Service API Reference

Digital Connect

Version 1.0

Sabre

Software version 1.0

Document Edition 1.0 (May 2016)

Template Version 2.5

Table of Contents

Introduction 1

This document contains reference information about the services provided by Digital Connect, Release 1.0.

1.1 Overview: About this Guide

This document describes the goals and current capabilities of an API that gives Sabre Airline Solutions customers the ability to access Sabre technologies, such as flight search, seat map, and flight purchase, as discrete services. Except for some basic dependencies (for example, you cannot call the purchase service until the passenger has used other services to build an itinerary), airlines are able to call these services in the order they choose. This gives airlines great freedom when writing their own client software. Airlines will not only be able to establish a distinctive look and feel for their websites, but they will also be able to design their own flows for displaying information to their passengers and prompting their passengers for input.

Digital Connect (DC) is a service-oriented, RESTful API. The remainder of this document provides detail about the 1.0 release of this API.

Digital Connect Services

2

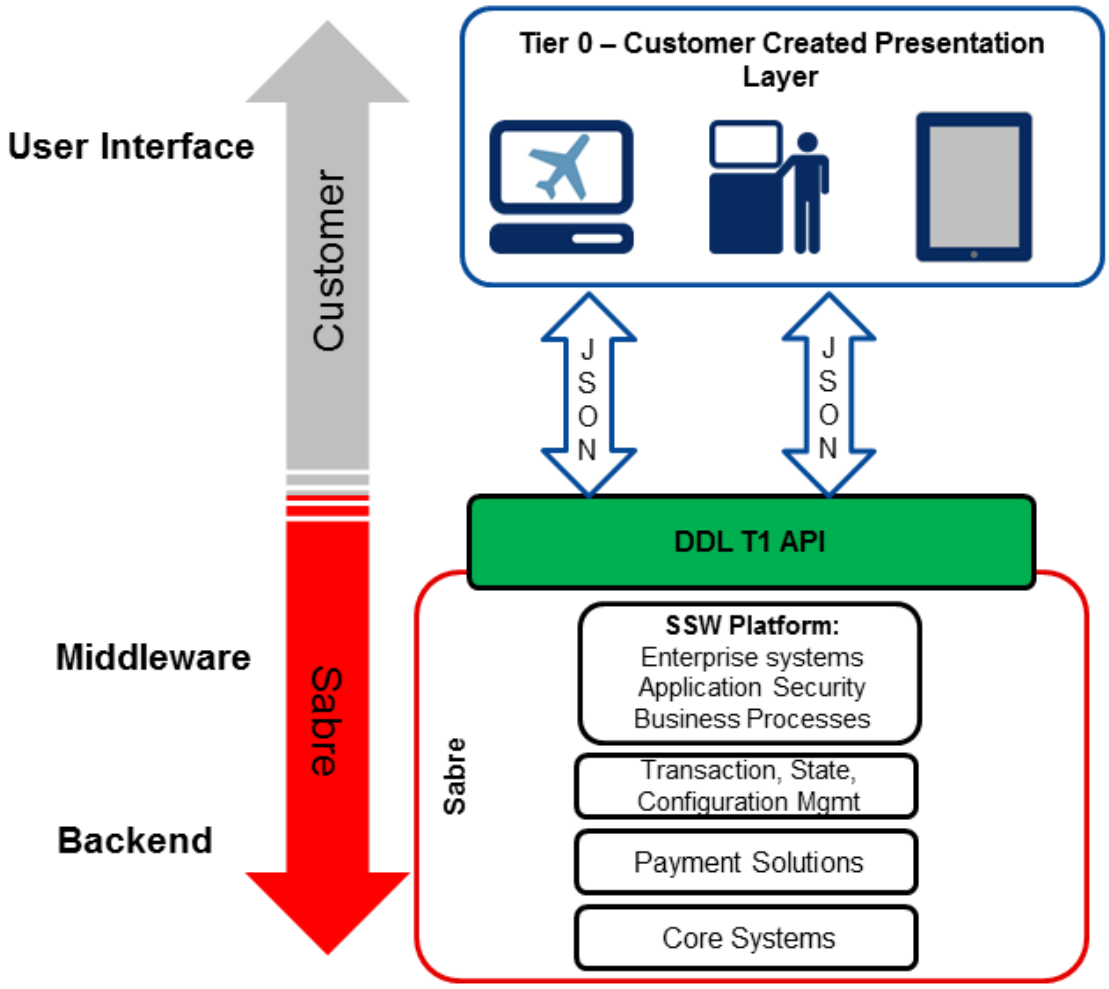
This chapter summarizes the goals of the Digital Connect (DC) service API and describes the technologies that support it.

2.1 Architecture

This section describes the current architecture for DC services.

Sabre technologies can be accessed in a Tier 1 layer (a web/presentation layer) that offers such capabilities as configuration management, session management, cache management, and so on. The goal of the Digital Connect project is to build a set of state-of-the-art services that allow for an entirely airline-created presentation layer, that enables airlines to integrate external content, cross-sell features, and build native applications that access these services and the existing Tier 1 business and technical capabilities.

The Digital Connect API is an API that allows access to Tier 1 services, as shown in the following diagram:



The architecture of the Digital Connect API is being guided by the following principles:

- *Uniform Interfaces* – the API provides an interface between client side and server side that simplifies and decouples the architecture, which enables each side to evolve independently.
- *Client/Server Relationship between DC Services and Their Users* – this makes it possible for both client-side and server-side actors to be developed and replaced independently, as long as the interface is not altered.
- *State Management on the Server Side* – state management enables greater flexibility since the server will be maintaining, updating, or communicating the session state.
- *A Layered System* – adherence to the preceding principles facilitates the use of intermediary hardware servers which further improve system scalability by enabling load-balancing and by providing shared caches. It also facilitates enforcement security policies.
- *Cacheable* – caching eliminates client/server interactions, further improving scalability and performance. (Not available in Version 1.0.)

2.2 Security Policy

This section describes the security policy intended for the DC Service API.

- Leverages current SE 2SG (Service Gateway) for security and throttling
  - Only accepts SSL/TLS connections
  - Authentication and Session Management
  - Web services use session-based authentication, by establishing a session token via a POST or by using an API key as a POST body argument or as a cookie
  - Authorization – Allows privileged actions on a resource

2.3 Sabre Contact

Please contact Joyce Schofield (Joyce.Schofield@sabre.com) for more information.

Accessing DC Services 3

To access DC services, your requests need to pass several HTML headers and a URL.

- The required headers are:
  - Authorization
  - Accept
  - Content-Type (for POST operations only)
- The URLs concatenate a base "api_context_path" that locates the overall DC API with a service URL that identifies a specific DC service.

This chapter covers the information required to successfully format these headers; it also describes how to format URLs to locate DC services.

Note that additional HTML headers are used to create and maintain DC user sessions. The session headers are covered in Chapter 4.

3.1 Obtaining an Authorization Token from 2SG

In order to use the Authorization header, you need to acquire an authorization token from 2SG. The procedure for this is outlined here:

1. Construct your Sabre client ID. This will be in the format:

   V1:{your_userid}:{your_group}:{your:domain}

2. Encode your client ID via Base64.
3. Use your encoded client ID to get an authorization token.
4. When accessing DC services, include the authorization token in the Authorization header.
5. OAuth 2.0 tokens have a finite lifetime of 604800 (seconds), so you will want to track this and refresh it when it becomes necessary.

For details of this procedure, see the following page in Sabre Dev Studio:
https://developer.sabre.com/docs/read/rest_basics/authentication

3.2 Using the Authorization Token in the Authorization Header

After obtaining the authentication token, it must be passed to 2SG in every call to a DC service via an HTML Authorization header. The format of the Authentication header is:

```
Authorization: Bearer <authentication_token>
```

3.3 Header Requirements for DC Services

The header requirements are slightly different for GET and POST operations.

3.3.1 For GET Operations

For GET operations, the following headers must be passed:

Header name	Header Value
Accept	application/json
Authorization	Bearer <*authentication_token*>

3.3.2 For POST Operations

For POST operations, the following headers must be passed:

Header name	Header Value
Accept	application/json
Content-Type	application/json
Authorization	Bearer <*authentication_token*>

3.4 DC Service URLs

Individual DC service URLs combine the "api_context_path" that locates the overall DC API and a service URL that locates a specific DC service. For example, the /search service is invoked by combining the DC API's URL:

https*://*<*host*>*:*<*port*>/<*api_context_path*>

With the service URL /search, resulting in:

https://<*host*>*:*<*port*>/<*api_context_path*>/products/air/search?jipcc=<*storefront*>

Notice that the storefront code is not part of URL but is passed as a request parameter. In practice, an actual URL will look something like the following example:

https://api.test.sabre.com/v1/ssw/products/air/search?jipcc=D5EE

Some other examples of DC service URLs are given in the following table:

Service Name	Service URL	Supported
Search	/products/air/search?jipcc=<*storefront*>	POST
Air (select Flights)	/products/air?jipcc=<*storefront*>	POST,GET
Passengers	/passengers?jipcc=<*storefront*>	POST,GET
Ancillaries	/products/ancillaries?jipcc=<*storefront*>	POST,GET
Seats	/products/seats?jipcc=<*storefront*>	POST,GET
Products	/products?jipcc=<*storefront*>	GET
Purchase	/purchase?jipcc=<*storefront*>	POST

User Session Management

4

DC services have been designed as a set of stateful, JSON-based services, which can be consumed directly from code running directly in a web browser.

Your first request will create a session and return session-identifying information. To maintain the session in subsequent requests, you simply pass the session-identifying information. Some of it is passed in your HTTP headers and some in URL parameters. This chapter provides details of obtaining and using the session-identifying data.

4.1 Session data

Session data consists of the following items.

