

# Electronic reservation of seats/berths and electronic production of travel documents - exchange of messages - Annex B2 - OMSA

*Telematics TSI - Technical Document - B5 - Annex B2 - OMSA*  
*Version 4.0*

## *Contents*

A.	Document management .....	4
A.1	Document properties.....	4
A.2	Change management.....	4
A.3	Configuration management.....	4
A.4	Availability.....	4
A.5	Application and actors in the scope.....	4
A.6	Document history .....	4
B.	Acronyms, definitions and external references.....	6
B.1	Acronyms .....	6
B.2	Definitions.....	6
B.3	External references .....	6
1.	Foreword.....	7
2.	Objectives .....	7
3.	Summary .....	7
4.	Terms and Definitions.....	8
5.	Roles & architecture .....	8
6.	Design principles .....	9
7.	Mapping of APIs (Appendix A to Appendix B).....	11
8.	Use Cases covered .....	14
8.1.	Search offers for a trip pattern .....	14
8.2.	Selecting offers for purchase .....	15
8.3.	Purchase package .....	15
8.3.1.	Direct.....	15
8.3.2.	2-phase.....	15
8.4.	Add ancillaries to a package.....	16
8.5.	Select and change an asset (e.g. seat) .....	16
8.6.	Cancel a package.....	17
8.7.	Refund complete purchase .....	17
8.8.	Add or change personal data .....	17
9.	Advanced use cases .....	18
9.1.	Partial cancellation by reducing the number of passengers.....	18
9.2.	Refund ancillary .....	18
9.3.	Exchange offers in a purchased package .....	19
10.	Additional use cases.....	19
10.1.	Passengers with reduced mobility .....	19
10.2.	Passengers travelling with bicycle .....	20
10.3.	Passengers bringing a car.....	20
10.4.	Redresses .....	21
11.	Technical aspects .....	22

11.1.	Self-explaining.....	22
11.2.	Authentication .....	22
11.2.1.	Authentication flows.....	22
11.2.2.	JWT structure.....	23
11.2.3.	Authentication endpoint.....	24
11.3.	Authorisation .....	24
11.3.1.	Open endpoints .....	24
11.3.2.	24	
11.3.3.	Secured endpoints .....	25
11.4.	Pagination .....	25
12.	Appendix A: OGC.....	26
12.1.	OGC API Records & Features .....	26
12.2.	OGC API Process.....	26
13.	Appendix B: States .....	28
13.1.	Offer module.....	28
13.2.	Pre-sales module.....	28
13.3.	Purchase module .....	28
13.4.	After-sales module.....	29
14.	Appendix C: Endpoints.....	30
15.	Offer module.....	30
15.1.	Search offers .....	30
15.2.	Get data sources .....	33
15.3.	Pre-sales module.....	34
15.4.	Select offer(s).....	34
15.5.	Update traveller .....	35
15.6.	Remove traveller.....	36
15.7.	Find assets.....	37
15.8.	Assign asset (e.g. a seat) .....	39
15.9.	Find ancillaries .....	39
15.10.	Assign ancillary.....	41
16.	Purchase module .....	42
16.1.	Purchase package, 2-phase purchase package, confirm package, release package .....	42
16.2.	Get travel documents .....	43
17.	After-Sales module.....	45
17.1.	Find refund options for a package .....	45
17.2.	Claim a refund option .....	47
17.3.	Confirm a refund option .....	47
17.4.	Refund without options (in case of emergency).....	48
18.	Copyrights .....	49

## A. Document management

### A.1 Document properties

- File name: ERA\_TD\_B5\_B2.docx
- Subject and document type: Telematics TSI - Technical Document - B5 - Annex B2 - OMSA
- 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.

Disclaimer:

Specific legal references to technical documents and legal acts shall be revised after the enter into force of the Telematics TSI. In some sections this text can be highlighted.

### 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

--

### B.2 Definitions

Terms contained in this document are defined in the ERA Ontology. A specific set of terms is defined also in chapter 4.

### 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]				

## 1. Foreword

The aim of this document is to map the XML/SOAP API requests provided in the TAP-TSI B.5 Annex B (II) to the JSON-requests provided in the Open Mobility Sales API (OMSA).

The direct mapping between the previous XML/SOAP requests and the proposed JSON-requests addressing the required use cases presented can be found in the TAP-TSI B.5 Frame document.

## 2. Objectives

The primary objective of this document is to replace the existing XML/SOAP API (see Appendix A) with a new JSON/REST API that complies with Transmodel and the ongoing work in CoRoM.

## 3. Summary

The sales API includes the process of reserving and modifying PACKAGE on a TRIP and performing after sales on the purchased PACKAGE.

The sales flow is as follows

- Search for offers
- Select an offer or offers, resulting in a package
  - Look for and add Ancillaries to package
  - Look for and assign Assets to package (this includes seats)
- Purchase & confirm the package
- Collect Travel Documents

The after sales flow is as follows

- Calculate refund options
- Claim refund option
- Confirm refund option

The API is based on 2-phase commit to make it easier for an aggregator to keep the state consistent.

## 4. Terms and Definitions

The **API**: the Transmodel based REST API this document describes.

**Booking**: Due to the ambiguity we avoid the term booking. A booking is realized through creating and purchasing a package.

**Package**: The basket containing the selected / purchased offers for a given *Trip Pattern*.

**Leg**: in the context of B5, a *service journey* on a particular *date*, with a *from* and *to* stop place.

**Trip Pattern**: the movement of a passenger(s) (or another person, e.g. the driver) from an origin to a destination consisting of a list of consecutive legs.

**Client**: the software that uses the API. It could belong to an aggregator that combines the offerings of several railway undertakings, or a railway undertaking itself.

**Traveller**: An individual or group using the transport services.

**Product**: References the *Fare Product* for the leg(s) and contains a short description of the conditions provided.

**Customer**: The person or organization that is creating and purchasing a package.

**Offer**: A collection of products responding to the request of a customer on a set of legs containing prices and conditions.

**Guarantee**: The promises that a certain service will be provided to a certain quality level.

**Redress**: If a *Guarantee* is not met, then the *Traveller* can claim compensation in the form of a redress.

**Asset**: A physical object that the *Traveller* uses to consume their access right on a given *Leg*.

**Travel Document**: A particular support (ticket, card, etc.) to be held by a customer, allowing the right to travel or to consume joint services, to proof a payment (including possible discount rights), to store a subset of the fare contract liabilities or a combination of those.

**Ancillary**: Secondary *Products* that require an already chosen *Product* to be added to the *Package*. Can be fare related products, or simply additional services to be provided at a charge.

**Refund Option**: Describes the calculated offer for a refund operation on a given package. Covers refunds, cancellations and refunds performed with an overriding reason.

**Exchange**: When the traveller wants to change the time, journey or quality of the confirmed package.

## 5. Roles & architecture

### Customer

See explanation under Terms and Definitions.

### Traveller

See explanation under Terms and Definitions.

### Reseller

books on behalf of the customer and offers the end user experience to the customer. Also known as 'Retailer' (TSI context).

### Mobility Service provider

who provides the mobility services on behalf of the Mobility providers. Could also be known as a distributor. This entity is able to (combine) rail products into offers. In this TSI context, it must be in a fair and non-discriminatory manner independently of the railway undertakings involved.



**Mobility Providers**

organization who provides a flexible offering of transportation services.

Examples: scooters, bikes or car sharing.

**Transport provider**

organization who provides a regular offering of transportation services.

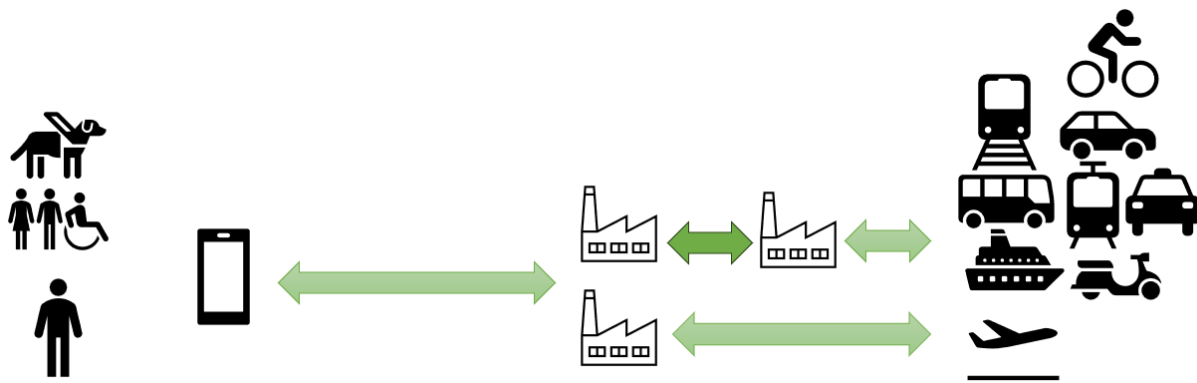
Examples : rail, bus or tram.

