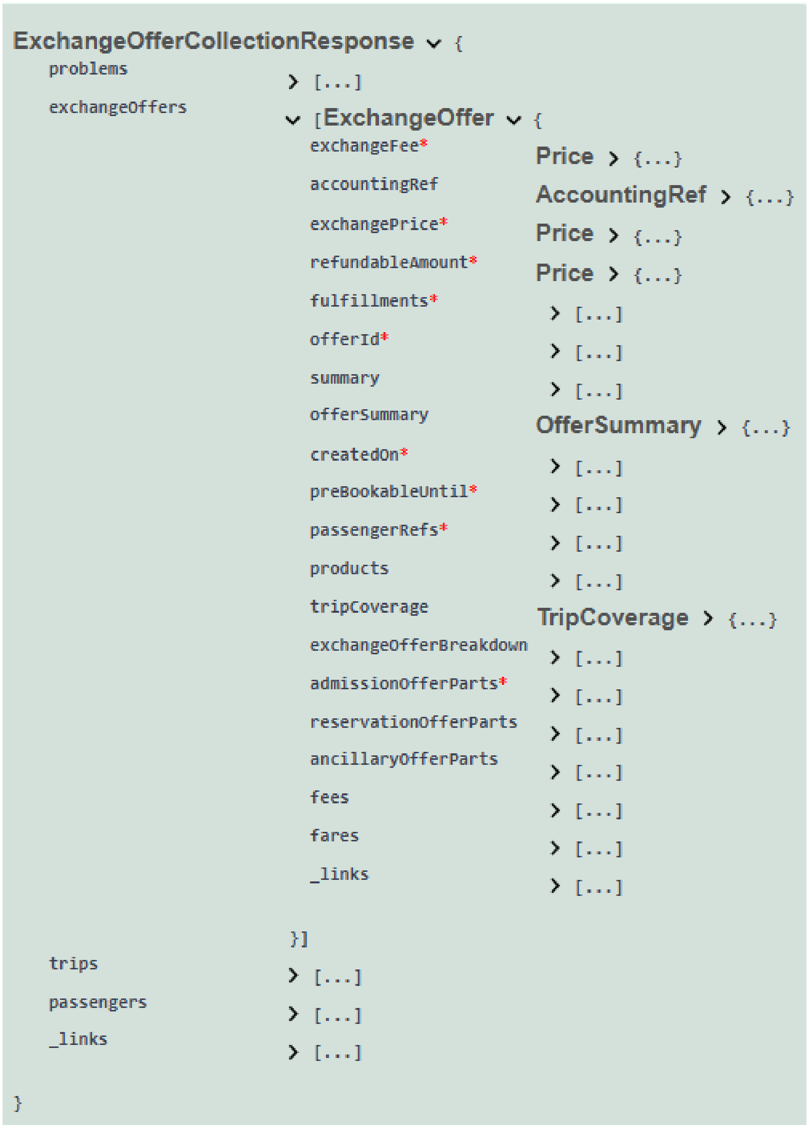
POST `/bookings/{bookingId}/exchange-offers`

The remaining or extended passengers must be provided. A new trip can be provided in case the trip should be changed.