Session Data Item	Description
JSESSIONID Cookie	A value generated by the initial flight search (/products/air/search) request, and included in the response to that request. This value is used by the application server. To maintain the session, pass this value in all subsequent requests. Example: Cookie: JSESSIONID=8A53D60D47E457ECF7DB5547AE001D03; WLPCOOKIE=sswhla123
WLPCOOKIE Cookie	A value generated by the initial flight search (/products/air/search) request, and included in the response to that request. This value is used by the load balancer. Example from an output header: Set-Cookie: WLPCOOKIE=sswhli4553; Expires=Tue, 22-Sep-2015 16:10:50 GMT; Secure; HttpOnly Note: The expiration date returned in the header is a recommendation. While it is possible to still use the cookie's value after the expiration date, it is recommended to not include this cookie in the user sessions that are created after the expiration date – this helps to ensure proper balancing of loads in the DC server farm.
SSWGID Cookie	A value generated by the initial flight search (/products/air/search) request, and included in the response to that request. This is a tracking cookie that supports DC's analytics capabilities. Example from an output header: Set-Cookie: SSWGID=8AFC582B7C0D4C5F9CBDCC4C0D41A62D; Expires=Mon, 17-Nov-2025 20:38:09 GMT; Path=/; Secure; HttpOnly
Execution ID	A value generated by the initial flight search (/products/air/search) request, and included in the response to that request. Its value is the execution key of the current conversation, and it should be included as URL parameter in all subsequent web service requests within the scope of the conversation. It is possible to open multiple conversations within an HTTP user session, for example when one passenger is creating multiple independent bookings in the session. Example of execution ID value: e1s1 Example of execution ID as a URL parameter in an HTTP request: https://<*someurl.com*>/api/products/air/search?execution=e1s1

4.2 Session lifecycle

The basics of the DC session lifecycle are listed below:

- User sessions are created implicitly when a request is sent without a JSESSIONID cookie or an execution key parameter in the URL.
- Sessions are not explicitly deleted, but are removed automatically after expiration due to inactivity for a predefined amount of time (configuration option sat.session.timeout - typically 20 minutes).
- Executing any service within the scope of a session (by passing session identifiers in a request) automatically resets the session expiration time.
- Execution of DC web services happens within a conversation. A conversation exists in a scope of a user session and is identified by the execution key. This approach allows – if there is such a need - to maintain multiple independent transactional flows for the same user, within a single user session. Use of this functionality is optional.

4.2.1 Session creation

In order to create a session, send a /products/air/search request without cookie headers.

HTTP request

The request should be similar to the following example.

```
POST https://<someurl.com>/<apicontextpath>/products/air/search
```

The request headers should contain the following:

```
Content-Type: application/json
Accept: */*
Accept-Encoding: gzip, deflate
```

HTTP response

The responses will contain the values that you can use to maintain the session, the execution ID, the JSESSIONID, the WLPCOOKIE, and the SSWGID.

```
Execution: e4s1
Access-Control-Expose-Headers: Execution
Access-Control-Allow-Headers: Execution
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS, HEAD, PUT, DELETE
Access-Control-Allow-Credentials: true
Vary: Accept-Encoding
Set-Cookie: JSESSIONID=9D7CB644D22ECAF84B7A5EA88F48FC2D; Path=/SSW2010/;
Secure; HttpOnly
Set-Cookie: WLPCOOKIE=sswhli4553; Expires=Tue, 22-Sep-2015 17:40:49 GMT;
Path=/SSW2010; Secure; HttpOnly
Content-Encoding: gzip
Content-Type: application/json
Content-Length: 2125
Date: Tue, 22 Sep 2015 15:33:11 GMT
```

In the example:

- The execution ID is e4s1
- The JSESSIONID is 9D7CB644D22ECAF84B7A5EA88F48FC2D
- The WLPCOOKIE is sswhli4553

The execution ID identifies the initial conversation within the session.

To maintain the session (or a conversation within the session), pass these values in subsequent requests.

4.2.2 Session use

To execute a request within the scope of an existing session and conversation, include the user conversation and session coordinate in the HTTP Cookie header(s) and the execution ID as the URL parameter “execution”:

```
<HTTP method name> https://someurl.com/<serviceName>?execution=<executionID>
Cookie: JSESSIONID=<JSESSIONIDvalue>; WLPCOOKIE=<WLPCOOKIEvalue>
```

The execution ID may be used in any service – including air shopping. In such case results of shopping call in the scope of the user conversation are overwritten.

The following example uses the session coordinates and conversation created in the preceding section.

HTTP request

The request includes the execution ID.

```
POST https://wl19-int.sabresonicweb.com/SSW2010/D5DE/api/products/air/
search?execution=e4s1
```

The request headers include the JSESSIONID and WLPCOOKIE cookie values.

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/45.0.2454.93 Safari/537.36
Origin: chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo
Content-Type: application/json
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=9D7CB644D22ECAF84B7A5EA88F48FC2D; WLPCOOKIE= sswhli4553
```

HTML response

The response includes the execution ID.

```
Execution: e4s1
Access-Control-Expose-Headers: Execution
Access-Control-Allow-Headers: Execution
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS, HEAD, PUT, DELETE
Access-Control-Allow-Credentials: true
```

```
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Type: application/json
Content-Length: 2125
Date: Tue, 22 Sep 2015 16:33:11 GMT
```

4.2.3 Conversation creation

To create a new conversation in the scope of an existing user session, send a request that supplies the user session coordinates in the HTTP Cookie header(s) but omits the execution ID URL parameter.

```
<HTTP method name> https://<someurl.com>/<serviceName>
Cookie: JSESSIONID=<JSESSIONIDvalue>; WLPCOOKIE=<WLPCOOKIEvalue>
```

The following example uses the session coordinates from the previous section.

HTTP request

The request omits the execution ID.

```
POST https://wl19-int.sabresonicweb.com/SSW2010/D5DE/api/products/air/search
```

As before, the request header contains the session coordinates.

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/45.0.2454.93 Safari/537.36
Origin: chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo
Content-Type: application/json
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=9D7CB644D22ECAF84B7A5EA88F48FC2D; WLPCOOKIE= sswhli4553
```

HTML response

The response contains the execution ID for the newly created conversation.

```
Execution: e5s1
Access-Control-Expose-Headers: Execution
Access-Control-Allow-Headers: Execution
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS, HEAD, PUT, DELETE
Access-Control-Allow-Credentials: true
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Type: application/json
Content-Length: 2125
Date: Tue, 22 Sep 2015 16:33:11 GMT
```

Error Reporting

5

This chapter covers the reporting of errors by DC services.

5.1 Message Formats

The format of a DC message is illustrated below.

```
{
  "status": "<status>",
  "type": "<type>",
  "errorCode": "<errorCode>"
  "timeStamp": "2015-12-05T21:59:11",
  "message": "<messageText>"
   "details" :
  {
      "<messageDetails>"
  }
}
```

The possible status codes are Complete, Incomplete, NotProcessed, and Unknown.

The message types are: Transport, Validation, Application, and BusinessLogic. Currently there are no messages of type Transport, which is used for 2SG errors.

5.2 Message Examples

An actual message, with values, would look like the following example. In this case it is a validation message, as indicated by its type, and the details identify the field that failed validation, "cabinClass," and the specific failure, "invalid.format."

```
{
  "status": "NotProcessed",
  "type": "Validation",
  "errorCode": "ERR.SSW.CLIENT.INVALID_REQUEST"
  "timeStamp": "2015-12-05T21:59:11",
  "message": "Validation error"
  "details" :
  {
      "cabinClass": [
           "invalid.format"
           ]
  }
}
```

If a request results in multiple validation failures, it is reported like the next example. This example shows multiple errors on multiple fields of the passenger details data. Each error shows the field and error detail. Note the use of the passenger index to identify which passenger in the booking each of the errors applies to.

```
{
  "status": "NotProcessed",
  "type": "Validation",
  "errorCode": "ERR.SSW.CLIENT.INVALID_REQUEST”
  "timeStamp": "2015-12-05T21:59:11",
  "message": "Validation error”
  “details” :
  {
    "passengers[0].passengerDetails.lastName": [
      "validation.text.tooLong"
    ],
    "passengers[0].passengerDetails.prefix": [
      "validation.error.not_valid"
    ],
    "passengers[0].passengerDetails.firstName": [
      "validation.text.tooLong"
     ],
    "passengers[1].passengerDetails.firstName": [
      "validation.text.tooLong"
     ]
  }
}
```

Any error that results from the passed value not matching an AllowedValue statement will end validation and return the error.

In the case of multiple errors, other than an AllowedValue error, validation continues until all fields have been validated and multiple errors are returned as shown above.

Note that the error code is typically not a message that can be displayed directly to passengers; the airline needs to substitute message text of its own for display.

Note that the formats for some basic data types are consistent across the DC services:

- Date should be in the format *YYYY-MM-DD.*

Other data formats, such as frequent flyer account number format(s) are defined per airline.

The DC Service Suite (Shopping Services)

6

This chapter lists the services that have been implemented for the DC services API for the DC 1.0 release. The following tables list the available services by their functional names. The next chapter presents a typical sequence for calling these services in order to display information to passengers, build an itinerary, and book the itinerary.

6.1 The Ancillaries Shopping (/products/ancillaries) Service

"Products," in the context of DC services, means air transportation, ancillaries, bags, and seats (including paid seats). Use the /products/ancillaries service to obtain a list of ancillary products that are available for any flights that the passenger has selected. Use the returned information to populate a display of available ancillary products for the passenger. Also use the POST operation to add any ancillary products selected by the passenger to the itinerary.

Note that itineraries built with the /products services are held in session until you use the /purchase service to book the itinerary and create a PNR. The /purchase service can only be used after the flights have been selected and added to the itinerary with the Select Flights (/products/air) service.

Service Name	Business Function	Operations
/products/ancillaries	Obtain a list of ancillary products available on the passenger-selected flights, and add any ancillaries selected by the passenger to the itinerary.	GET – returns a list an ancillary products available on flights that have been selected by the passenger. POST – adds any ancillary products selected by the passenger to the itinerary. Note that the itinerary remains in the session.

6.1.1 Customer Requirements

In order to use this service, the ancillaries must be defined in advance.

6.2 The Baggage Allowance (/products/air/bags) Service

"Products," in the context of DC services, means air transportation, ancillaries, bags, and seats (including paid seats). Use the /products/air/bags service to obtain information about bag allowances for any flights that the passenger has added to the current itinerary. Use the information to populate a display of bag allowance information for the passenger.

Note that itineraries built with the /products services are held in session until you use the /purchase service to book the itinerary and create a PNR. This service can only be used after the flights have been selected and added to the itinerary with the Select Flights (/products/air) service.

Service Name	Business Function	Operations
/products/air/bags	Obtain information about bag allowances on flights the passenger has selected.	GET – returns information about bag allowances for flights in the current itinerary; includes the number of free bags allowed on the selected flights, the number of carry-ons allowed, and the availability of additional paid bags and carry-ons.

6.2.1 Customer Requirements

Bag allowances must be set up in advance.

6.3 The Fare Rules (/products/air/farerules) Service

This service allows an airline to obtain the fare rules for flights that have been selected by a customer and added to the itinerary (via the /products/air service). The airline can use this information to populate a display of fare rules.

Service Name	Business Function	Operations
/products/air/farerules	To display the fare rules that apply to flights selected by the passenger and held in the session.	GET - Returns information suitable for populating a fare rules display.

6.4 The Flight Search/Flight Shopping (/products/air/search) Service

Use the /products/air/search service to obtain lists of available flights that match search criteria supplied by passengers and display those flights to passengers for their selection. Also, use the POST operation to build an itinerary based on passenger selections.

Itineraries built with these services are held in session until you use the /purchase service to book the itinerary and create a PNR.