**Identity Provider (IdP)**

verifies identity and provides authentication of customers for the reseller

**Payment Service Provider (PSP)**

handles payment processing between customer and reseller



Applying the same specification in every layer between reseller and providers (mobility service, mobility and transport. Resellers are responsible for payment using their own PSP and ensuring their identity using an IdP. Synchronous requests between systems are made for exchanging data. No central routing registry has been established yet, so the reseller would know which providers to request for every transport- and mobility provider available in the network. Every other system would also need to have their own routing for their supported providers.

**6. Design principles**

A short overview of design principles we took into account:

- *“Don’t break the bank”*
  - Short return of investment
  - The implementing party is in control, describing what it has implemented and how. This allows to extend the implementation over time.
- *“Proactive error prevention”*
  - Easy to understand and use, be intuitive
  - Provide meta data for all functions
  - Make clear what is possible at any time (RFC 5988)
  - Provide context per returned entity (e.g. ‘this is an offer’)
  - Don’t patch data, we use logical operations. Patch requires external parties to know internal business logic.
  - Meta data must be provided in human AND machine readable format
  - Operations always return the created/modified resource, not parts of it

- Be clear about technical aspects, like identification, authenticity and authorisation (certificates, tokens)
  - Be clear about versions (semantic versioning).
- *“Don’t redo what others have done already”* - Reuse data sets that are already available (externally, like in NAPs)
- *“Mobility for all”* - Provide an API that is usable for all modes (including shared mobility)
- *“Provide trust”* - Prescribe price transparency
- *“There is a lot of knowledge out there”* - Comply to existing technical (meta) standards, techniques and ontologies

- Transmodel/NeTEx
- OGC API standards
- RestFULL API
- OpenAPI 3
- (Geo)JSON
- OAuth2, JWT, PKI
- RFC's, for e.g. date formats, problem details, language and country specifications
- Verifiable credentials/presentations

When applying these principles, there are quite a few technical problems already solved. In the next chapter, technical aspects are enlisted, and quite a few are referring back to the list above.

## 7. 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 delivery, 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
The customer wants an overview of available offers or fares.	<code>AvailabilityRequest</code>	POST /processes/search-offers/execute { travelSpecification, travellers, entitlements, requirements }  * Note: the returned offers contain a price & products, to facilitate fare transparency
The customer wants to view available offers.	<code>BookingRequest</code> with <code>informationOnly=true</code>	POST /processes/search-offers/execute { travelSpecification, travellers, entitlements, requirements }
The customer wants to select an offer	<code>BookingRequest</code>	POST /processes/select-offers/execute { offerId(s) }

Use Case	XML/SOAP Implementation	JSON/REST Implementation
presented to them.		
The customer wants to select a seat on the asset.	<p><code>BookingRequest near-by</code></p> <p><code>BookingRequest /</code> <code>BookingRequest with</code> <code>informationOnly=true</code></p> <p><code>using preference</code> <code>parameter</code></p> <p><code>using specific seat</code> <code>numbers</code></p> <p><code>GET CoachLayouts</code></p>	<p>GET /collections/assets/items { packageId, offerId, legId }</p> <p>POST /processes/assign-asset/execute { packageId, offerId, legId, assetId, travellerId }</p>
The customer support wants to perform a complete refund with override	<p><code>CancelRequest</code></p> <p>with <code>reason code</code></p>	<p>POST /processes/refund-package/execute { packageId, reason }</p>
The customer wants to perform a refund.	<p><code>CancelRequest</code></p>	<p>GET /collections/refund-options/items { packageId, type='refund-package' }</p> <p>POST /processes/claim-refund-option/execute { packageId, refundOptionId }</p> <p>POST /processes/confirm-refund-option/execute { packageId, refundOptionId }</p>
The sales system wants to perform a	<p><code>synchro request</code></p>	<p>In pre-purchase phase, it MUST be: POST /processes/release-offer/execute { packageId }</p>

Use Case	XML/SOAP Implementation	JSON/REST Implementation
technical rollback		In the purchased phase, it MUST be : POST /processes/refund-package/execute { packageId, reason=TechnicalFailure }

## Advanced use cases:

Use Case	HOSA	OMSA
The customer wants to add personal data to a selected offer	<b>BookingRequest</b> including personal data	POST /processes/search-offers/execute  POST /processes/select-offers/execute  POST /processes/update-traveller/execute { packageId, travellerId, personal data }
The customer wants to reduce the number of travellers in the travel party	<b>PartialCancellation-Request</b>	GET /collections/refund-options/items { packageId, type='remove-traveller', travellerId }  POST /processes/claim-refund-option/execute { packageId, refundOptionId }  POST /processes/confirm-refund-option/execute { packageId, refundOptionId }
The customer wants to remove an ancillary	–	GET /collections/refund-options/items { packageId, type='remove-ancillary, ancillaryId }  POST /processes/claim-refund-option/execute { packageId, refundOptionId }  POST /processes/confirm-refund-option/execute { packageId, refundOptionId }
The customer wants to exchange	<b>BookingRequest</b>	POST /processes/search-offers/execute { include packageToExchange }

Use Case	HOSA	OMSA
their previously purchased offer for a new offer.	including an exchange reference to the original booking	POST /processes/select-offers/execute  POST /processes/purchase-package/execute  POST /processes/confirm-package/execute, packageToExchange is cancelled.
The customer wants to claim their travel document	<b>BookingReply</b>  including a download link (depending on the Hermes version in use)	GET /collections/travel-documents/items { packageId }

## 8. Use Cases covered

### 8.1. Search offers for a trip pattern

The offer is requested by the Distributor and the response is provided by the Attributor. The precondition is, that the retailer (on behalf of the customer) has access to a journey planner or national access points, so that they are able to determine which service journeys they want to travel by.

Once a set of service journeys, origin and destination stop places and departure times have been selected, they can be combined into a list of legs, called a trip pattern. The reseller can request offers for the trip pattern through The API.

In order to accommodate different needs like Person with Reduced Mobility (PRM), pets or luggage requirements, requirements can be specified in the offer request.

If a traveller has certain entitlements (like reduction cards or existing tickets) they can be supplied.

If the customer provides a list of fare products, the offer should contain the results for these fare offers.

The reseller receives a list of all offers for the specified legs, along with a list of applicable ancillaries.. The reseller is then responsible for presenting the offers to the customer.

There may be several offers for the same trip pattern, with for instance different service level, operator or flexibility. Each offer may cover one or more legs in the trip pattern, and the customer must decide which offers to select for purchase.

---

*POST /process/search-offers/execute { tripPattern, travellers }*

*POST /process/search-offers/execute { travelSpecification, travellers }*

---

## 8.2. Selecting offers for purchase

The selecting offer is requested by the Distributor and the response is provided by the Contributor.

We assume that the reseller has received a list of offers, and that the customer is ready to select one or more offers for purchase, with an option to modify the selected offers.

---

*POST /processes/select-offers/execute { offerId(s) }*

---

This will result in a *package*, containing the selected offers. The package has an expiry time, after which the package will be expired and claimed resources will be released.

If the customer decides not to continue with the purchase process, it can release the package:

---

*POST /processes/release-package/execute { packageId }*

---

If the customer (or reseller) needs more time to complete the purchase process, it can request additional time:

---

*POST /processes/extend-expiry-time/execute { packageId }*

---

## 8.3. Purchase package

The Purchase package is requested by the Distributor and the response is provided by the Contributor.

Having selected offers for purchase the customer may choose to purchase the created package. Continuing with the customer flow the reseller would require a payment from the customer.

Depending on the needs of the reseller this can be done with two different workflows.

### 8.3.1. Direct

If there is no need for data consistency, the reseller may choose to directly purchase the products from a provider.

---

*POST /processes/purchase-package/execute { packageId }*

---

### 8.3.2. 2-phase

Having the ability to purchase in a two phase commit the resellers have a better way of handling errors in payment flows and ensuring data consistency. The flow starts with a process of 2-phase-purchase-package, and transition over to commit-package when everything has been finalized with the customer interaction.

---

*POST /processes/2-phase-purchase-package/execute { packageId }*

---

---

*POST /processes/commit-package/execute { packageId }*

---

#### **8.4. Add ancillaries to a package**

The Add ancillaries to a package is requested by the Distributor and the response is provided by the Attributor.

When a package is selected ('pre-purchased'), it can be modified. One of the options is to add additional ancillaries to the package. The process to do this, is requesting available ancillaries for the package (and optionally leg), and you can add them using the add-ancillaries function.

---

*GET /collections/ancillaries/items { packageId, [legId] }*

*POST /processes/add-ancillaries/execute { packageId, ancillaryId }*

---

#### **8.5. Select and change an asset (e.g. seat)**

Allowing customers to specify their preferred asset (specific seat) from a deck plan. Some products allow for this to be selected by the customer. Other products do not allow this workflow to be executed. Customer is only allowed to set the seat parameter which is within their products possibilities.