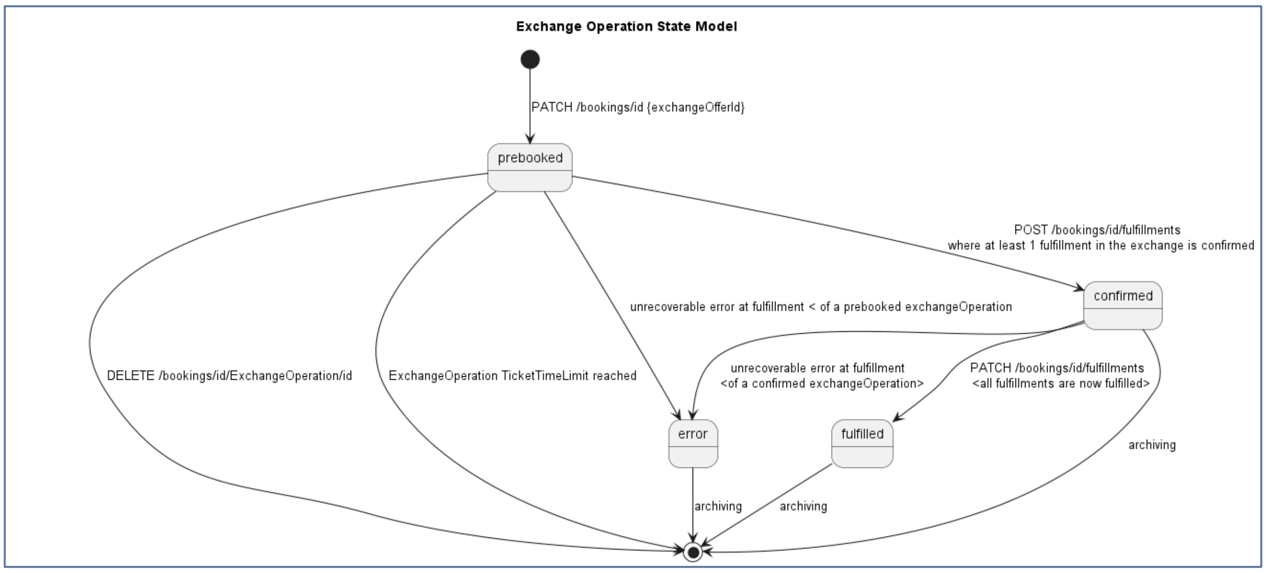
```
ExchangeOfferCollectionRequest {
    description:                Defines the parameters needed to request an exchange offer, either based on either an existing trip (that is then passed in) plus a set of offer
                                search criteria, or based on trip-search and offer search criteria. At least one of the trip or tripSearchCriteria must be set.

    fulfillmentIds*              > [...]
    overruleCode                OverruleCode > [...]
    tripSpecifications           > [...]
    tripIds                      > [...]
    tripSearchCriteria          TripSearchCriteria > {...}
    nonTripSearchCriteria       NonTripSearchCriteria > {...}
    requestedSections            > [...]
    offerSearchCriteria         OfferSearchCriteria > {...}
    anonymousPassengerSpecifications* > [...]
    corporateCodes               > [...]
    promotionCodes               > [...]
    requestedFulfillmentOptions  > [...]
    embed                        > [...]
}
```

The resulting exchange offers provide the new offers and the exchange fees:


```
ExchangeOfferCollectionResponse ⌄ {
   problems                   › [...]
   exchangeOffers             ⌄ [ExchangeOffer ⌄ {
                                   exchangeFee*            Price › {...}
                                   accountingRef           AccountingRef › {...}
                                   exchangePrice*          Price › {...}
                                   refundableAmount*       Price › {...}
                                   fulfillments*           › [...]
                                   offerId*                › [...]
                                   summary                 › [...]
                                   offerSummary            OfferSummary › {...}
                                   createdOn*              › [...]
                                   preBookableUntil*       › [...]
                                   passengerRefs*          › [...]
                                   products                › [...]
                                   tripCoverage            TripCoverage › {...}
                                   exchangeOfferBreakdown  › [...]
                                   admissionOfferParts*    › [...]
                                   reservationOfferParts   › [...]
                                   ancillaryOfferParts     › [...]
                                   fees                    › [...]
                                   fares                   › [...]
                                   _links                  › [...]

                                }]
   trips                      › [...]
   passengers                 › [...]
   _links                     › [...]

}
```

7.6. /bookings/{bookingId}/exchange-operations

The exchange offer needs to be selected. As new booking parts will be created this cannot be made in a single service. The exchange operation is stateful:


Exchange Operation State Model
PATCH /bookings/id {exchangeOfferId}
prebooked
POST /bookings/id/fulfillments
where at least 1 fulfillment in the exchange is confirmed
confirmed
unrecoverable error at fulfillment < of a prebooked exchangeOperation
DELETE /bookings/id/ExchangeOperation/id
ExchangeOperation TicketTimeLimit reached
unrecoverable error at fulfillment
<of a confirmed exchangeOperation>
PATCH /bookings/id/fulfillments
<all fulfillments are now fulfilled>
error
fulfilled
archiving
archiving
archiving