Service Name	Business Function	Operations
/products/air/search	Obtains a list of air transport products (available flights) that meet the passenger's search criteria.	POST –performs a search of flights using the search criteria supplied in the parameters; the response contains a list of matching flights with fare offerings and detailed price breakdowns GET /products/air/search/latest

6.5 The Frequent Flyer Login and Profile Display (/login) Service

The /login service enables an airline to give passengers access to their accounts and display their account information. The airline prompts passengers for their user IDs and passwords and then uses the service to log the passenger in and display account details.

Service Name	Business Function	Operations
/login	Allows passenger to log in to their frequent flyer accounts and view current account information.	POST –submits passenger frequent flyer ID and password for authentication. GET – returns current account information for any authenticated passenger.

6.6 The Passenger Information (/passengers) Service

Use this service to add and modify passenger information associated with the current itinerary. Note that the current itinerary is the itinerary in the current session.

Service Name	Business Function	Operations
/passengers	List and add passengers associated with an itinerary; modify passenger details. To add passengers to an itinerary, use the POST operation, supplying details for every passenger in the itinerary. To update passenger data, use the POST operation, supplying details for every passenger in the itinerary, including the modified information.	GET – returns current passenger information for the current itinerary. POST – adds passenger information to the current itinerary.

6.6.1 Customer Requirements

An itinerary must have been created, via the Select Flights (/products/air) service, before any passenger information can be added.

6.7 The Payment Options (/paymentOptions) Service

The airline uses the /paymentOptions service to determine which forms of payment are applicable to the products the passenger has selected and added to the itinerary, and to obtain payment combinability information.

Service Name	Business Function	Operations
/paymentOptions	Obtains a list of payment options available for the products in the current itinerary. Adds information about the payment options selected by the passenger.	GET -Returns combinability mapping. This provides information the airline can use to populate a display of allowable payment methods and combinable forms of payment in a multiple forms of payment situation. It also provides information which products can be paid separately. POST – Maps payment types to products. Will return any leftover amount with possible payment methods. Use this operation to record the payment types the passenger has selected for products in the itinerary.

6.8 The Purchase Itinerary and Create PNR (/purchase) Service

The airline uses the /purchase service to actually book the flights and purchase any other products identified in the itinerary. Obtain the passenger's payment details and then use this service to authorize the payment, and, after authorization, issue a PNR and any other needed documents.

Service Name	Business Function	Operations
/purchase	Books the flights identified in the itinerary and purchases any other products selected by the passenger. Authorizes the passenger's payment and issues a PNR and any other needed documents. Note that the itinerary built with the /products services is held in session until this service is called.	POST – submits payment details provided by the passenger for authorization, and if authorization is successful, issues a PNR and any other needed documents.

6.9 The Retrieve PNR Details (/pnr) Service

The /pnr service retrieves PNR information for an existing booking. Allows the airline to prompt a passenger for PNR identifiers and then display the corresponding PNR to the passenger.

Service Name	Business Function	Operations
/pnr	Obtain PNR details for an existing booking.	GET – returns PNR details for a booking, which can be displayed to the passenger. (Booking can be identified by PNR number, or by some combination of passenger first name, passenger last name, and passenger email address. Which of these are required can be configured.)

6.9.1 Customer Requirements

- This service must be enabled before it can be used.
- An itinerary must have been purchased and a PNR created before this service can be used.

6.10 The Seatmap and Seat Selection (/products/seats) Service

Use the /products/seats service to obtain information about seats that are available for any flights that the passenger has selected. Use the information to populate a seatmap for the passenger. Also use the POST operation to add any seats the passenger selects to the itinerary.

This service can only be used after the one or more flights have been selected and added to the itinerary with the Select Flights (/products/air) service.

Itineraries built with these services are held in session until you use the /purchase service to book the itinerary and create a PNR.

Service Name	Business Function	Operations
/products/seats	Obtain a list of seat products available on the passenger-selected flights, and add any seats selected by the passenger to the itinerary.	GET – returns information used to populate a seatmap for display to the passenger POST – adds any seats selected by the passenger to the itinerary. Note that the itinerary remains in the session until the /purchase service is used to book the itinerary.

6.10.1 Customer Requirements

Seatmap information must be set up in advance.

6.11 The Select Flights Service (/products/air) Service

Use the/products/air service to add air transportation products to an itinerary (this includes adding the first flight and creating the itinerary). Also, use this service to retrieve the current state of the itinerary from the session.

Note that itineraries built with these services are held in session until you use the /purchase service to book the itinerary and create a PNR.

Service Name	Business Function	Operations
/products/air	Add any flights selected by the passenger to the itinerary; obtains a list of flights already selected.	POST – adds one or more flights selected by the passenger to the session; the first step in building an itinerary. GET – returns a list of flights that have been selected by the passenger.

6.12 The Shopping Cart (/products/cart) Service

Use the /products/cart service to obtain a list of all products that have been selected by passengers and a price breakdown for all of the selected products. Use this information to build a shopping cart display.

Note that itineraries built with the /products services are held in session until you use the /purchase service to book the itinerary and create a PNR.

Service Name	Business Function	Operations
/products/cart	Gets a price breakdown of all products already selected by the passenger and held in the session; use this information to format and display a shopping cart. The airline can call this service and format a shopping cart on each page displayed to the passenger.	GET – returns a list of all products that have been selected by the passenger with a price breakdown; suitable for populating a shopping cart display.

6.13 The Text Translations (/translations) Service

The Translations service returns all of the translations for the specified language.

Service Name	Business Function	Operations
/translations	Obtains all of the translation text for a specified language. The airline can use the translation texts to populate various displays for passengers. For example, the list of translations will include the text that maps equipment-type codes to aircraft names for display to passengers.	GET – returns a list of all translation text for a specified language.

6.13.1 Customer Requirements

Translation keys must be set up in advance.

6.14 The Upgrade (/products/upgrade) Service

The /products/upgrade service returns information about cabin classes higher than those originally selected by the passenger in current itinerary. The airline can format this information and display it to the passenger in order to offer the passenger the opportunity to move to a higher cabin class. The airline can set a "threshold" amount, which limits the fare increase that will be included in the upgrade offers.

Note that if the passenger has already selected the highest cabin class available on the flights, there will be no available upgrade information.

Service Name	Business Function	Operations
/products/upgrade	Obtains a list of higher cabin classes for display to the passenger, including the fare difference. Also modifies the itinerary if the passenger accepts an upgrade offer.	GET – returns a list higher cabin classes available to the passenger on the flights in the current itinerary. POST – if the passenger accepts an upgrade offer, updates the itinerary with the newly selected cabin class.

6.14.1 Customer Requirements

- The airline must be using low fare shopping.
- The upgrade service must be enabled.

Using the DC Service Suite 7

This chapter outlines the basic sequence of events for using the DC 1.0 services. To make use of the services and book flights, use the services in the following order:

1. Search for Flights (required):

   Obtain search criteria from the passenger via the client and submit a /products/air/search request, POST operation to get a list of flights matching the search criteria. Display the returned flights to the passenger.

   This request will create a session and a conversation. From the response, collect the execution ID, JSESSIONID cookie, WLPCOOKIE cookie, and SSWGID cookie, which you will use in subsequent requests to maintain the session. For more information on creating and using sessions and conversations, see Chapter 4, User Session Management.
   The response will also include a ShoppingBasketHashCode for each flight.

2. Select Flight (required):

   After the passenger selects one or more flights via the client, use the /products/air POST operation to add the flights to the itinerary and store the itinerary in the session.

   The request must include the header data, execution ID, JSESSIONID cookie, WLPCOOKIE cookie and SSWGID cookie from the search response to maintain the session. Use the hash codes for the passenger-selected flights to identify the flights in the POST operation. Use the GET operation to retrieve the itinerary, or additional POST operations to update the itinerary. Omitting the execution ID from these operations will lead to an error.

3. Shopping cart (optional):

   At any time after an itinerary has been stored in the session, use a /products GET operation to obtain a current list, with prices, of products that the passenger has added to the itinerary. Use this information to build and display a shopping cart. You will probably build and display a shopping cart on every page of your user interface.

   The request must include the header data, execution ID, JSESSIONID cookie, WLPCOOKIE cookie and SSWGID cookie from the original search response to maintain the session. Note that you can submit a /products GET request at any time after the first /products/air POST request has created an itinerary in the user session.

4. Add Passenger Details (required):

   After an itinerary has been stored in the session, use /passengers operations to add passengers to the itinerary. Requests must include the header data, execution ID, JSESSIONID cookie, WLPCOOKIE cookie and SSWGID cookie from the original search response to maintain the session.

   - Submit a /passengers POST request to add passengers to an itinerary. You can also use POST to update, by submitting all of the passenger data including the updated fields, and to delete passengers from the itinerary, by submitting all of the passenger data, including blank fields for the deleted passengers.
   - Submit a /passengers GET request to display the current passenger data for the itinerary.

5. Add Ancillaries (optional):

   After an itinerary has been stored in the session, use /products/ancillaries operations to show passengers the ancillaries available on the flights they have selected and to add any ancillaries selected by the passenger to the itinerary. Requests must include the header data, execution ID, JSESSIONID cookie, WLPCOOKIE cookie and SSWGID from the original search response to maintain the session.

   - Submit a /products/ancillaries GET request to obtain a list of ancillaries available for the flights in the itinerary. You can display these ancillaries to the passenger for selection.
   - Submit a /products/ancillaries POST request to add ancillaries to the itinerary. You can also use POST to update, by submitting all of the ancillary data including the updated fields, and to delete ancillaries from the itinerary, by submitting all of the ancillary data, including blank fields for the deleted ancillaries.

6. Add Seats (optional):

   After an itinerary has been stored in the session, use /products/seats operations to show passengers the seats available on the flights they have selected and to add any seats selected by the passenger to the itinerary. Requests must include the header data, execution ID, JSESSIONID cookie, WLPCOOKIE cookie and SSWGID from the original search request to maintain the session.

   - Submit a /products/seats GET request to the information used to construct seat maps for the flights in the itinerary. The seat information will indicate available seats and reserved seats, free seats and paid seats, and so on. You can use this information to format a seat map for display to the passenger for selection.
   - Submit a /products/seats POST request to add ancillaries to the itinerary. You can also use POST to update, by submitting all of the seat data including the updated fields, and to delete seats from the itinerary, by submitting all of the seat data, including blank fields for the deleted seats.

7. Payment Options (required):

   After the passenger has completed building the itinerary in the session, including ancillaries and seats use /paymentOptions to retrieve available FOPs .The requests must include the header data, execution ID, JSESSIONID cookie, WLPCOOKIE cookie and SSWGID from the original search request to maintain the session.

   - Submit a /paymentOptions GET request to obtain a list of available forms of payment (FOP) and amounts (surcharges included). You can display these forms of payment to the passenger for selection.
   - Submit a /paymentOptions POST request to send FOP code with amount to get in the response a list with all combinable forms of payment with amount (residual amount).