---

*POST /processes/assign-asset/execute*

---



## 8.6. Cancel a package

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

To cancel the entire package after purchase, use the cancel-package endpoint. This is initiated by the reseller, to correct errors and request a full refund of the package (if payment has already been settled).

---

```
POST /processes/cancel-package/execute { packageId, reason }
```

---

## 8.7. Refund complete purchase

The Refund complete purchase is requested by the Distributor and the response is provided by the Attributor.

After the purchase, it is possible to retrieve possible refund options for this package.

In order to safely perform the refund of a package which can include offers from multiple providers, there are endpoints to claim and confirm the refund (2-phase). This ensures that all offers in the package are refunded and all resources are released for all participants (assets, ancillaries).

The refund options may vary for the different offers in the booking, ranging from not refundable to fully refundable, and the options can be different according to the time left to departure. The refund options typically have an expiration date and time, and the refund options should be claimed and confirmed before they expire.

---

```
GET /collections/refund-options/items  
{ packageId, type='refund-package' }
```

```
POST /processes/claim-refund-options/execute  
{ packageId, refundOptionId }
```

```
POST /processes/confirm-refund-option/execute  
{ packageId, refundOptionId }
```

---

Partial refunds are further described in the Advanced use cases section.

## 8.8. Add or change personal data

The Add or change personal data is requested by the Distributor and the response is provided by the Attributor.

Supplying personal data allows operators to contact travellers directly affected by service interruptions during travel. This information could either be supplied up front on the traveller or changed on a later time, if there is a change in mind from the traveller.

---

```
POST /processes/add-traveller/execute
```

```
POST /processes/update-traveller/execute
```

---

## 9. Advanced use cases

### 9.1. Partial cancellation by reducing the number of passengers

The Partial cancellation by reducing the number of passengers is requested by the Distributor and the response is provided by the Attributor.

The endpoints for calculating, claiming and confirming refund options also support refunding offers for specified travellers, while keeping the offers for the other travellers in the package.

---

```
GET /collections/refund-options/items  
{ packageId, type='remove-traveller', travellerId(s) }
```

```
POST /processes/claim-refund-option/execute  
{ packageId, refundOptionId }
```

```
POST /processes/confirm-refund-option/execute  
{ packageId, refundOptionId }
```

---

### 9.2. Refund ancillary

The Refund ancillary is requested by the Distributor and the response is provided by the Attributor.

By specifying the ancillary and leg, ancillary products can be refunded from a purchased package. The process is similar to the other refunding scenarios,

---

```
GET /collections/refund-options/items  
{ packageId, type='remove-ancillary', legId, ancillaryId }
```

```
POST /processes/claim-refund-options/execute  
{ packageId, legId, refundOptionId }
```

```
POST /processes/confirm-refund-option/execute  
{ packageId, legId, refundOptionId }
```

---

### 9.3. Exchange offers in a purchased package

The Exchange offers in a purchased package is requested by the Distributor and the response is provided by the Contributor.

To exchange all the offers in a purchased package, the reseller should include the purchased package in a request for a new set of offers that reflect the changes that the traveller wants, e.g. to another date or a different class of use. When new offers are selected with the select-offers endpoint, the new package should reflect the cost of the exchange, including potential fees. The original package (packageToExchange) is cancelled without refund.

---

```
POST /processes/search-offers/execute  
    { packageToExchange }
```

```
POST /processes/select-offers/execute  
    { offerIds }
```

```
POST /processes/2-phase-purchase-package/execute  
    { packageId }
```

```
POST /processes/confirm-package/execute { packageId }
```

---

## 10. Additional use cases

### 10.1. Passengers with reduced mobility

Passengers with reduced mobility should be able to describe their needs when planning a trip. The only facility the API can offer is to make it possible to supply the fact that the traveller has reduced mobility / mobility needs. The part where the traveller can request help, is not in scope (yet) of this API.

## 10.2. Passengers travelling with bicycle

The Exchange offers in a purchased package is requested by the Distributor and the response is provided by the Attributor.

There are two ways to facilitate this function. One way is to add the bike-spot as an ancillary: this case is handled through the normal endpoints for searching and selecting offers and ancillaries. Passengers travelling with bicycles will find ancillary products for bringing a bicycle on board in endpoint for retrieving ancillary products. It can then be added using the add-ancillaries endpoint.

---

*GET /collections/ancillaries/items { packageId, [legId] }*

*POST /processes/add-ancillaries/execute { packageId, ancillaryId }*

---

The other way is to provide the information as requirement in the search-offer request.

---

*POST /processes/search-offers/execute { travelParty-requirement }*

---

## 10.3. Passengers bringing a car

The Exchange offers in a purchased package is requested by the Distributor and the response is provided by the Attributor.

Further work is needed on how to plan a trip when bringing a car or other vehicles on board. An accompanied car transport on train, where the passengers use normal transport products and need to reserve an additional spot for their car can be handled with the endpoints for searching and selecting offers for the passengers. The process of reserving a spot allocation for the car can then be handled in the same way as the ancillary selection and asset allocation using the already described endpoints.

---

*GET /collections/ancillaries/items { packageId, [legId] }*

*POST /processes/add-ancillaries/execute { packageId, ancillaryId }*

---

Accompanied cars on car shuttle trains, where the passengers stay in their vehicle, would be handled by selecting an offer for a product covering both the car and the passengers. The offer can thus be searched and selected.

---

*POST /processes/search-offers/execute { travelParty-requirement }*

*POST /processes/select-offers/execute { offerId(s) }*

---

#### 10.4. Redresses

If one or more of the quality levels guaranteed in the package is not met, the traveller can request a redress to amend the broken guarantee. There are different ways to redress a broken guarantee, depending on to which degree the guarantee was broken.

The specific set of endpoints for handling redresses are left for future work, but can be accomplished by adding something like the following:

---

*GET /collections/trip-repair-options/items { packageld, offerId, legId, assetId,  
ancillaryId, .. }*

*POST /processes/claim-trip-repair-option/execute*

*POST /processes/confirm-trip-repair-option/execute*

---

## 11. Technical aspects

### 11.1. Self-explaining

By complying to the OGC API Records, Features and Processes, we have to implement a set of endpoints, facilitating 'discovery'. Once the root of the API is published and found, the reseller should be able to discover the API.

OGC prescribes there that the discovery (the meta-data) must be available in a human and in a machine readable format.

The set of OGC endpoints that must be implemented are (in html and json):

- '/' – the root or landingpage
- /api – containing the OpenAPI definition
- /conformance – gives a list of standards used in the API
- /collection – returns a list of available collections
- /collections/{collectionId} – returns a description of the collection
- /processes – returns a list of available processes
- /processes/{processId} – returns a description of the process

All these endpoints deliver a fixed output, so it is normally a one-time effort to create them.

### 11.2. Authentication

The not open part of the API requires to obtain access and authorisation by supplying a JWT ('token', RFC 7519) to identify yourself, and so you can be authenticated and authorized to use the endpoint.

#### 11.2.1. Authentication flows

The several ways to get access to data sets or operations are:

- Using PKI, in the handshake protocol (mTLS) the certificate is delivered and the JWT token can be created using the credentials in the certificate when a JWT token request is executed
- Using OAuth (Client credentials flow), to obtain a JWT token
- Using username/password to obtain a JWT token
- Using a fixed token, provided in another way

To obtain the JWT token, endpoints are provided in the API, but they COULD be implemented by independent Identity Provider.

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.

#### 11.2.2. JWT structure

The JWT token itself is a standard JWT token. In the so-called 'payload' these fields MUST be available:

iss	The issuer, like "iss": "https://railwayundertaking.com/"
sub	The user id, that can be <ul style="list-style-type: none"><li>- the (O) or (CN) from the certificate,</li><li>- the client_id from the client credential flow or</li><li>- the username from the username/password authentication</li></ul>
exp	Expiry time (UNIX timestamp, in seconds)
iat	Issuing time (UNIX timestamp, in seconds)

There are also a few optional fields:

nbf	Not before (this token is only valid after this UNIX timestamp, in seconds)
jwt	A unique identifier for this JWT

The 'header' part of the JWT is complying to the JWT standard, specifying explicitly that the used encryption method is SHA 256 and the type is JWT (and not JWS or JWE):

```
{ "alg": "HS256", "typ": "JWT" }
```

### 11.2.3. Authentication endpoint

The authentication endpoint, that will deliver a JWT as described above, can be found in the ‘tech’ part of the description.

---

```
POST /oauth/token { username/password }
```

```
POST /oauth/token { client_id/client_secret }
```

```
POST /oauth/token () { PKI (mTLS) }
```

---

## 11.3. Authorisation

Within the API we can distinguish open data sets, data sets that could be open and data sets that require authorisation.