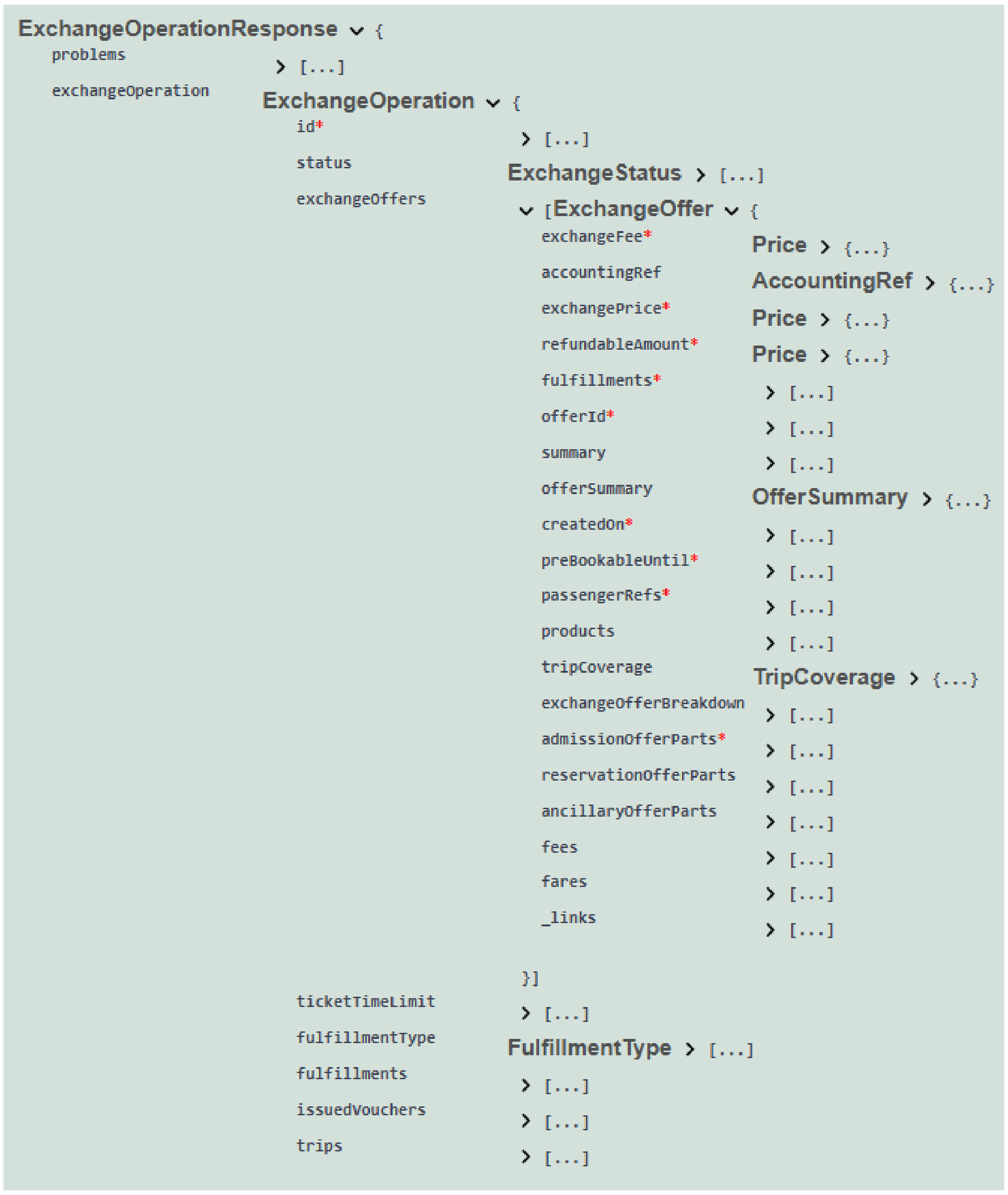
An exchange operation is created for an exchange offer to manage the remaining process steps:

Create an exchange operation:

POST `/bookings/{bookingId}/exchange-operations`

The booking parts included in the exchange operation are in prebooked state.