8. Purchase (required):

   After the passenger has completed building the itinerary in the session, including ancillaries and seats, collect payment data from the passenger and submit a /purchase request. The request must include the header data, execution ID, JSESSIONID cookie, WLPCOOKIE cookie and SSWGID from the original search response to maintain the session.

The /purchase request will submit billing data to the FOPs supplied by the passenger and, if payment is authorized, create a PNR. For details of the processing logic, and errors that may occur during this processing, see Chapter 9.

Products Services

8

This group of services enables an airline to:

- Obtain lists of available products for display to passengers
- Add passenger-selected products to the passenger's itinerary, which is stored in the session.

8.1 /products Service

This service returns a detailed price breakdown of all products and passenger details in the itinerary that is currently held in the session. Use this information to populate a shopping cart display in the client.

8.1.1 GET Operation

This operation returns a detailed price breakdown of all products in the current itinerary.

Request URL Syntax

```
http://<host>:<port>/<apiContextPath>/products/total?jipcc=<storefront>
```

Request CURL Syntax

```
curl -X GET --header "Accept: application/json"
"http://<host>:<port>/<apiContextPath>/products/total?jipcc=<storefront>"
```

Timeout and Retry

For this operation, the timeout is 240 seconds. The timeout interval allows for complete processing of the request, including processing performed by downline systems. Retries are allowed, but keep in mind that retrying may change reservation state.

Response Model

```
BreakdownResponse {
  itineraryInformation (ItineraryInformation, optional): Itinerary for
    the current booking. Used as a reference.,
  total (ApiPrice, optional): Total for all products contained in
    breakdown.,
  productBreakdowns (Array[ProductBreakdown], optional): List of per
    product breakdowns.
}
ItineraryInformation {
  itinerary (Itinerary, optional): Selected itinerary and flight details,
  promoCodesUsed (Array[string], optional): Promo codes used to obtain
    current offers.,
  offersSelected (Array[OfferWithoutPrice], optional): Selected offers
    from search results
}
ApiPrice {
```

```
  alternatives (Array[Inline Model 1], optional): List of price
    alternative lists. Each element of main list is a full price. e.g.
    list [[700 AUD],[300 AUD, 80000 FFCURRENCY]
}
ProductBreakdown {
  label (string, optional): Product label like 'air', 'ancillaries',
    etc.,
  breakdownElements (Array[BreakdownElement], optional): Breakdown for
    this products price.,
  total (ApiPrice, optional): Total for this product breakdown.
}
Itinerary {
  origin (string, optional),
  departure (string, optional),
  arrival (string, optional),
  destination (string, optional),
  type (string, optional) = ['SEGMENT', 'ITINERARY_PART', 'ITINERARY'],
  itineraryParts (Array[ItineraryPart], optional): List of itinerary
    parts (aka legs).
}
OfferWithoutPrice {
  shoppingBasketHashCode (integer, optional): Offer identifier. Might not
    be unique. Should be consistent between searches.,
  brandId (string, optional): Brand associated with this offer.,
  seatsRemaining (SeatsRemaining, optional): Information about seats
    remaining for this offer.,
  itineraryPart (Array[ItineraryPart], optional): Offered itinerary
    parts.
}
Inline Model 1 [
  Amount
]
BreakdownElement {
  label (string, optional): Label for this breakdown element. In a form
    of a machine readable/parseable code. Defined per product type. So
    it can be used for user translation display.,
  price (ApiPrice, optional): Price for this element or total for all
    subelements if this is not a leaf.,
  breakdownElementAssignment (BreakdownElementAssignment, optional):
    Additiona information about what this element is assigned to.,
  subElements (Array[BreakdownElement], optional): List of all breakdown
    pieces for this element. e.g. if this element is total for taxes
    'subElements' will contain a list of specific taxes with
    corresponding detailed prices.
}
ItineraryPart {
  segments (Array[Segment], optional): List of itinerary part segments.,
  stops (integer, optional): Number of stops,
  totalDuration (integer, optional): Total flight duration in minutes
```

```
}
SeatsRemaining {
  count (integer, optional): Number of remaining seats.,
  lowAvailability (boolean, optional): Indicator if current seat count is
    considered low. This is just a guideline for frontend.
}
Amount {
  amount (number, optional): Amount.,
  currency (string, optional): Currency. Three letter ISO code or
    'FFCURRENCY' for points/miles.
}
BreakdownElementAssignment {
  travelPart (TravelPart, optional): Assigned travel part.,
  passenger (Passenger, optional): Specifies to whom this breakdown
    element is assigned. If passenger is specify passengerType has no
    meaning.,
  passengerType (string, optional): Specifies to which passengerType this
    breakdown element is assigned. Total should be for all passengers
    with this type.,
  quantity (integer, optional): Quantity of the same values existing in
    this assignment e.g. number of passengers of type for air fare,
    amount of ancillaries with specific code etc.
}
Array[BreakdownElement] {
}
Segment {
  segmentOfferInformation (SegmentOfferInformation, optional): Additional
    segment information.,
  duration (integer, optional): Flight duration in minutes,
  cabinClass (string, optional) = ['Economy', 'Business', 'First'],
  equipment (string, optional): Aircraft equipement code.,
  aircraftLeaseText (string, optional): Aircraft lease disclosure text.,
  flight (Flight, optional): Flight details of this segment.,
  origin (string, optional): IATA airport code of origin.,
  destination (string, optional): IATA airport code of destination.,
  departure (string, optional): Departure date time.,
  arrival (string, optional): Arrival date time.
}
TravelPart {
  origin (string, optional),
  departure (string, optional),
  arrival (string, optional),
  destination (string, optional),
  type (string, optional) = ['SEGMENT', 'ITINERARY_PART', 'ITINERARY'],
  itineraryParts (Array[ItineraryPart], optional): List of itinerary
    parts (aka legs).
}
Passenger {
  passengerDetails (PassengerDetails, optional): Passenger Details,
```

```
  passengerInfo (PassengerInfo, optional): Passenger Information,
  preferences (Preferences, optional): Preferences like Meal/Seat/Special
    Requests,
  documentInfo (PassengerDocumentInfo, optional): Passenger Document
    Information,
  visaInfo (PassengerVisaInfo, optional): Passenger Visa Information
}
SegmentOfferInformation {
  flightsMiles (integer, optional): Flight miles,
  flightAncillaries (FlightAncillaries, optional): List of all
    ancillaries available to flights in this segment.,
  awardFare (boolean, optional): Flag indicating if this segment is an
    award fare.
}
Flight {
  flightNumber (integer, optional): Flight number.,
  airlineCode (string, optional): Marketing airline IATA code.,
  operatingAirlineCode (string, optional): Operating airline IATA code.,
  stopAirports (Array[string], optional): Stop airports associated with
    this flight.
}
PassengerDetails {
  firstName (string, optional): Passenger first name,
  middleName (string, optional): Passenger middle name,
  maidenName (string, optional): Passenger maiden name,
  lastName (string, optional): Passenger last name,
  prefix (string, optional): Passenger prefix,
  suffix (string, optional): Passenger suffix
}
PassengerInfo {
  dateOfBirth (string, optional): Passenger date of birth,
  gender (string, optional): Passenger gender like Male/Female
    = ['Male', 'Female'],
  redressNumber (string, optional): Redress number is issued to customer
    as part the TSA's Secure Flight program to speed identification and
    security clearance. more information available at
    www.tsa.gov/stakeholders/secure-flight-program,
  knownTravelerNumber (string, optional): Known Travler/TSA PreCheck
    Number,
  type (string, optional):
  Passenger type Adult/Child = ['ADT', 'YCB', 'YTH', 'CHD', 'C04', 'C11',
    'INF', 'INS', 'STU', 'MIL', 'SRC', 'CMA', 'CMP'],
  emails (Array[string], optional): List of passenger emails,
  phones (Array[Phone], optional): List of passenger phones,
  address (Address, optional): Passenger address
}
Preferences {
  specialPreferences (SpecialPreferences, optional): Passenger
    specialPreferences like meal, seat and special request,
```

```
  frequentFlyer (FrequentFlyer, optional): Passenger frequentflyer
    account details
}
PassengerDocumentInfo {
  documentNumber (string, optional): passenger document number,
  issuingCountry (string, optional): document issued ISO country code
    like : https://en.wikipedia.org/?title=ISO_3166-1,
  countryOfBirth (string, optional): passenger countryOfBirth ISO country
    code like : https://en.wikipedia.org/?title=ISO_3166-1,
  issueDate (string, optional): document issued date,
  documentType (string, optional): document type = ['P', 'A', 'C', 'F',
    'M', 'N', 'T', 'V', 'I'],
  nationality (string, optional): passenger nationality ISO country code
    like : https://en.wikipedia.org/?title=ISO_3166-1,
  expirationDate (string, optional): document expiration date
}
PassengerVisaInfo {
  documentNumber (string, optional): passenger document number,
  issuingCountry (string, optional): document issued ISO country code
    like : https://en.wikipedia.org/?title=ISO_3166-1,
  countryOfBirth (string, optional): passenger countryOfBirth ISO country
    code like : https://en.wikipedia.org/?title=ISO_3166-1,
  issueDate (string, optional): document issued date,
  issuePlace (string, optional): document issued place
}
FlightAncillaries {
  flightAncillary (Array[FlightAncillary], optional)
}
Phone {
  type (string, optional): Phone type = ['WORK', 'HOME', 'MOBILE', 'FAX',
    'CELLULAR', 'AGENCY', 'BUSINESS'],
   countryCode (string, optional): Phone country code like
     https://en.wikipedia.org/wiki/Country_code,
   areaCode (string, optional): Phone area code,
   number (string, optional): Phone number,
   extension (string, optional): Phone extension
}
Address {
  addressType (string, optional): Address type = ['HOME', 'INVOICE',
    'BUSINESS', 'OTHER', 'RESIDENCE', 'AGENCY'],
  street1 (string, optional): First line of street address,
  street2 (string, optional): Second line of street address like apt,
    suite...,
  postcode (string, optional): Address postal code,
  state (string, optional): Name of the state,
  city (string, optional): Name of the city,
  country (string, optional): ISO country code like:
    https://en.wikipedia.org/?title=ISO_3166-1
```

```
}
SpecialPreferences {
  mealPreference (string, optional): Choice of Meal = ['BBML', 'CHML',
    'DBML', 'FPML', 'GFML', 'HNML', 'KSML', 'LCML', 'LFML', 'LSML',
    'MOML', 'NLML', 'RVML', 'SFML', 'SPML', 'VGML', 'VJML', 'VLML',
    'VOML'],
  seatPreference (string, optional): Seat preference = ['NSSA', 'NSSB',
    'NSSR', 'NSST', 'NSSW'],
  specialRequests (Array[string], optional): SpecialRequests selected by
    passenger
}
FrequentFlyer {
  airline (string, optional): Frequentflyer account airline code (IATA
    code),
  number (string, optional): Frequentflyer number,
  tierLevel (string, optional): Frequentflyer tier Level
}
FlightAncillary {
  id (string, optional),
  subCode (string, optional): ATPCO sub code for ancillary.
}
```