### 11.3.1. Open endpoints

The open endpoints can be found in the API description, they are solely marked with

security:

- None

/	<b>Discovery:</b> Landing page, in HTML and JSON, providing general information and ‘bootstrap’ links.
/conformance	<b>Discovery:</b> Exposing all formal formats, data standards etc that are used.
/api	<b>Discovery:</b> A formal OpenAPI specification of the API
/collections	<b>Discovery:</b> Describing all collections that are available in the API (open and secured)
/processes	<b>Discovery:</b> Describing all processes facilitated in the API (normally secured)
/collection/{collectionId}	<b>Discovery:</b> Describing the information that is in that specific collection, including the format
/processes/{processId}	<b>Discovery:</b> Describing the information that is in that specific collection, including the format

### 11.3.2.



### 11.3.3. Secured endpoints

The majority of the endpoints needs authentication & authorization, they are marked with other security schemes, like BearerAuth (authentication with a JWT, obtained anywhere), OAuth (also a JWT authentication, but the token is obtained by the client identity flow) or OAuthPKI (this requires mTLS, the token is obtained by supplying a PKI certificate).

/collection/{collectionId}/items	Containing the data of collections like refund-options, assets, ancillaries, travel-documents
/processes/{processId}/execute	Triggering a process on packages, legs, products, etc. You need to be authenticated & authorised to execute these.

### 11.4. Pagination

OGC API Record and Features describe a standard way to handle pagination for large data sets. Roughly said, there is at the end of a result (a collection) a facility for 'links', that include links to the next (and previous) part of the complete data set. That requires also that there are a few arguments by default:

- Offset (where the requested part of the data set should start)
- Limit (the number of rows/features that have to be returned at max)

#### Example

- First, we call /collections/{some collection}/items, and it returns the first 100 entries.
- At the bottom of the result, there is a link, using the relation "next" and the url specified there is collections/{some collection}/items?offset=100&limit=100, to retrieve the next 100 entries.
- Using the link will return up to the next 100 entries, and at the bottom, we'll find 2 links: next - collections/{some collection}/items?offset=200&limit=100 and previous - collections/{some collection}/items?offset=0&limit=100

## 12. Appendix A: OGC

In the API we comply to the OGC standards, especially the OGC API Records, Features and Processes. This makes it easy to understand, and have a structure to hang on to.

### 12.1. OGC API Records & Features

All requests for information are handled by a

---

*GET /collections/{collectionId}/items*

---

where you can filter the collection based on provided arguments.

The result (response) of a call to one of these endpoints do always have this structure (unless it is described and underpinned why it is different):

All results have the format of:

```
{ "type": "<conceptName>Collection"
, "<conceptName>":
[ { "type": "<conceptName>"
, "geometry": only applicable in OGC API Feature calls
, "id" : id of the instance
, "properties": {
" type": "<conceptName>",
, "id" : id of the instance
... (other properties of the concept)...
}
, "links": [ ...available detail links AND links for possible operations on this concept ... ]
} (end of concept 1)
, { concept 2 }
]
, "properties": { ... properties of the collection ... }
}
, "links": [ ... links for the collection, like next part of the data set ... ]
}
```

If the endpoint is OGC API Feature compliant, the conceptName is 'Feature', but in the properties, the type will remain the requested concept type. This will result in a valid GeoJSON.

This structure allows us to structure things. The 'properties' of the feature will contain Transmodel compliant concepts and the links of the feature will expose possible next steps for the concept (link 'purchase' or '2-phase-purchase' for an offer). The links could also contain additional information, like 'details', where you will be able to 'open' the offer, and look at the details of the legs and products.

At the bottom of the response, you'll find the 'properties' of the collection, like 'type': 'offers', which will clarify that this is a list of offers. The links at the bottom will again provide additional information, or things like 'next', referring to the next part of the result.

### 12.2. OGC API Process

All request that modifies data, have this format:

---

## POST /processes/{processId}/execute

---

The only exception on this rule is 'search-offers', but this is due to the limitations of the http-get. According to the rules, it should have been a collection.

The format of all process requests have this format:

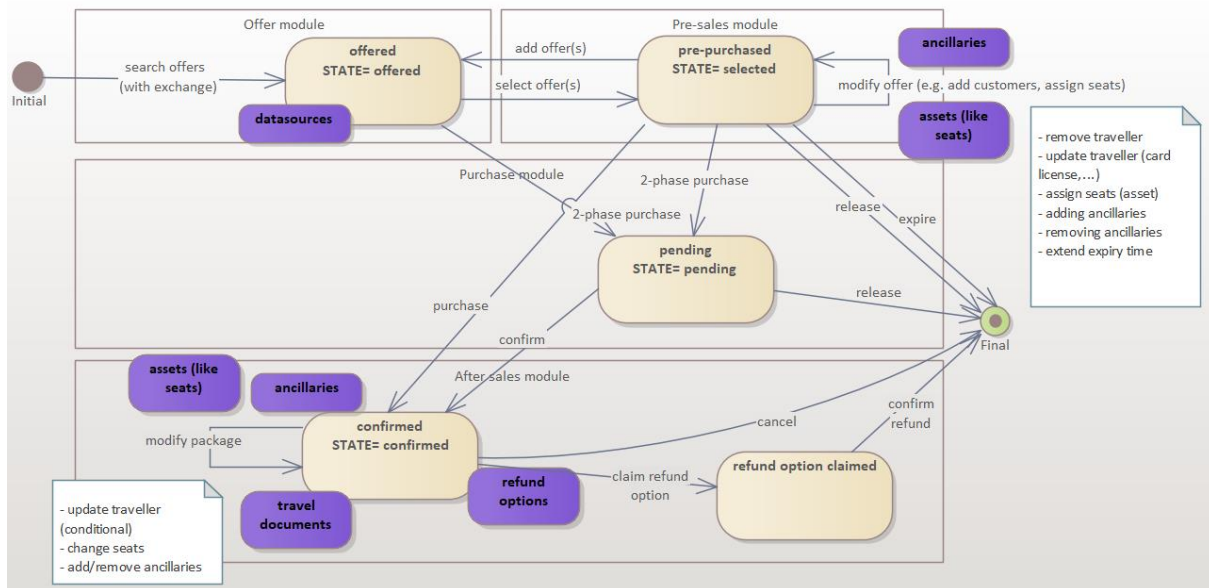
```
{  
  "inputs": { the functional input arguments }  
  , "subscriber": { this part is only relevant when you work a-synchronously, what we support as well }  
}
```

In the documentation below, we will only describe the 'inputs' part.

The output of the processes described return very often a single, but complete package.

### 13. Appendix B: States

This diagram below shows the states and the transitions between them.



It is NOT required to have all transitions implemented. The transitions we formalize as OGC processes: /processes/{transition}/execute, where it is formatted as (an operation)-(an concept), like 'purchase-package'.

The purple boxes are data sets that are required in the process, like when you want to add an ancillary, you have to know the ancillaries that apply. After that, you can add one (or more) ancillaries from the list. These data sets are formalized as OGC records (or in case of the assets, as OGC Features).

#### 13.1. Offer module

The main concept in the 'Offer module' is the offer, supported with the list of data sources to look up the tariffs in the source data.

- Search-offers
- Get datasources

#### 13.2. Pre-sales module

In the 'Pre-sales module', you can select offer(s) into a package, execute a variety of operations on the travellers or legs, and can request to have additional expiry time. The main concepts in here are 'traveller', 'asset' and 'ancillary'.

- Select-offers
- Remove-traveller
- Update-traveller
- Assign-asset
- Assign-ancillary
- Expend-expiry-time

#### 13.3. Purchase module

When the package is ready to purchase, you can purchase it directly OR purchase it in a 2-phase purchase. That means that you can purchase the package, but have to confirm the purchase, allowing you to coordinate the purchase with other dependencies, like a confirmation of a bank, or operators of other legs. The package itself is the main concept in this module.

- Purchase-package
- 2-phase-purchase-package
- Confirm-package
- Release-package

#### **13.4. After-sales module**

In the 'After-sales module', a refund can be requested by the reseller (with a valid reason, like repairing mistakes, using the cancel-flow), or requested by the customer (using the refund-flow).