The already described services on bookings and sub-resources of bookings are used to change data on the new booking parts.



```
ExchangeOperationResponse ⌄ {
    problems                    > [...]
    exchangeOperation           ExchangeOperation ⌄ {
                                    id*                 > [...]
                                    status              ExchangeStatus > [...]
                                    exchangeOffers      ⌄ [ExchangeOffer ⌄ {
                                                            exchangeFee*               Price > {...}
                                                            accountingRef              AccountingRef > {...}
                                                            exchangePrice*             Price > {...}
                                                            refundableAmount*          Price > {...}
                                                            fulfillments*              > [...]
                                                            offerId*                   > [...]
                                                            summary                    > [...]
                                                            offerSummary               OfferSummary > {...}
                                                            createdOn*                 > [...]
                                                            preBookableUntil*          > [...]
                                                            passengerRefs*             > [...]
                                                            products                   > [...]
                                                            tripCoverage               TripCoverage > {...}
                                                            exchangeOfferBreakdown     > [...]
                                                            admissionOfferParts*       > [...]
                                                            reservationOfferParts      > [...]
                                                            ancillaryOfferParts        > [...]
                                                            fees                       > [...]
                                                            fares                      > [...]
                                                            _links                     > [...]

                                                        }]
                                    ticketTimeLimit     > [...]
                                    fulfillmentType     FulfillmentType > [...]
                                    fulfillments        > [...]
                                    issuedVouchers      > [...]
                                    trips               > [...]
```

The exchange-operation is confirmed by confirming by confirming the booking.

An unconfirmed exchange operation can be deleted. This is needed as there can not be multiple exchange operations in progress at the same time. So is new exchange offers should be made the current exchange operations need to be deleted.

`DELETE /bookings/{bookingId}/exchange-operations/{exchangeOperationId}`

7.7. /availabilities

Availabilities provide Information that are part of a booking. They are mainly used to retrieve information on available seats to feed the seat selection in the booking.

The requests always need to specify the related booking part or offer part from the booking to restrict the reply to the options compatible with the offer part or booking part.

POST `/availabilities/place-map`

GET `/availabilities/nearby`

GET `/availabilities/preferences`

8. Pagination

OSDM uses cursor based pagination and the `_links` concept of **HATEOAS** for pagination. Thus, for responses where pagination is relevant, pagination links will be added, following URL to `next` and `previous` pages are provided if pagination shall be supported.

```
{
  "id": "trip-1",
  "trips": [],
  "_links": [
    {
      "rel": "self",
      "href": "https://api.osdm.com/bookings/123124"
    },
    {
      "rel": "next",
      "href": "https://api.osdm.com/bookings/123124?page=next"
    },
    {
      "rel": "previous",
      "href": "https://api.osdm.com/bookings/123124?page=previous"
    }
    ...
  ]
}
```

The nature of the link is indicated by the `rel` attribute.

Where semantically valid, additional links to the `first`and `last` pages can be provided, i.e., the`/bookings` resource.

8.1. Resources Supporting Pagination

With this version of the specification, the following resources should support pagination:

- **GET** `/exchange-offers/`
- **GET** `/bookings/`
- **GET** `/coachLayouts/`

Note that while a `page` query parameter is provided for verbs supporting pagination, it is not mandatory to use it: One implementor might use the parameter to scroll with a fixed collectionId, while another could prefer consider scrolling the retrieval of previous and next collections, thus ignoring the parameter and linking to collections with a different id.