Response Content Type

application/JSON

Field Information

Tax Codes

TaxCode	Description
AA	Airport Departure Tax Dominican Republic
AB	Airport Tax
AC	Value Added Tax Malta
AD	Airport Tax
AE	Passenger Service Charge (Intl) United Arab Emirates
AF	Airport Departure Fee Afghanistan
AG	Ticket Tax Antigua and Barbuda
…	…
ZM	Sales Tax Zambia
ZN	Passenger Charge Norway
ZO	Passenger Service Charge Greenland
ZP	Flight Segment Tax United States of America
ZQ	Airport Facility Charge Chile

ZR	Airport Tax

Errors

The errors that may be returned by this operation are listed below:

HTTP Code	Error Code	Message Code or Text	Type	Cause
400	ERR.SSW.CLIENT.INVALID_REQUEST	conversation '*<id>*' was completed or doesn't exist	Application	Service was called with a non-existent or expired execution Id, or the session has expired. Note that once the /purchase service is called the session and conversation (executionId) end.
500	ERR.SSW.INTERNAL_ERROR	*<message>*	Application	The service is unable to complete the request, due to a failure of a downline system.
503	ERR.SSW.INTERNAL.SERVICE_UNAVAILABLE	Service is disabled in configuration.	Application	Service is disabled in configuration.

8.2 /products/air Service

This service enables an airline to work with the list of air transport products that have been added to an itinerary and are held in the session.

8.2.1 GET operation

This operation returns a list of flight segments that have been selected by the passenger and added to the itinerary.

Request URL Syntax

```
http://<host>:<port>/<apiContextPath>/products/air?jipcc=<storefront>&selectFlights={99999}
```

Request CURL Syntax

```
curl -X POST --header "Content-Type: application/json" --header "Accept: application/json"
"http://<host>:<port>/<apiContextPath>/products/air?jipcc=<storefront>&selectFlights={99999}"
```

Timeout and Retry

For this operation, the timeout is 240 seconds. The timeout interval allows for complete processing of the request, including processing performed by downline systems. Retries are allowed, but keep in mind that retrying may change reservation state.

Response Model

```
Inline Model [
  ItineraryPart
]
ItineraryPart {
  segments (Array[Segment], optional): List of itinerary part segments.,
  stops (integer, optional): Number of stops,
  totalDuration (integer, optional): Total flight duration in minutes
}
Segment {
  segmentOfferInformation (SegmentOfferInformation, optional): Additional
        segment information.,
  duration (integer, optional): Flight duration in minutes,
  cabinClass (string, optional) = ['Economy', 'Business', 'First'],
  equipment (string, optional): Aircraft equipement code.,
  aircraftLeaseText (string, optional): Aircraft lease disclosure text.,
  flight (Flight, optional): Flight details of this segment.,
  origin (string, optional): IATA airport code of origin.,
  destination (string, optional): IATA airport code of destination.,
  departure (string, optional): Departure date time.,
  arrival (string, optional): Arrival date time.
}
SegmentOfferInformation {
  flightsMiles (integer, optional): Flight miles,
  flightAncillaries (FlightAncillaries, optional): List of all
        ancillaries available to flights in this segment.,
  awardFare (boolean, optional): Flag indicating if this segment is
        an award fare.
}
Flight {
  flightNumber (integer, optional): Flight number.,
  airlineCode (string, optional): Marketing airline IATA code.,
  operatingAirlineCode (string, optional): Operating airline IATA code.,
  stopAirports (Array[string], optional): Stop airports associated with
        this flight.
}
FlightAncillaries {
  flightAncillary (Array[FlightAncillary], optional)
}
FlightAncillary {
  id (string, optional),
  subCode (string, optional): ATPCO sub code for ancillary.
}
```

Response Content Type

application/JSON

Errors

The errors that may be returned by this operation are listed below:

HTTP Code	Error Code	Message Code or Text	Type	Cause
400	ERR.SSW.CLIENT.INVALID_REQUEST	conversation '*<id>*' was completed or doesn't exist	Application	Service was called with a non-existent or expired execution Id, or the session has expired. Note that once the /purchase service is called the session and conversation (executionId) end.
500	ERR.SSW.INTERNAL_ERROR	*<message>*	Application	The service is unable to complete the request, due to a failure of a downline system.
503	ERR.SSW.INTERNAL.SERVICE_UNAVAILABLE	Service is disabled in configuration.	Application	Service is disabled in configuration.

8.2.2 POST Operation

Adds one or more flights selected by the passenger (via the client) to the itinerary held in the session.

Request URL Syntax

```
http://<host>:<port>/<apiContextPath>/products/air?jipcc=<storefront>&selectFlights={99999}
```

Request CURL Syntax

```
curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" –d “body”
"http://<host>:<port>/<apiContextPath>/products/air?jipcc=<storefront>&selectFlights=<99999>"
```

Parameters

The parameters passed in this operation are:

Parameter Name	Parameter Type	Data Type	Definition
selectFlights	query	Array[integer]	Array of flight ID reference numbers (shopping basket hash codes)
Body	body	String	Pass a null value for this operation.

Parameter Model

```
{
}
```

Field Information

Field Group	Field Name	Required?	Notes
	selectFlights	Y	One value should be passed for each selected flight; the value should be the shoppingBasketHashCode from /products/air/search response. If the passenger selects more than one flight, pass a value for each selected flight. For a round trip, for example, the parameter value would look like this: selectFlights=XXXXX&selectFlights=XXXXXX Validation is applied to this field. The validation is not configurable. The validation is: • The number of select Flights passed must match the number of flights in the itinerary.

Timeout and Retry

For this operation, the timeout is 240 seconds. The timeout interval allows for complete processing of the request, including processing performed by downline systems. Retries are allowed, but keep in mind that retrying may change reservation state.

Response Model

The following response indicates a successful POST operation.

```
{} – HTTP 200 OK
```

Response Content Type

application/JSON

Errors

The errors that may be returned by this operation are listed below:

HTTP Code	Error Code	Message Code or Text	Type	Cause
400	ERR.SSW.CLIENT.INVALID_REQUEST	conversation '*<id>*' was completed or doesn't exist	Application	Service was called with a non-existent or expired execution Id, or the session has expired. Note that once the /purchase service is called the session and conversation (executionId) end.
400	ERR.SSW.PRICING	Error during reprice of types <*types*>	BusinessLogic	An error occurred during the pricing of the selected flight(s).
400	ERR.SSW.APP.BUSINESS_ERROR	<*message*>	BusinessLogic	An error possibly connected to reaching the pricing quota in Web Abuse IP blocking mechanism.
400	ERR.SSW.CLIENT.INVALID_REQUEST	Validation error.	Validation	The value supplied is invalid. Please see the details.
500	ERR.SSW.APP.UNKNOWN	<*message*>	Application	An unknown error occurred during selecting flights.
503	ERR.SSW.INTERNAL.SERVICE_UNAVAILABLE	Service is disabled in configuration.	Application	Service is disabled in configuration.

8.3 /products/air/bags Service

This operation returns a list of bag, carryon, and embargo information available on the flights the passenger has selected (the itinerary that is in the session). The information includes the number of free bags provided on the selected flights, the number of carry-ons, and information about the availability of additional paid bags and carry-ons.

Airlines can use this information to display bags information to the passenger via the client.

8.3.1 GET Operation

Request URL Syntax

```
http://<host>:<port>/<apiContextPath>/bags?jipcc=<storefront>
```

Request CURL Syntax

```
curl -X POST --header "Content-Type: application/json" --header "Accept:
application/json" -d "body"
"http://<host>:<port>/<apiContextPath>/bags?jipcc=<storefront>"
```

Response Model

```
BaggageDisclosure {
  segmentDisclosure (Array[SegmentDisclosure], optional): List of baggage,
    carryon and embargo information per segment.
}
SegmentDisclosure {
  passengerBaggageDisclosure (Array[PassengerBaggageDisclosure], optional):
    Baggage/CarryOn Allowance Information for each Passenger type. ,
  embargoInformation (EmbargoInformation, optional): Embargo restrictions for
    this segment. ,
  segment (SegmentKey, optional): Segment origin and destination code.
}
PassengerBaggageDisclosure {
  checkedInBaggage (CheckedInBaggage, optional): Bags allowed and price
    information for this segment. ,
  carryOnBaggage (CarryOnBaggage, optional): CarryOns allowed and price
    information for this segment. ,
  passengerType (string, optional): Passenger Type like Adult, Child etc., =
    ['ADT', 'YCB', 'YTH', 'CHD', 'C04', 'C11', 'INF', 'INS', 'STU', 'MIL',
    'SRC', 'CMA', 'CMP']
}
EmbargoInformation {
  airlineIataCode (string, optional): Airline IataCode like
    http://www.iata.org/about/members/pages/airline-list.aspx?All=true. ,
  restrictionList (Array[string], optional): List of embargo restrictions.
}
SegmentKey {
  origin (string, optional): IATA airport code of origin. ,
  destination (string, optional): IATA airport code of destination. ,
  departure (string, optional): Departure date time. ,
  arrival (string, optional): Arrival date time.
}
CheckedInBaggage {
  freeBaggageAllowance (FreeBaggageAllowance, optional): Free Checked In
    Baggage Allowance size and weight information. ,
  excessBaggage (Array[ExcessBaggage], optional): List of Excess baggage
    price, size and weight information.
}
CarryOnBaggage {
  freeBaggageAllowance (FreeBaggageAllowance, optional): Free CarryOn
    allowance name, size and weight information. ,
  excessBaggage (Array[ExcessBaggage], optional): List of Excess baggage
    price, size and weight information.
}
```

```
FreeBaggageAllowance {
  totalUnits (string, optional): Free Baggage Information Disclosure. ,
  totalWeight (string, optional): Total Weight allowed on Flight. ,
  baggageRestrictions (Array[BaggageRestriction], optional): Baggage Name,
     Number Of Units, Size and Weight Restrictions.
}
ExcessBaggage {
  bagPrice (ApiPrice, optional): Price of the bag/carryOn. ,
  baggageRestriction (BaggageRestriction, optional): Baggage Size and Weight
    descriptions ,
  allowedStatus (string, optional): Returns whether Baggage is allowed for
     free, permitted or not = ['FREE', 'NOT_PERMITTED', 'UNKNOWN']
}
BaggageRestriction {
  dimensionDescription (string, optional): Measurment of bag/carryOn in linear
    inches/centimeters with detail description. for eg., UP TO 62 LINEAR
    INCHES/158 LINEAR CENTIMETERS ,
  weightDescription (string, optional): Weight of the bag/carryOn in
    Kilograms/Pounds with detail description. for eg., UP TO 50 POUNDS/23
    KILOGRAMS ,
  allowedUnits (integer, optional): Number of bags/carryOns allowed. ,
  baggageName (string, optional): Baggage like Musical Instrument etc.,.
}
ApiPrice {
  alternatives (Array[Inline Model 1], optional): List of price alternative
    lists. Each element of main list is a full price. e.g. list [[700
    AUD],[300 AUD, 80000 FFCURRENCY]
}
Inline Model 1 [
  Amount
]
Amount {
  amount (number, optional): Amount. ,
  currency (string, optional): Currency. Three letter ISO code or 'FFCURRENCY'
    for points/miles.
}
```

Processing Logic

You must call /products/air/search and /products/air before calling this service.