- Upselling:
  - Update-traveller (under strict conditions)
  - Get assets & Assign-asset
  - Get ancillaries & Assign-ancillary
- Cancel-package
- Get refund-options, claim-refund-option & confirm-refund-option

## 14. Appendix C: Endpoints

### 15. Offer module

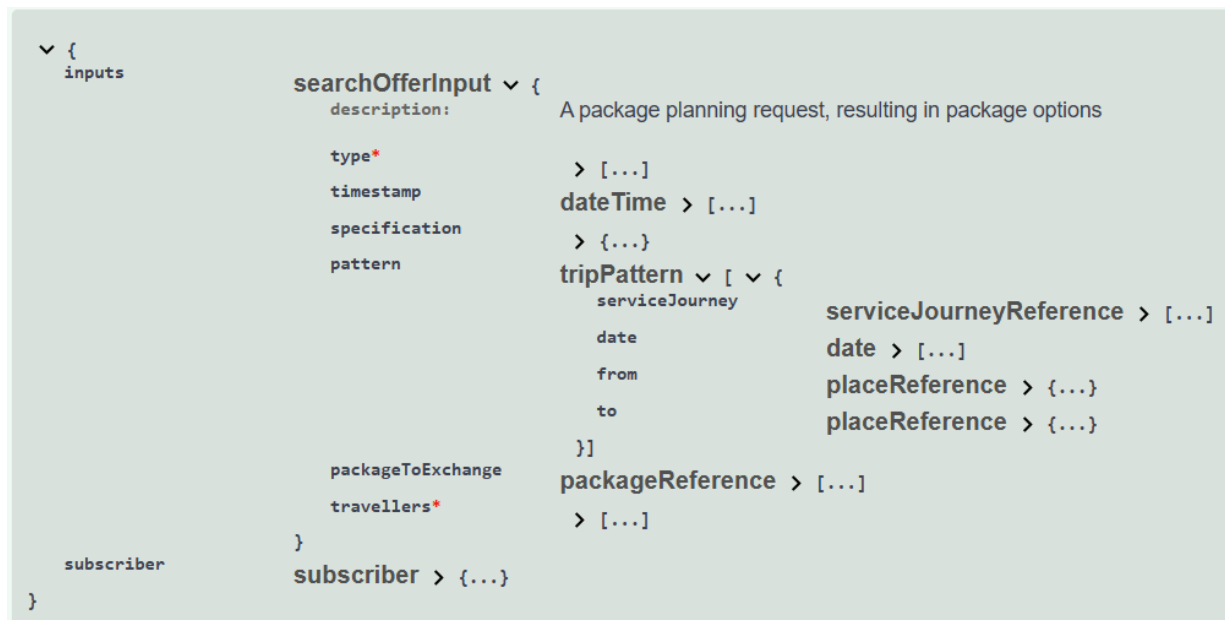
The first module you need after executing a trip planning (using a trip planner like OTP or OJP), is requesting per 'leg' offers. When the requested offer is already in the response of the search, it can be purchased immediately, using the functionality in the Purchase Module. Otherwise, you have to 'tailor' the offer(s) to what you need, and purchase it in a single package.

#### 15.1. Search offers

**Endpoint:** `POST /processes/search-offers/execute`

This endpoint is OGC API Process compliant, but it returns a data set that is also compliant with the OGC API Records.

**Request:**



*Example request to travel from Oslo to Bergen:*

```

{
  "type": "search_offer",
  "pattern": [
    {
      "from": { "id": "NSR:StopPlace:337", "name": "Oslo S" },
      "to": { "id": "NSR:StopPlace:548", "name": "Bergen stasjon" },
      "serviceJourneyId": "PEN:ServiceJourney:67-LOD_253085-R" },
    { "serviceDate": "2025-04-02" } ],
  "travellers": [ { "type": "user_profile",
    "id": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8",
    "ageGroup": "adult" } ] }

```

**Response:**

```
offerCollection v {
  type* > [...]
  offers v [offer v {
    id
    type
    properties
    shortString > [...]
    v {
      legs v [leg > {...}]
      ancillaries v [ancillary > {...}]
      products v [product > {...}]
      price amountOfMoney > {...}
      summary > {...}
      guarantees v [guarantee > {...}]
    }
    links v [link > {...}]
    x-tm "SALES OFFER PACKAGE"
  }]
  numberMatched > [...]
  numberReturned > [...]
  links > [...]
}
```

*Example response to travel from Oslo to Bergen:*

```
{ "type": "OfferCollection",
  "offers": [
    { "type": "Offer"
      , "id": "96217fe2-c7ac-4f49-9561-4f6245c5e28b"
      , "properties": {
        "legs": [ { "type": "leg"
          , "id": "ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
          , "startTime": "2025-04-02T21:03:00Z"
          , "endTime": "2025-04-03T04:27:00Z"
          , "from": { "id": "NSR:StopPlace:337", "name": "Oslo S" }
          , "to": { "id": "NSR:StopPlace:548", "name": "Bergen stasjon" }
          , "traveller": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8"
          , "products": [ { "id": "PEN:Product:NightRail" } ] }
        , "products": [ { "type": "product",
          "id": "PEN:Product:NightRail" } ]
        , "price": { "amount": 1193, "currencyCode": "NOK" }
        , "mode": "TRAIN"
        , "id": "96217fe2-c7ac-4f49-9561-4f6245c5e28b"
      }
    ]
    , "links": [
      { "rel": "purchase-package",
        "url": "/processes/purchase- package /execute"
        , "body": { "offerId": "96217fe2-c7ac-4f49-9561-4f6245c5e28b" } }
      { "rel
        "url
        , "body": { "offerId": "96217fe2-c7ac-4f49-9561-4f6245c5e28b" }
    }
  ]
  , "numberMatched
  , "numberReturned
}
```



## 15.2. Get data sources

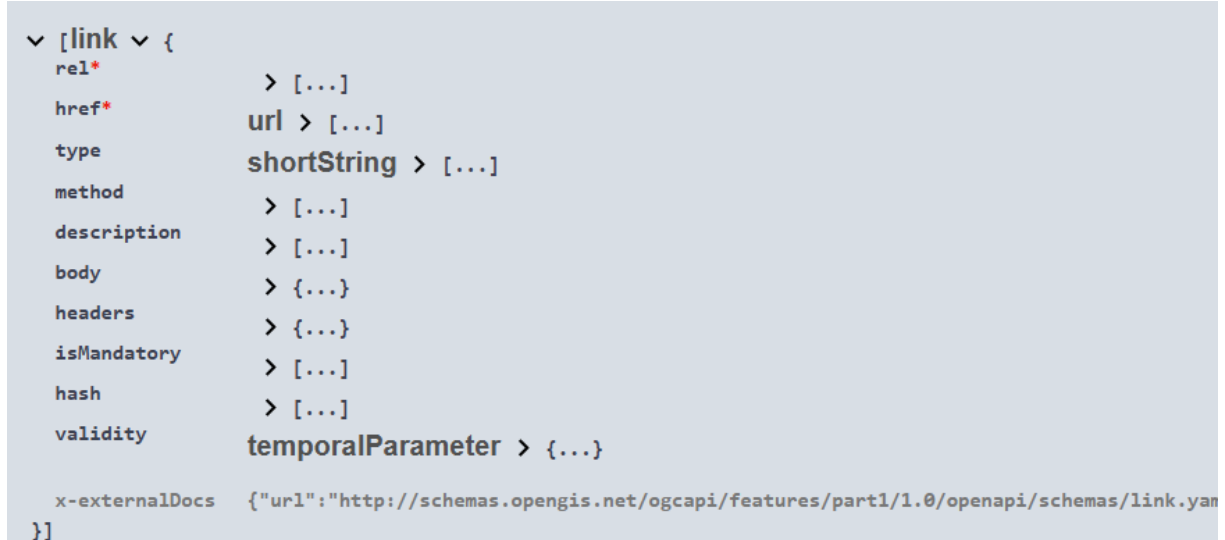
**Endpoint:** GET /collections/datasources/items

This endpoint returns all used external data sources in the implementation of the API, referring to where the data can be obtained.

**Request:**

This endpoint does not have any parameters, nor a body.

**Response:**



Example:

```

[ { "rel": "PEN:ServiceJourney",
  "url": "https://storage.googleapis.com/marduk-production/outbound/netex/rb_pen-aggregated-netex.zip",
  "type": "application/xml+NeTEx" },
  ...
]

```

### 15.3. Pre-sales module

#### 15.4. Select offer(s)

**Endpoint:** POST /processes/select-offers/execute

This endpoint is OGC API Process compliant, selecting one or more offers and combining it into one package.

**Request:**

```

  {
    inputs

    selectOffersInput {
      description: superclass for all request bodies. Not every subclass has to be in your implementation, it depends on your selection of modules.

      type*
        string
        Enum:
          [ select_offers ]

      timestamp
        dateTime > [...]

      offerIds*
        [offerReference > [...]]
    }

    subscriber
      subscriber > {...}
  }
}

```

*Example request to select a single offer from the search result*

```
{ "type": "select_offer"
, "offerIds": ["96217fe2-c7ac-4f49-9561-4f6245c5e28b"]} }
```

**Response:**

```

package v {
  description:
    from
    via
    to
    startTime
    endTime
    placeDefinitions
    type*
    id
    status*
    price*
    offers*
    guarantees
    travellers
    links

  x-tm

}

```

*Example response, a selected offer in a package*

```
{ "type": "package",
  "id": "5f75913f-a8bc-4e2d-b060-bd431744bc7f",
  "status": "selected",
  "price": { "amount": 1193, "currencyCode": "NOK" },
  "offers": [ an offer, like the one from the search-offers output ],
  "guarantees": [],
  "travellers": [{ "type": "user_profile",
                   "id": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8",
                   "ageGroup": "adult" }]
}
```

### 15.5. Update traveller

**Endpoint:** `POST /processes/update-traveller/execute`

This endpoint is OGC Processes compliant, allowing to update traveller details, like updating a name or adding additional information to make it from an anonymous traveller (`user_profile`) to a reference of a physical person (`individual_traveller`).