9. Error Handling

9.1. General HTTP error codes and generic situations

The following standard HTTP error codes are used in the specification:

Error Code	Description
`400`	Bad request
`401`	Unauthorized
`403`	Forbidden
`404`	Not found
`405`	Method not allowed
`409`	Conflict
`429`	Too many requests
`500`	Internal server error
`501`	Not implemented
`503`	Service unavailable

9.2. Functional Errors

The OSDM API makes use of the JSON Problem structure to return information about functional errors in the handling of a request. The problem structure is defined in RfC-9457 which defines 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.

For OSDM, the following properties are supported:

Property	Description	Mandatory
`code`	The `code` property should begin with the code of the error type	yes
`title`	Short, human-readable summary of the problem type which **should not** change from occurrence to occurrence of the problem.	yes
`type`	An absolute URI that identifies the problem type. When dereferenced, it SHOULD provide human-readable documentation for the problem type (e.g., using HTML).	yes
`detail`	Human-readable explanation specific to this occurrence of the problem	no
`status`	The HTTP status code generated by the origin server for this occurrence of the problem.	no
		no

A problem message is typically made of a code, title and a detail. The title and detail are not translated and the expected language there should be in English. It is up to the implementers to foresee a translation based on the code if relevant. Optionally, a problem message can contain reference to resources in the model.

The list of all error codes that can be returned by the API, needs to exposed on an URI:

`https://<host>/errors/<error-code>`

where host denotes the host of the API.

9.3. Examples

No result found for given request

```
"problem": {
  "code": "urn:uic:problem:NO_RESULTS",
  "type": "https://osdm.io/errors/no-results",
  "title": "The search did not return any result",
  "detail": "The place `Duckburg` could not be found",
  "status": 404,
}
```

Request not conform OpenAPI schema

```
"problem": {
  "code": "urn:uic:problem:MALFORMED_REQUEST",
  "type": "https://osdm.io/errors/malformed_request",
  "title": "Request failed schema validation",
  "detail": "The request does not validate the OSDM OpenAPI schema
specification",
  "status": 400,
}
```

Request with invalid input

```
"problem": {
  "code": "urn:uic:problem:VALIDATION_ERROR",
  "type":
"https://provider.domain/osdm/documentation/errors/validation_error",
  "title": "Passenger details missing",
  "detail": "The passenger does not contain all mandatory information to
confirm the booking",
  "status": 400,
}
```

9.4. Standardized Technical Errors

In order that OSDM implementations behave consistently in error situations, the following error codes must be supported in case of technical errors by all implementations:

Problem code	Description
`RESOURCE_NOT_FOUND`	The requested (sub) resource could not be found. Could be deleted or expired
`OPERATION_NOT_PERMITTED`	Trying to perform an operation that is not permitted.
`NO_RESULTS`	The search did not return any result
`VALIDATION_ERROR`	The request contains incorrect information
`MALFORMED_REQUEST`	The request does not match the OSDM specification. Possible version mismatch
`MISSING_INFORMATION`	Missing information. Provide the mandatory information and try again

Problem code	Description
`PARAMETER_NOT_SUPPORTED`	A given request parameter is not supported and ignored while handling the request
`INVALID_INPUT`	Provided input is invalid.
`UNKNOWN_ERROR`	Unexpected or unspecified error occurred
`PROPERTY_SUBSTITUTED`	Requested property is not available and is substituted. Check the response for the substitute
`PARTIAL_SUCCESS`	The request could not be fully processed and is partially processed
`SERVICE_UNAVAILABLE`	The service is currently not available
`UNAUTHORIZED`	Client is no authorized

9.5. Standardized Functional Errors

In order that OSDM implementations behave consistently in error situations, the following error codes must be supported in case of functional errors by all implementations:

Functional area	Error Code	Title
Places	`PLACE_INVALID_CHARACTERS`	Invalid characters in the search string
Places	`PLACE_NO_RESULTS`	The search did not return any result
Trips	`TRIP_INVALID_CHARACTERS`	A search criteria value contains invalid value or invalid characters
Trips	`TRIP_SEARCH_CRITERIA_OUTSIDE_BOUNDARY`	A search criteria lies outside accepted boundaries
Trips	`TRIP_PLACE_UNKNOWN`	A provided place is not known