1. This service checks whether a user has successfully selected an itinerary.
   a. If yes, then returns the information about baggage rules for the flights in the itinerary.
   b. If no, an error is returned.
2. This service returns allowed free baggage, carryon allowance per passenger, and baggage restrictions.
3. The airline can display this information to the passenger via the client.

Sample Response:

```
{
  "segmentDisclosure": [
    {
      "passengerBaggageDisclosure": [
        {
          "checkedInBaggage": {
            "freeBaggageAllowance": {
              "totalUnits": "01P",
              "baggageRestrictions": [
                {
                  "dimensionDescription": "62 LINEAR INCHES, or 158 LINEAR
                    CENTIMETERS",
                  "weightDescription": "50 POUNDS, or 23 KILOGRAMS"
                }
              ]
            },
            "excessBaggage": [
              {
                "bagPrice": {
                  "alternatives": [
                    [
                      {
                        "amount": 750,
                        "currency": "MXN"
                      }
                    ]
                  ]
                },
                "baggageRestriction": {
                  "dimensionDescription": "62 LINEAR INCHES, or 158 LINEAR
                    CENTIMETERS",
                  "weightDescription": "50 POUNDS, or 23 KILOGRAMS"
                }
              }
            ]
          },
          "carryOnBaggage": {
            "freeBaggageAllowance": {
              "baggageRestrictions": []
            },
            "excessBaggage": []
          },
          "passengerType": "ADT"
        }
      ],
      "embargoInformation": {},
      "segment": {
        "origin": "MEX",
        "destination": "SFO"
```

```
      }
    },
    {
      "passengerBaggageDisclosure": [
        {
          "checkedInBaggage": {
            "freeBaggageAllowance": {
              "totalUnits": "01P",
              "baggageRestrictions": [
                {
                  "dimensionDescription": "62 LINEAR INCHES, or 158 LINEAR
                     CENTIMETERS",
                  "weightDescription": "50 POUNDS, or 23 KILOGRAMS"
                }
              ]
            },
            "excessBaggage": [
              {
                "bagPrice": {
                  "alternatives": [
                    [
                      {
                        "amount": 750,
                        "currency": "MXN"
                      }
                    ]
                  ]
                },
                "baggageRestriction": {
                  "dimensionDescription": "62 LINEAR INCHES, or 158 LINEAR
                     CENTIMETERS",
                  "weightDescription": "50 POUNDS, or 23 KILOGRAMS"
                }
              }
            ]
          },
          "passengerType": "ADT"
        }
      ],
      "embargoInformation": {},
      "segment": {
        "origin": "SFO",
        "destination": "MEX"
      }
    },
    {
      "passengerBaggageDisclosure": [
        {
          "carryOnBaggage": {
```

```
            "freeBaggageAllowance": {
              "baggageRestrictions": []
            },
            "excessBaggage": []
          },
          "passengerType": "ADT"
        }
      ],
      "embargoInformation": {},
      "segment": {
        "origin": "SFO",
        "destination": "GDL"
      }
    },
    {
      "passengerBaggageDisclosure": [
        {
          "carryOnBaggage": {
            "freeBaggageAllowance": {
              "baggageRestrictions": []
            },
            "excessBaggage": []
          },
          "passengerType": "ADT"
        }
      ],
      "embargoInformation": {},
      "segment": {
        "origin": "GDL",
        "destination": "MEX"
      }
    }
  ]
}
```


Errors

The errors that may be returned by this service are listed below:

HTTP Code	Error Code	Message Code or Text	Type	Cause
400	ERR.SSW.CLIENT.INVALID_REQUEST	conversation '*<id>*' was completed or doesn't exist	Application	Service was called with a non-existent or expired execution Id, or the session has expired. Note that once the /purchase service is called

				the session and conversation (executionId) end.
400	ERR.SSW.APP.BUSINESS_ERROR	Error during reprice of types <*types*>	BusinessLogic	An error occurred during the pricing of the selected flight(s).
400	ERR.SSW.APP.BUSINESS_ERROR	'Reservation' object has to be set first.	BusinessLogic	You have called /products/air/bags before the passenger has selected an itinerary and added it to the session.
500	ERR.SSW.INTERNAL_ERROR	<*message*>	Application	The service is unable to complete the request, due to a failure of a downline system.
503	ERR.SSW.INTERNAL.SERVICE_UNAVAILABLE	Service is disabled in configuration.	Application	Service is disabled in configuration.

8.4 /products/air/search Service

Searches for flights that match the criteria specified in the parameters. Airlines can collect search criteria from passengers via their client and then use this service to obtain a list of flights meeting the passenger's search criteria. Airlines can use the search results to populate a selectable list of flights in the client.

8.4.1 POST Operation

Performs a search for flights using the search criteria specified in the parameters. The response contains a list of flights meeting the search criteria, with fare offerings and detailed price breakdowns.

Request URL Syntax

```
http://<host>:<port>/<apiContextPath>/products/air/search?jipcc=<storefront>
```

Request CURL Syntax

```
curl -X POST --header "Content-Type: application/json" --header "Accept:
application/json" -d "body"
"http://<host>:<port>/<apiContextPath>/products/air/search?jipcc=<storefront>"
```

Parameters

The parameters passed in this operation are:

Parameter Name	Parameter Type	Data Type	Definition
body	body	See parameter model.	Flight search criteria in JSON format.

Parameter Model

```
AirSearch {
  cabinClass (string): Specify cabin class = ['Economy', 'Business',
        'First'],
  awardBooking (boolean, optional): True if search is for award fares.,
  searchType (string, optional): Search type = ['CALENDAR30', 'BRANDED',
        'LOWESTFARE'],
  promoCodes (Array[string], optional): Promo codes associated with
        trip/passengers.,
  itineraryParts (Array[SearchItineraryPart], optional): Air search
        parameters,
  passengers (object, optional): Specify type and number of passengers
}
SearchItineraryPart {
  from (Location, optional): Desired origin.,
  to (Location, optional): Desired destination.,
  when (Time, optional): Desired departure time.
}
Location {
  code (string, optional): 3 letter location code.,
  useNearbyLocations (boolean, optional): True if we should use nearby
        location search.
}
Time {
  date (string, optional): Specific date.
}
```

Field Information

The request parameter model indicates that all fields are optional for this service. However, for backend processing to succeed a number of the parameter fields are required. The required fields are listed in the following table.

Field Group	Field Name	Required?	Notes
	cabinClass	Y	Validation is applied to this field. The validation is not configurable. The validation is: • Must be from: Economy, Business, First
	awardBooking	N	To obtain results for award booking, “true” must be sent. Validation is not applied to this field.
	searchType	Y	Validation is applied to this field. The validation is not configurable. The validation is: • Must be from: BRANDED, LOWESTFARE

itineraryParts		Y	If the request contains one ItineraryPart it will be regarded as a request for a one-way itinerary; if it contains two ItneraryParts it will be regarded as a request for a round-trip itinerary, and if it contains more than two ItineraryParts is will be regarded as a request for a multi-city itinerary.
	/from/code	Y	A value is required for each ItineraryPart. The value must conform to the following: max=3 min=3 regExp=^[a-zA-Z]{3}$
	/from/useNearbyLocations	N	Validation is not applied to this field.
	/to/code	Y	A value is required for each ItineraryPart. The value must conform to the following: max=3 min=3 regExp=^[a-zA-Z]{3}$
	/to/useNearbyLocations	N	Validation is not applied to this field.
	when/date		A value Is required for each ItineraryPart. The date must be in the format: yyyy-mm-dd

passengers	ADT	Y	The number and types of passengers can be passed in the following format. At least one passenger of type ADT is required.
			“passengers”: {
			"ADT" :"2",
			"CHD" :"1"
			}
			Validation is applied to this field.
			The validation is configurable.
			The validation is:
			• Must be from:
			ADT,CHD,INF,YTH,STU,MIL, SRC,INS,YCB

Timeout and Retry

For this operation, the timeout is 240 seconds. The timeout interval allows for complete processing of the request, including processing performed by downline systems. Retries are allowed, but keep in mind that retrying may change reservation state.