This endpoint can be used as before and after the purchase, but after the purchase it must be used more stringently. Things like modifying the age shouldn't be allowed.

**Request:**



*Example request to set the name of the traveller:*

```
{ "type": "traveller",
  "packageId": "5f75913f-a8bc-4e2d-b060-bd431744bc7f",
  "travellerId": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8",
  "updatedTraveller": { "type": "individual_traveller",
                        , "fullName": "J.Doe" }
}
```

**Response:**

It is again the package response. It contains the same data, except that the 'travellers' field:

```
"travellers": [{ "type": "user_profile",
                  "id": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8",
                  "ageGroup": "adult" },
                { "type": "individual_traveller",
                  "id": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8",
                  "fullName": "J.Doe" } ]
```

It contains now an additional entry, containing the new data for the same person. If the 'updatedTraveller' would have been the same type, this entry would have been replaced by the new one.

**15.6. Remove traveller**

Only allowed before purchase. After the purchase, a (partial) refund request is needed.

**Endpoint:** POST </processes/remove-traveller/execute>



The same input is used as in the 'update-traveller', but now the updatedTraveller-field will have no meaning.

**Request:**

*Example request to remove the traveller*

```
{ "type": "traveller",
  "packageId": "5f75913f-a8bc-4e2d-b060-bd431744bc7f",
  "travellerId": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8" }
```

**Response:**

The result is again the package result.

Of course, this example would lead to an error, since it is the last (and first) traveller.

If there would have been more travellers, the traveller entries with the same travellerId should be removed from the package result, and of course also all its legs, and if there are products/ancillaries/guarantees that are not used anymore (since their legs have been removed), these should be removed as well.

**15.7. Find assets**

**Endpoint:** `GET /collections/assets/items`

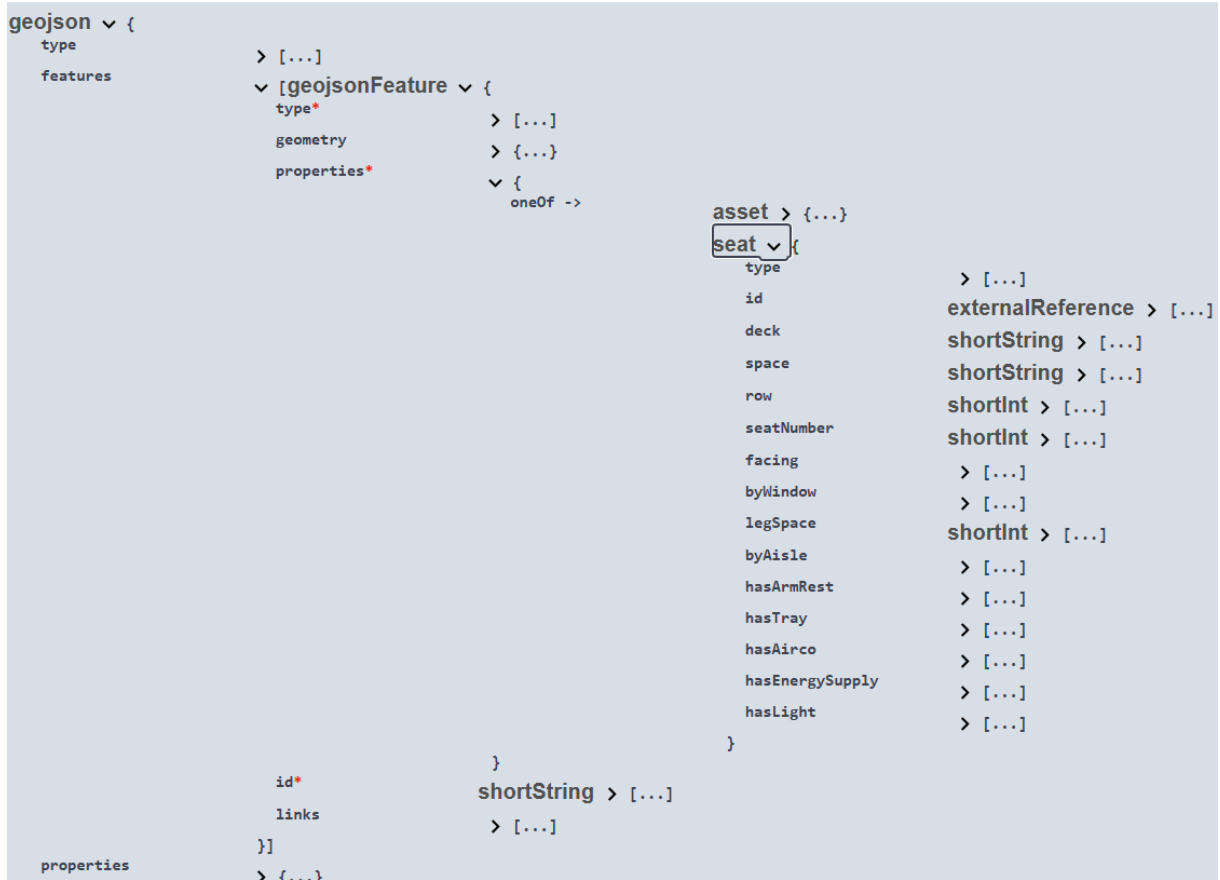
This functionality allows us to retrieve all assets, and their status, that are applicable for a specific leg. It returns a list of e.g. seats or assets on the street (bikes, cars). The endpoint complies to OGC API Features, and thereby delivers geoJSON. For the overview of seats, geoJSON with a local coordinate system is used.

**Request:**

Name	Description
<b>packageId</b> * required string (query)	the identifier of a package <input type="text" value="packageId"/>
<b>legId</b> * required string (query)	leg identifier <input type="text" value="legId"/>

*Example request to retrieve all available seats for a specific leg in a package*

`GET /collections/assets/items?packageId=5f75913f-a8bc-4e2d-b060-bd431744bc7f &legId=ff6c6c75-8cf0-4d5c-9e54-962e06a12e86`

**Response:**

The geoJSON is used to depict the assets for shared mobility or seats. In case it contains seats, the coordinate system is a bit strange, but it has to be changed to a local CRS:

```
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:EPSG::99999" } }
```

```
{ "type": "FeatureCollection"
```

```
, "features": [ { "type": "Feature"
```

```
, "geometry": some geometry
```

```
, "properties": { "type": "seat"
```

```
, "id": "462-52"
```

```
, "space": "462", "spot": "52"
```

```
, "facing": "forward"
```

```
, "byWindow": true }
```

```
, "links": [ { "rel": "assign-asset",
```

```
"url": "/processes/assign-asset/execute",
```

```
"body": { "type": "assign-asset"
```

```
, "packageId": "5f75913f-a8bc-4e2d-b060-bd431744bc7f"
```

```
, "legId": "ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
```

```
, "assetId": "462-52" }
```

```
}
```

```
]
```

```
]
```

```
}
```

### 15.8. Assign asset (e.g. a seat)

This function allows us to assign an asset (like a seat) to a leg. It is OGC Processes compliant. When you want to replace one asset with another (that must be available), you put the old asset in 'replaceAsset', and the new one in 'asset'.

When you want to remove the assigned asset, you simply put it in the 'replaceAsset', leaving the 'asset' property empty.

**Endpoint:** `POST /processes/assign-asset/execute`



#### Request:

Example request to assign seat 462-52 to the package/offer/leg combination.

```
{ "type": "assign-asset"
, "packageId": "5f75913f-a8bc-4e2d-b060-bd431744bc7f"
, "legId": "ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
, "assetId": "462-52" }
```

#### Response:

It is again the complete package, but now in the leg you'll find this:

```
"legs": [ { "type": "leg"
, "id": "ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
, "startTime": "2025-04-02T21:03:00Z"
, "endTime": "2025-04-03T04:27:00Z"
, "from": { "id": "NSR:StopPlace:337", "name": "Oslo S" }
, "to": { "id": "NSR:StopPlace:548", "name": "Bergen stasjon" }
, "traveller": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8"
, "products": [ "id": "PEN:Product:NightRail" ]
, "assets": [ { "id": "462-52" } ] }
```

### 15.9. Find ancillaries

This function is OGC API Features compliant, returning all applicable ancillaries.