Functional area	Error Code	Title
Trips	`TRIP_NO_SEARCH_RESULT`	The search did not return any result
Offers	`OFFER_TRIP_NOT_FOUND`	The referenced trip cannot be found (expired ?)
Offers	`OFFER_INVALID_CHARACTERS`	A search criteria value contains invalid value or invalid characters
Offers	`OFFER_SEARCH_CRITERIA_OUT_OF_BOUNDS`	A search criteria lies outside accepted boundaries
Offers	`OFFER_PLACE_UNKNOWN`	A provided place is not known
Offers	`OFFER_SCHEDULE_MISMATCH`	Schedule mismatch between systems
Offers	`BOOKING_RESERVATION_OPTION_NOT_AVAILABLE`	The requested reservation option is not available on this vehicle
Offers	`BOOKING_PASSENGER_PROPERTY_NOT_MODIFIABLE`	Attempted to modify a read-only property (passenger)
Offers	`BOOKING_OFFERPART_PROPERTY_NOT_MODIFIABLE`	Attempted to modify a read-only property (reservation, ancillary or fare)
Booking	`BOOKING_OFFER_NOT_FOUND`	Referenced Offer or offer part not found (offer expired ?)
Booking	`BOOKING_INCOMPATIBLE_OFFER_PART`	Incompatible offer part with the offer
Booking	`BOOKING_INFORMATION_MISSING`	Missing information

Functional area	Error Code	Title
Booking	`BOOKING_INSUFFICIENT_AVAILABILITY`	Insufficient availability for one of the requested products
Booking	`BOOKING_PLACE_NOT_AVAILABLE`	The requested place is not available
Booking	`BOOKING_MODIFY_READ_ONLY_PROPERTY`	Attempted to modify a read-only property
Booking	`BOOKING_BOOKING_ALREADY_CONFIRMED`	The booking is already confirmed
Booking	`BOOKING_BOOKING_ALREADY_CANCELLED`	The booking is already cancelled
Booking	`BOOKING_MODIFICATION_NOT_ALLOWED`	The booking and does not allow modifications
Booking	`BOOKING_VEHICLE_TOO_HEAVY`	A vehicle is too heavy to be transported by car carriage. This relates to the current load of the train or coach, so booking might be possible at another
Booking	`BOOKING_VEHICLE_WEIGHT_MISSING`	A vehicle is not possible without providing the weight of the vehicle
Confirm	`CONFIRMATION_PARTIAL_SUCCESS`	Partial success
Confirm	`CONFIRMATION_OPERATION_NOT_SUPPORTED`	Operation not supported on one of the offer parts
Confirm	`CONFIRMATION_UNKNOWN_ERROR`	Unknown error on provider side
Confirm	`CONFIRMATION_INFORMATION_MISSING`	Missing information in the booking

Functional area	Error Code	Title
Confirm	`CONFIRMATION_FULFILLMENT_TYPE_NOT_SELECTED`	Fulfillment type not selected
Confirm	`CONFIRMATION_BOOKING_ALREADY_CONFIRMED`	Booking already confirmed
Confirm	`CONFIRMATION_BOOKING_ALREADY_FULFILLED`	Booking already fulfilled
Confirm	`CONFIRMATION_BOOKING_ALREADY_CANCELLED`	Booking already cancelled

The lists can be extended by an implementor but at least these errors must be captured and they must be presented with the codes listed here above.

In case they wish to pass additional problems specific to their situation and not covered by any of the case below, they can do so by replacing the OSDM namespace with custom namespace starting with an X_ followed by an unique identifier for the provider (ex: X_NVS_NOMEAL).

9.6. Dealing with Unsupported Parameters in Requests

Some OSDM requests potentially support a large number of parameters (e.g. filters). Supporting all of them is not always possible, e.g. due restrictions of the underlying systems. Thus the following rule applies:

- If the parameter is required, return an *error* “PARAMETER_NOT_SUPPORTED”.
- If the parameter is optional, return a *response* including a *warning* indicating that the optional request parameter was ignored.