Response Model

```
AirSearchResults {
```

```
  searchResultMetaData (SearchResultMetaData, optional): Additional
        results properties.,
  fareFamilies (Array[FareFamily], optional): List of all available fare
        families aka. brands.,
  unbundledOffers (Array[Inline Model 1], optional): List of lists of
        unbundled offers. Sorted by legs.,
  unbundledAlternateDateOffers (Array[Inline Model 2], optional): List of
        lists of unbundled offers for alternate dates. Sorted by legs.,
  bundledOffers (Array[Offer], optional): List of all bundled offers.
        Offer for whole itinerary.,
  bundledAlternateDateOffers (Array[Offer], optional): List of all
        bundled offers for alternate dats. Offer for whole itinerary.,
  brandedResults (BrandedResults, optional): Results of branded offers
        grouped by itinerary parts.,
  resultMapping (ResultMapping, optional): Combinability mapping for
        unbundled offers.,
  soldOutDatesOutbound (Array[string], optional),
  soldOutDatesInbound (Array[string], optional),
  noneScheduledDatesOutbound (Array[string], optional),
  noneScheduledDatesInbound (Array[string], optional),
  warnings (Array[string], optional),
  currency (string, optional): Main currency used in search results.
        Three letter ISO currency code.,
  promocodeValid (boolean, optional): Flag indicating if used promocode
        was valid.,
  negotiateFarePresent (boolean, optional): Flag indicating if negotiated
        fares are present in response.,
  conversionRatesFound (boolean, optional): Flag indicating if conversion
        rates were found for this RBE conversion method search. Applies only
        for RBE with conversion method.
}
SearchResultMetaData {
  branded (boolean, optional): Is result branded.,
  multipleDateResult (boolean, optional): Is multiple date results either
        with alt dates or a calendar result.,
  composedResult (boolean, optional),
  interlineRoute (boolean, optional): Is it on interline route.
}
FareFamily {
  brandId (string, optional): FareFamily identifier.,
  brandLabel (Array[BrandLabel], optional),
  brandText (string, optional): Simple brand text in default language.,
  marketingTexts (Array[MarketingText], optional),
  brandAncillaries (FlightAncillaries, optional): List of all ancillaries
        available to flights in this brand.,
  fareFamilyRemarkRPH (integer, optional)
}
Inline Model 1 [
  Offer
```

```
]
Inline Model 2 [
  Offer
]
Offer {
  shoppingBasketHashCode (integer, optional): Offer identifier. Might not
        be unique. Should be consistent between searches.,
  brandId (string, optional): Brand associated with this offer.,
  soldout (boolean, optional): Flag indicating if the offer is sold out.,
  fare (ApiPrice, optional): Fare sum up according to currencies.,
  taxes (ApiPrice, optional): Taxes sum up according to currencies.,
  total (ApiPrice, optional): Price sum up according to currencies.,
  seatsRemaining (SeatsRemaining, optional): Information about seats
        remaining for this offer.,
  itineraryPart (Array[ItineraryPart], optional): Offered itinerary
        parts.,
  cabinClass (string, optional): Cabin class for this offer = ['Economy',
        'Business', 'First']
}
BrandedResults {
  itineraryPartBrands (Array[Inline Model 3], optional)
}
ResultMapping {
  simplifiedRoundTripMapping (object, optional): Simple one to many
        mapping by offer shoppingBasketHashCode,
  combinabilityMapping (Array[BrandedFareItineraryOfferMapping],
        optional): More complex and more combinability mapping.
}
BrandLabel {
  programId (string, optional),
  languageId (string, optional): Language id. In format %LANG%_%COUNTRY%
        e.g. en_US, en_GB where %LANG% is lowercase 2 to 8 language code.
        and (optional) %COUNTRY% country uppercase two-letter ISO-3166 code
        and numeric-3 UN M.49 area code.,
  marketingText (string, optional): Simple brand label to be used.,
  brandLabelUrl (string, optional): URL to brand description.
}
MarketingText {
  programId (string, optional),
  languageId (string, optional): Language id. In format %LANG%_%COUNTRY%
        e.g. en_US, en_GB where %LANG% is lowercase 2 to 8 language code.
        and (optional) %COUNTRY% country uppercase two-letter ISO-3166 code
        and numeric-3 UN M.49 area code.,
  marketingText (string, optional): Detailed HTML code brand description.
}
FlightAncillaries {
  flightAncillary (Array[FlightAncillary], optional)
}
ApiPrice {
```

```
  alternatives (Array[Inline Model 4], optional): List of price
        alternative lists. Each element of main list is a full price. e.g.
        list [[700 AUD],[300 AUD, 80000 FFCURRENCY]
}
SeatsRemaining {
  count (integer, optional): Number of remaining seats.,
  lowAvailability (boolean, optional): Indicator if current seat count is
        considered low. This is just a guideline for frontend.
}
ItineraryPart {
  segments (Array[Segment], optional): List of itinerary part segments.,
  stops (integer, optional): Number of stops,
  totalDuration (integer, optional): Total flight duration in minutes
}
Inline Model 3 [
  ItineraryPartBrands
]
BrandedFareItineraryOfferMapping {
  from (Array[integer], optional): List of all 'from' offers.,
  to (Array[integer], optional): List of all 'to' offers.
}
FlightAncillary {
  id (string, optional),
  subCode (string, optional): ATPCO sub code for ancillary.
}
Inline Model 4 [
  Amount
]
Segment {
  segmentOfferInformation (SegmentOfferInformation, optional): Additional
        segment information.,
  duration (integer, optional): Flight duration in minutes,
  cabinClass (string, optional) = ['Economy', 'Business', 'First'],
  equipment (string, optional): Aircraft equipement code.,
  aircraftLeaseText (string, optional): Aircraft lease disclosure text.,
  flight (Flight, optional): Flight details of this segment.,
  origin (string, optional): IATA airport code of origin.,
  destination (string, optional): IATA airport code of destination.,
  departure (string, optional): Departure date time.,
  arrival (string, optional): Arrival date time.
}
ItineraryPartBrands {
  itineraryPart (ItineraryPart, optional): Offered itinerary part.,
  brandOffers (Array[Offer], optional): Brand bucket offers,
  duration (integer, optional): Flight duration in minutes,
  departure (string, optional): Departure date time.,
  arrival (string, optional): Arrival date time.
}
Amount {
```

```
  amount (number, optional): Amount.,
  currency (string, optional): Currency. Three letter ISO code or
        'FFCURRENCY' for points/miles.
}
SegmentOfferInformation {
  flightsMiles (integer, optional): Flight miles,
  flightAncillaries (FlightAncillaries, optional): List of all
        ancillaries available to flights in this segment.,
  awardFare (boolean, optional): Flag indicating if this segment is an
        award fare.
}
Flight {
  flightNumber (integer, optional): Flight number.,
  airlineCode (string, optional): Marketing airline IATA code.,
  operatingAirlineCode (string, optional): Operating airline IATA code.,
  stopAirports (Array[string], optional): Stop airports associated with
        this flight.
}
```

Response Content Type

application/JSON

Errors

The errors that could be returned by this operation are listed below:

HTTP Code	Error Code	Message Code or Text	Type	Cause
400	ERR.SSW.APP.BUSINESS_ERROR	NO_AVAILABILITY_FOUND	BusinessLogic	There are no flights available that meet the search criteria.
400	ERR.SSW.APP.BUSINESS_ERROR	*<warning1>, < warning2>, …*	BusinessLogic	Warnings that occurred during search operation.
400	ERR.SSW.APP.BUSINESS_ERROR	*<error1>, < error2>, …*	BusinessLogic	Errors that occurred during search operation.
400	ERR.SSW.CLIENT.INVALID_REQUEST	Validation error.	Validation	The value supplied is invalid. Please see the details.
400	ERR.SSW.CLIENT.JSON_MAPPING_ERROR	Validation error	Validation	
500	ERR.SSW.INTERNAL_ERROR	*<message>*	Application	The service is unable to complete the request, possibly due to a failure of a downline system.
503	ERR.SSW.INTERNAL.SERVICE_UNAVAILAB	Technical issue	Application	Unknown issues have been

	LE			encountered during search operation.
503	ERR.SSW.INTERNAL.SERVICE_UNAVAILABLE	Service is disabled in configuration.	Application	Service is disabled in configuration.

Search Validation Errors

The detailed information regarding the cause of the error is given in the details field, where the key or keys identify the fields which failed validation and the value summarize the cause of the failed validation.

Key	Value	Cause
various fields given in the Field Information table	validation.error.required_but_not_found	A value is required for the specified field but not supplied.
various fields given in the Field Information table	validation.error.not_valid	The value supplied is invalid.
various fields given in the Field Information table	validation.text.tooShort	The value supplied is greater than the maximum allowed for the field.
various fields given in the Field Information table	validation.text.tooLong	The value supplied does not match the regular expression pattern established for the field.
various fields given in the Field Information table	validation.text.pattern	The value supplied does not match the regular expression pattern established for the field.
searchType	invalid.format	Illegal searchType field value has been given.
itineraryParts[<*index*>].when.date	invalid.date|<*date*>	The service received an invalid date, e.g. a date in the past.
passengers	invalid.type|<*passengerType*>	Invalid passenger type.
promoCodes	invalid.amount	The service received an invalid number of promo codes; there should be no more than one code per passenger.
promoCodes	invalid|<*promoCode1*>|<*promoCode2*>	The service received one or more invalid promotional codes; promotional codes must match codes defined in the admin tool.

8.5 /products/cart Service

This service returns a list of selected segments, ancillaries, seats, discount, price information, bags, carryon, and embargo information in the current itinerary, including information about the number of free bags, carry-ons and also any charges applicable to bags and carry-onsite airline can use this information to populate a shopping cart in the client.

8.5.1 GET Operation

Request URL Syntax

```
http://<host>:<port>/<apiContextPath>/products/cart?jipcc=<storefront>
```

Request CURL Syntax

```
curl -X POST --header "Content-Type: application/json" --header "Accept:
application/json" -d "body"
"http://<host>:<port>/<apiContextPath>/products/cart?jipcc=<storefront>"
```