**Endpoint:** POST /collections/ancillaries/items

GET /collections/ancillaries/items Handles ancillaries collections

Try it out

Name	Description
<b>packageId</b> * required string (query)	the identifier of a package <input type="text" value="packageId"/>
<b>legId</b> * required string (query)	leg identifier <input type="text" value="legId"/>

**Request:**

Example request to retrieve all available ancillaries for a specific leg in a package, comparable with the assets

GET /collections/ancillaries/items?packageId=5f75913f-a8bc-4e2d-b060-bd431744bc7f &legId=ff6c6c75-8cf0-4d5c-9e54-962e06a12e86

**Response:**

```

{
  "type": "AncillaryCollection",
  "ancillaries": [
    {
      "id": "standard_meal",
      "properties": {
        "ancillaryId": "standard_meal",
        "name": "A normal meal"
      },
      "links": [
        {
          "rel": "detail",
          "url": "a url to an webpage describing the ancillary"
        },
        {
          "rel": "assign-ancillary",
          "url": "/processes/assign-ancillary/execute",
          "body": {
            "type": "assign-ancillary"
          }
        }
      ]
    }
  ],
  "numberMatched": 1,
  "numberReturned": 1,
  "links": [
    {
      "rel": "self",
      "url": "/collections/ancillaries/items?packageId=5f75913f-a8bc-4e2d-b060-bd431744bc7f&legId=ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
    }
  ]
}

```

*Example response*

```

{ "type": "AncillaryCollection"
  , "ancillaries": [
    { "type": "ancillary",
      "id": "standard_meal",
      "properties": {
        "ancillaryId": "standard_meal"
        , "name": "A normal meal" }
      , "links": [ { "rel": "detail", "url": "a url to an webpage describing the ancillary" }
        , { "rel": "assign-ancillary"
          , "url": "/processes/assign-ancillary/execute"
          , "body": { "type": "assign-ancillary"

```



```

    , "packageId": "5f75913f-a8bc-4e2d-b060-bd431744bc7f"
    , "legId": "ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
    , "ancillaryId": "standard_meal" } }
  ]
}]]}

```

### 15.10. Assign ancillary

One of the results of the previous step can be used to call this endpoint, which is OGC API Processes compliant. Referencing to a package, leg (and optionally an offer) and an asset, it can assign the asset to the leg.

**Endpoint:** POST /processes/assign-ancillary/execute



#### Request:

*Example request to assign an ancillary*

```

{ "type": "assign-ancillary"
, "packageId": "5f75913f-a8bc-4e2d-b060-bd431744bc7f"
, "legId": "ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
, "ancillaryId": "standard_meal" }

```

#### Response:

It is again the complete package, but now in the leg you'll find this:

```

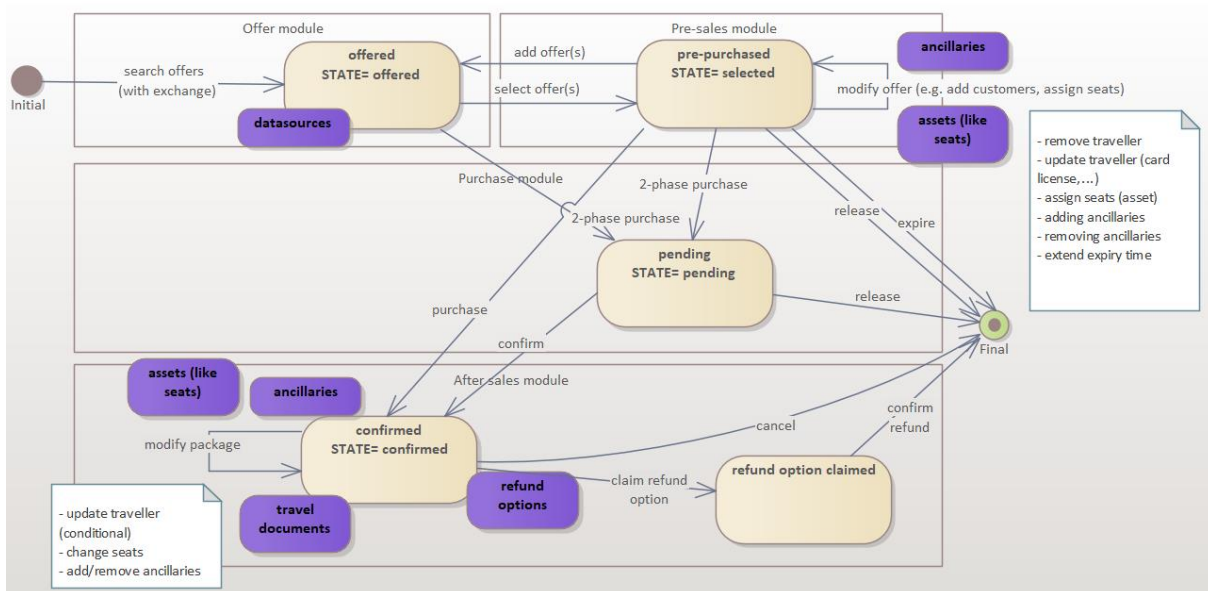
"legs": [ { "type": "leg"
, "id": "ff6c6c75-8cf0-4d5c-9e54-962e06a12e86"
, "startTime": "2025-04-02T21:03:00Z"
, "endTime": "2025-04-03T04:27:00Z"
, "from": { "id": "NSR:StopPlace:337", "name": "Oslo S" }
, "to": { "id": "NSR:StopPlace:548", "name": "Bergen stasjon" }
, "traveller": "c2ed41f5-2fd2-4f38-afb1-edbd479d30a8"
, "products": [ "id": "PEN:Product:NightRail" ]
, "assets": [ { "id": "462-52" } ]

```

```
, "ancillaries": [ { "id": "standard_meal" } ]
}]
```

## 16. Purchase module

Once the Offer module and the Pre-sales module have finished their activities, we'll end up in the Purchase module:



### 16.1. Purchase package, 2-phase purchase package, confirm package, release package

All of these endpoints do have the same request body and the same response. In the request the only thing that differs, is the 'type'. It is in order 'purchase-package', '2-phase-purchase-package', 'confirm-package' and 'release-package'. The response is always a package response.

They all comply with OGC API Processes.

#### Endpoints:

```
POST /processes/purchase-package/execute
POST /processes/2-phase-purchase-package/execute
POST /processes/confirm-package/execute
POST /processes/release-package/execute
```

#### Request:

```
{
  inputs: {
    packageInput: {
      type*: string
      timestamp: string
      packageId*: string
    }
    subscriber: string
  }
}
```

#### Response:

```

package {
  description: a purchased package is a registration of an agreement between end user and T0, to execute a package (=set of legs) according
               a specification, including all conditions
  from         placeReference > {...}
  via          > [...]
  to           placeReference > {...}
  startTime    dateTime > [...]
  endTime      dateTime > [...]
  placeDefinitions > [...]
  type*        > [...]
  id           > [...]
  status*      packageStatus > [...]
  price*       amountOfMoney > {...}
  offers*      > [...]
  guarantees   > [...]
  travellers   > [...]
  links        > [...]
  x-tm         "TRAVEL OFFER PACKAGE, CUSTOMER PURCHASE PACKAGE"
}

```

Example response for a package that is the response of a 'commit-package' request

```

{ "type": "package"
, ... the complete package, including legs, products etc ...
, "state": "confirmed" }

```

## 16.2. Get travel documents

**Endpoint:** `GET /collections/travel-documents/items`

When a package is confirmed, the travel documents can be retrieved using this endpoint, that is OGC API Records compliant.

**Request:**

GET
/collections/traveldocuments/items
Handles travel documents

Parameters
Try it out

Name	Description
<b>Accept-Language</b> * required string (header)	Accept-Language  <i>x-externalDocs: OrderedMap { "description": "A comma-separated List of BCP 47 (RFC 5646) Language tags and optional weights as described in IETF RFC7231 section 5.3.5. A List of the Languages/Localizations the user would like to see the results in. For user privacy and ease of use on the T0 side, this List should be kept as short as possible" }</i>
<b>authorization</b> * required string (header)	Header field, JWT must be supplied  authorization
<b>packageld</b> * required string (query)	the identifier of a package  packageld
legld string (query)	leg identifier  legld

**Response:**

```

travelDocumentCollection v {
  type* > [...]
  travelDocuments v [ v {
    id
    properties
    links
  }]
  numberMatched > [...]
  numberReturned > [...]
  links > [...]
}

normalString > [...]
  v {
    oneOf ->
      externalTicket v {
        description: External ticket,
          can be accessed
          using the links
          collection, with
          rel=ticket
        type > [...]
        x-tm [{"concept": "TRAVEL
          DOCUMENT"}]
      }
      binaryTicket v {
        description: Binary
          information,
          like a image
          or
          certificate
        type > [...]
        contentType* > [...]
        base64* longString
        version tinyString
        x-tm "lacking"
      }
    }
  }
  links > [...]
}

```

## 17. After-Sales module

### 17.1. Find refund options for a package

This function allows to request refund options, based on package, and e.g. leg, offer, traveller, asset or ancillary. It is OGC API Records compliant.





**Endpoint:** `GET /collections/refund-options/items`

**Request:**

GET

/collections/refund-options/items

Handles refund options collections

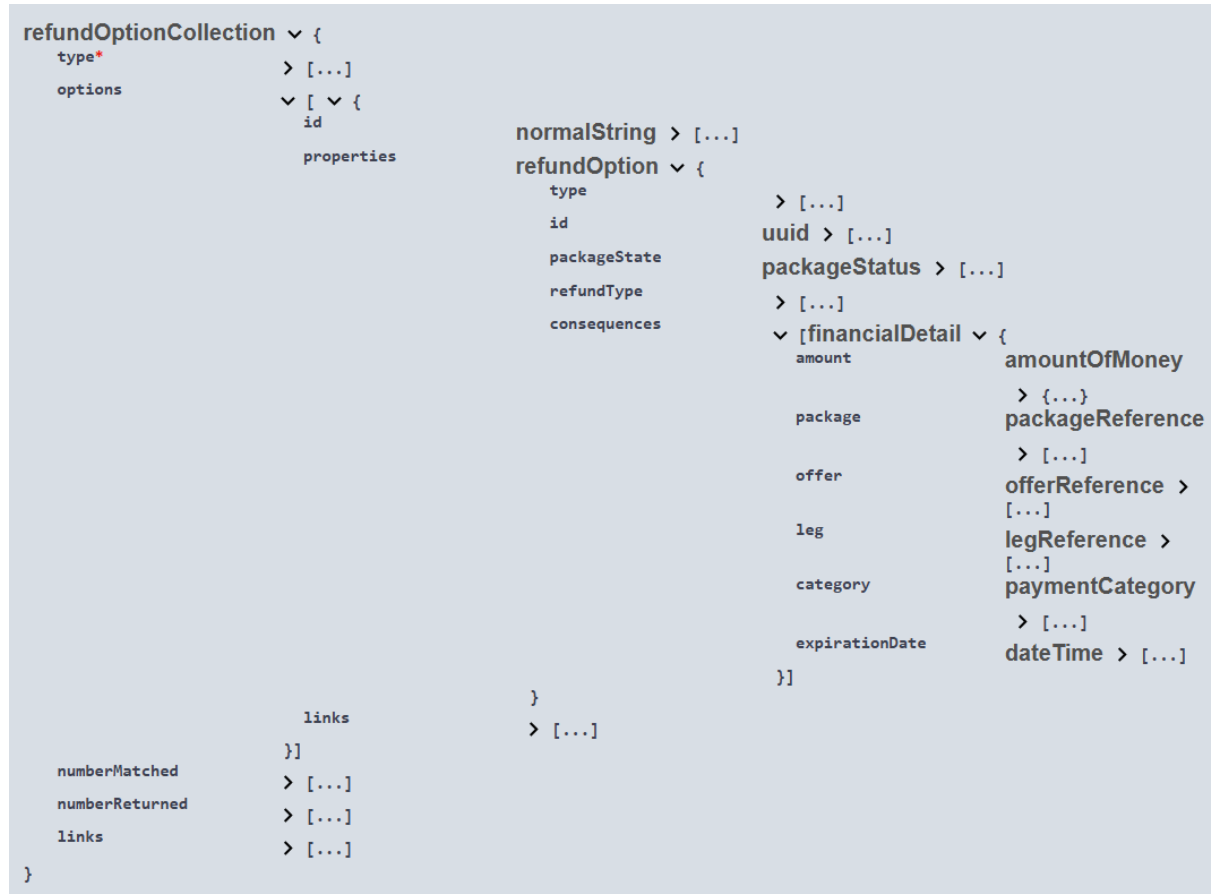


Parameters

Try it out

Name	Description
<b>packageId</b> * required	the identifier of a package
string (query)	<input type="text" value="packageId"/>
legId	leg identifier
string (query)	<input type="text" value="legId"/>
travellerId	traveller identifier
string (query)	<input type="text" value="travellerId"/>
ancillaryId	ancillary identifier
string (query)	<input type="text" value="ancillaryId"/>

*Example request to get refund options for a package (full refund)*  
`GET /collections/refund-options/items?packageId=5f75913f-a8bc-4e2d-b060-bd431744bc7f`

**Response:***Example response*

```

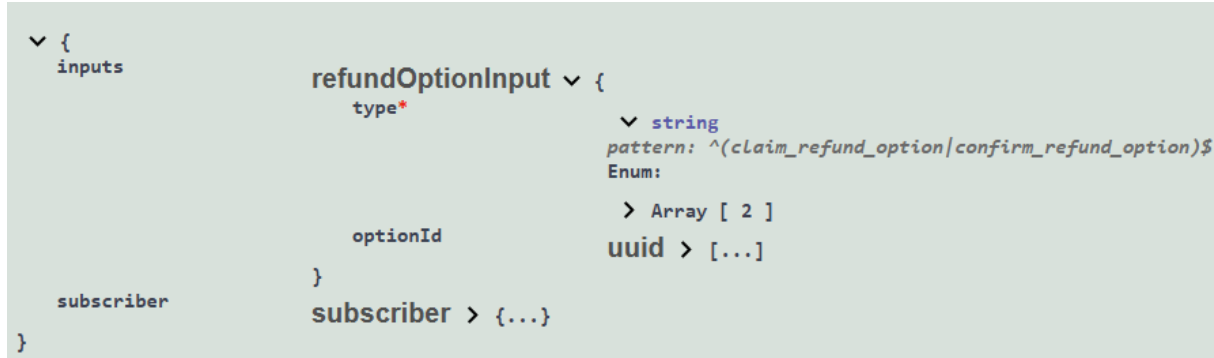
{ "type": "RefundOptionCollection"
  "options": [
    { "id": "07bfca3c-4f96-4920-b145-5f203ce0807f"
      , "properties":
        { "type": "refund-option"
          , "id": "07bfca3c-4f96-4920-b145-5f203ce0807f"
          , "packageState": "cancelled"
          , "refundType": "package_refund"
          , "consequences": [ { "category": "voucher"
                                , "amount": 1000
                                , "currencyCode": "NOK" } ]
        }
      , "links": [ ]
    }
  ]
}

```

## 17.2. Claim a refund option

With this endpoint, the selected refund option can be claimed, OGC API Processes compliant.

**Endpoint:** POST /processes/claim-refund-option/execute



### Request:

Example request to refund the option from the previous example

```
{ "type": "claim-refund-option"
, "optionId": "07bfca3c-4f96-4920-b145-5f203ce0807f" }
```

### Response:

It returns again a package response.

Example response

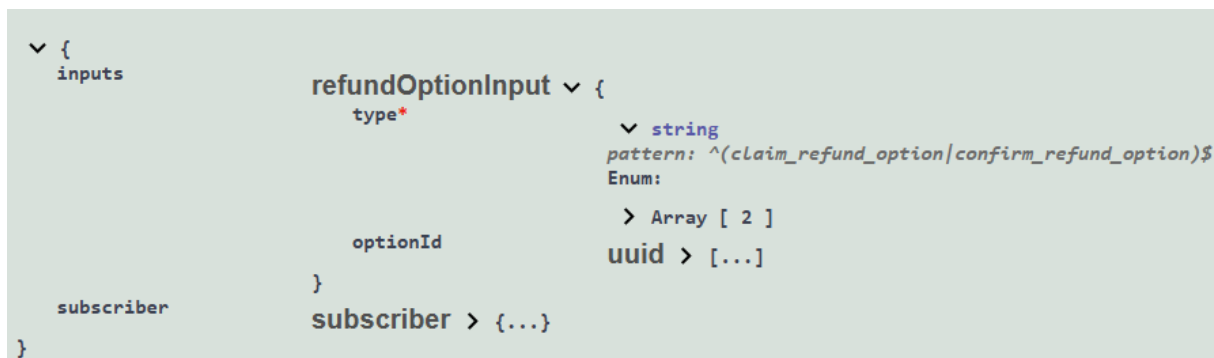
```
{ "type": "package"
, ... the complete package, including legs, products etc ...
, "state": "refund_claimed" }
```

## 17.3. Confirm a refund option

With this endpoint you can confirm the refund option that has been claimed before. It is OGC Process compliant.

**Endpoint:** POST /processes/confirm-refund-option/execute

### Request:



Example request to confirm the claim

```
{ "type": "confirm-refund-option"
, "optionId": "07bfca3c-4f96-4920-b145-5f203ce0807f" }
```

### Response:

It returns again a package response.

*Example response*

```
{ "type": "package"
, ... the complete package, including legs, products etc ...
, "state": "refunded" }
```

#### 17.4. Refund without options (in case of emergency)

This endpoint can be used to revert the purchase, only to be initiated by the reseller.

**Endpoint:** POST /processes/cancel-package/execute

**Request:**

```

{
  inputs
  packageInput {
    type*
    timestamp
    packageId*
  }
  subscriber
}

```

> [...]
   
dateTime > [...]
   
packageReference > [...]

*Example request to cancel the package*

```
{ "type": "refund-package",
  "packageId": "5f75913f-a8bc-4e2d-b060-bd431744bc7f" }
```

**Response:**

*Example response*

```
{ "type": "package"
, ... the complete package, including legs, products etc ...
, "state": "cancelled" }
```



## 18. Copyrights

Licensed under the Apache License, Version 2.0 (the "License");  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
See the License for the specific language governing permissions and  
limitations under the License.