Response Model

```
ProductsCart {
  productsInformation (BreakdownResponse, optional),
  baggageInformation (BaggageDisclosure, optional)
}
BreakdownResponse {
  itineraryInformation (ItineraryInformation, optional): Itinerary for the
    current booking. Used as a reference. ,
  total (ApiPrice, optional): Total for all products contained in breakdown. ,
  productBreakdowns (Array[ProductBreakdown], optional): List of per product
    breakdowns.
}
BaggageDisclosure {
  segmentDisclosure (Array[SegmentDisclosure], optional): List of baggage,
    carryon and embargo information per segment.
}
ItineraryInformation {
  itinerary (Itinerary, optional): Selected itinerary and flight details ,
  promoCodesUsed (Array[string], optional): Promo codes used to obtain current
     offers. ,
  offersSelected (Array[OfferWithoutPrice], optional): Selected offers from
    search results ,
  passengers (Passengers, optional): Passenger data for current booking.
}
ApiPrice {
  alternatives (Array[Inline Model 1], optional): List of price alternative
    lists. Each element of main list is a full price. e.g. list [[700
    AUD],[300 AUD, 80000 FFCURRENCY]
}
ProductBreakdown {
  label (string, optional): Product label like 'air', 'ancillaries', etc. ,
```

```
  breakdownElements (Array[BreakdownElement], optional): Breakdown for this
    products price. ,
  total (ApiPrice, optional): Total for this product breakdown.
}
SegmentDisclosure {
  passengerBaggageDisclosure (Array[PassengerBaggageDisclosure], optional):
    Baggage/CarryOn Allowance Information for each Passenger type. ,
  embargoInformation (EmbargoInformation, optional): Embargo restrictions for
    this segment. ,
  segment (SegmentKey, optional): Segment origin and destination code.
}
Itinerary {
  origin (string, optional): IATA airport code of origin. ,
  departure (string, optional): Departure date time. ,
  arrival (string, optional): Arrival date time. ,
  destination (string, optional): IATA airport code of destination. ,
  type (string, optional) = ['SEGMENT', 'ITINERARY_PART', 'ITINERARY'],
  segments (Array[Segment], optional): List of itinerary part segments. ,
  stops (integer, optional): Number of stops ,
  totalDuration (integer, optional): Total flight duration in minutes ,
  itineraryParts (Array[ItineraryPart], optional): List of itinerary parts
    (aka segments). ,
  segmentOfferInformation (SegmentOfferInformation, optional): Additiona
    segment information. ,
  duration (integer, optional): Flight duration in minutes ,
  cabinClass (string, optional) = ['Economy', 'Business', 'First'],
  equipment (string, optional): Aircraft equipement code. ,
  aircraftLeaseText (string, optional): Aircraft lease disclosure text. ,
  flight (Flight, optional): Flight details of this segment.
}
OfferWithoutPrice {
  shoppingBasketHashCode (integer, optional): Offer identifier. Might not be
    unique. Should be consistent between searches. ,
  brandId (string, optional): Brand associated with this offer. ,
  seatsRemaining (SeatsRemaining, optional): Information about seats remaining
    for this offer. ,
  itineraryPart (Array[ItineraryPart], optional): Offered itinerary parts.
}
Passengers {
  passengers (Array[Passenger], optional): Passenger List ,
  contact (Contact, optional): Passenger Contact Information
}
Inline Model 1 [
  Amount
]
BreakdownElement {
  label (string, optional): Label for this breakdown element. In a form of a
    machine readable/parseable code. Defined per product type. So it can be
    used for user translation display. ,
```

```
  price (ApiPrice, optional): Price for this element or total for all
    subelements if this is not a leaf. ,
  breakdownElementAssignment (BreakdownElementAssignment, optional): Additiona
    information about what this element is assigned to. ,
  subElements (Array[BreakdownElement], optional): List of all breakdown
    pieces for this element. e.g. it this element is total for taxes
    'subElements' will contain a list of specific taxes with correspondig
    detailed prices. This element is present only whe
}
PassengerBaggageDisclosure {
  checkedInBaggage (CheckedInBaggage, optional): Bags allowed and price
    information for this segment. ,
  carryOnBaggage (CarryOnBaggage, optional): CarryOns allowed and price
    information for this segment. ,
  passengerType (string, optional): Passenger Type like Adult, Child etc., =
    ['ADT', 'YCB', 'YTH', 'CHD', 'C04', 'C11', 'INF', 'INS', 'STU', 'MIL',
    'SRC', 'CMA', 'CMP']
}
EmbargoInformation {
  airlineIataCode (string, optional): Airline IataCode like
     http://www.iata.org/about/members/pages/airline-list.aspx?All=true. ,
  restrictionList (Array[string], optional): List of embargo restrictions.
}
SegmentKey {
  origin (string, optional): IATA airport code of origin. ,
  destination (string, optional): IATA airport code of destination. ,
  departure (string, optional): Departure date time. ,
  arrival (string, optional): Arrival date time.
}
Segment {
  origin (string, optional): IATA airport code of origin. ,
  departure (string, optional): Departure date time. ,
  arrival (string, optional): Arrival date time. ,
  destination (string, optional): IATA airport code of destination. ,
  type (string, optional) = ['SEGMENT', 'ITINERARY_PART', 'ITINERARY'],
  segments (Array[Segment], optional): List of itinerary part segments. ,
  stops (integer, optional): Number of stops ,
  totalDuration (integer, optional): Total flight duration in minutes ,
  itineraryParts (Array[ItineraryPart], optional): List of itinerary parts
    (aka segments). ,
  segmentOfferInformation (SegmentOfferInformation, optional): Additiona
    segment information. ,
  duration (integer, optional): Flight duration in minutes ,
  cabinClass (string, optional) = ['Economy', 'Business', 'First'],
  equipment (string, optional): Aircraft equipement code. ,
  aircraftLeaseText (string, optional): Aircraft lease disclosure text. ,
  flight (Flight, optional): Flight details of this segment.
}
ItineraryPart {
```

```
  origin (string, optional): IATA airport code of origin. ,
  departure (string, optional): Departure date time. ,
  arrival (string, optional): Arrival date time. ,
  destination (string, optional): IATA airport code of destination. ,
  type (string, optional) = ['SEGMENT', 'ITINERARY_PART', 'ITINERARY'],
  segments (Array[Segment], optional): List of itinerary part segments. ,
  stops (integer, optional): Number of stops ,
  totalDuration (integer, optional): Total flight duration in minutes ,
  itineraryParts (Array[ItineraryPart], optional): List of itinerary parts
    (aka legs). ,
  segmentOfferInformation (SegmentOfferInformation, optional): Additiona
    segment information. ,
  duration (integer, optional): Flight duration in minutes ,
  cabinClass (string, optional) = ['Economy', 'Business', 'First'],
  equipment (string, optional): Aircraft equipement code. ,
  aircraftLeaseText (string, optional): Aircraft lease disclosure text. ,
  flight (Flight, optional): Flight details of this segment.
}
SegmentOfferInformation {
  flightsMiles (integer, optional): Flight miles ,
  flightAncillaries (FlightAncillaries, optional): List of all ancillaries
    available to flights in this segment. ,
  awardFare (boolean, optional): Flag indicating if this segment is a award
    fare.
}
Flight {
  flightNumber (integer, optional): Flight number. ,
  airlineCode (string, optional): Marketing airline IATA code. ,
  operatingAirlineCode (string, optional): Operating airline IATA code. ,
  stopAirports (Array[string], optional): Stop airports associated with this
    flight.
}
SeatsRemaining {
  count (integer, optional): Number of remaining seats. ,
  lowAvailability (boolean, optional): Indicator if current seat count is
    considered low. This is just a guideline for frontend.
}
Passenger {
  passengerIndex (integer, optional): Passender Index ,
  passengerDetails (PassengerDetails, optional): Passenger Details ,
  passengerInfo (PassengerInfo, optional): Passenger Information ,
  preferences (Preferences, optional): Preferences like Meal/Seat/Special
    Requests ,
  documentInfo (PassengerDocumentInfo, optional): Passenger Document
    Information ,
  visaInfo (PassengerVisaInfo, optional): Passenger Visa Information
}
Contact {
  emails (Array[string], optional): List of booker contact emails ,
```

```
  phones (Array[Phone], optional): List of booker contact phone numbers ,
  addresses (Array[Address], optional): List of booker contact address
}
Amount {
  amount (number, optional): Amount. ,
  currency (string, optional): Currency. Three letter ISO code or 'FFCURRENCY'
    for points/miles.
}
BreakdownElementAssignment {
  travelPart (TravelPart, optional): Assigned travel part. ,
  passenger (Passenger, optional): Specifies to whom this breakdown element is
    assigned. If passenger is specify passengerType has no meaingn. ,
  passengerType (string, optional): Specifies to which passengerType this
    breakdown element is assigned. Total should be for all passengers with
    this type. ,
  quantity (integer, optional): Quantity of the same values existing in this
     assignment e.g. number of passengers of type for air fare, amount of
     ancillaries with specific code etc.
}
Array[BreakdownElement] {
}
CheckedInBaggage {
  freeBaggageAllowance (FreeBaggageAllowance, optional): Free Checked In
    Baggage Allowance size and weight information. ,
  excessBaggage (Array[ExcessBaggage], optional): List of Excess baggage
    price, size and weight information.
}
CarryOnBaggage {
  freeBaggageAllowance (FreeBaggageAllowance, optional): Free CarryOn
    allowance name, size and weight information. ,
  excessBaggage (Array[ExcessBaggage], optional): List of Excess baggage
    price, size and weight information.
}
FlightAncillaries {
  flightAncillary (Array[FlightAncillary], optional)
}
PassengerDetails {
  firstName (string, optional): Passenger first name ,
  middleName (string, optional): Passenger middle name ,
  maidenName (string, optional): Passenger maiden name ,
  lastName (string, optional): Passenger last name ,
  prefix (string, optional): Passenger prefix ,
  suffix (string, optional): Passenger suffix
}
PassengerInfo {
  dateOfBirth (string, optional): Passenger date of birth ,
  gender (string, optional): Passenger gender like Male/Female ,
  redressNumber (string, optional): Redress number is issued to customers as
    part the TSA's Secure Flight program to speed identification and security
```

```
    clearance. more information available at
    www.tsa.gov/stakeholders/secure- flight-program ,
  knownTravelerNumber (string, optional): Known Travler/TSA PreCheck Number ,
  type (string, optional): Passenger type Adult/Child ,
  emails (Array[string], optional): List of passenger emails ,
  phones (Array[Phone], optional): List of passenger phones ,
  address (Address, optional): Passenger address
}
Preferences {
  specialPreferences (SpecialPreferences, optional): Passenger special
    specialPreferences like meal, seat and special request ,
  frequentFlyer (FrequentFlyer, optional): Passenger frequentflyer account
    details
}
PassengerDocumentInfo {
  documentNumber (string, optional): passenger document number ,
  issuingCountry (string, optional): document issued ISO country code like :
    https://en.wikipedia.org/?title=ISO_3166-1 ,
  countryOfBirth (string, optional): passenger countryOfBirth ISO country code
    like : https://en.wikipedia.org/?title=ISO_3166-1 ,
  issueDate (string, optional): document issued date ,
  documentType (string, optional): document type ,
  nationality (string, optional): passenger nationality ISO country code like
    : https://en.wikipedia.org/?title=ISO_3166-1 ,
  expirationDate (string, optional): document expiration date
}
PassengerVisaInfo {
  documentNumber (string, optional): passenger document number ,
  issuingCountry (string, optional): document issued ISO country code like :
    https://en.wikipedia.org/?title=ISO_3166-1 ,
  countryOfBirth (string, optional): passenger countryOfBirth ISO country code
    like : https://en.wikipedia.org/?title=ISO_3166-1 ,
  issueDate (string, optional): document issued date ,
  issuePlace (string, optional): document issued place
}
Phone {
  type (string, optional): Phone type ,
  countryCode (string, optional): Phone country code like
    https://en.wikipedia.org/wiki/Country_code ,
  areaCode (string, optional): Phone area code ,
  number (string, optional): Phone number ,
  extension (string, optional): Phone extension
}
Address {
  addressType (string, optional): Address type ,
  street1 (string, optional): First line of street address ,
  street2 (string, optional): Second line of street address like apt, suite..,
  postcode (string, optional): Address postal code ,
  state (string, optional): Name of the state ,
```

```
  city (string, optional): Name of the city ,
  country (string, optional): ISO country code like :
    https://en.wikipedia.org/?title=ISO_3166-1
}
TravelPart {
  origin (string, optional): IATA airport code of origin. ,
  departure (string, optional): Departure date time. ,
  arrival (string, optional): Arrival date time. ,
  destination (string, optional): IATA airport code of destination. ,
  type (string, optional) = ['SEGMENT', 'ITINERARY_PART', 'ITINERARY'],
  segments (Array[Segment], optional): List of itinerary part segments. ,
  stops (integer, optional): Number of stops ,
  totalDuration (integer, optional): Total flight duration in minutes ,
  itineraryParts (Array[ItineraryPart], optional): List of itinerary parts
    (aka legs). ,
  segmentOfferInformation (SegmentOfferInformation, optional): Additional
    segment information. ,
  duration (integer, optional): Flight duration in minutes ,
  cabinClass (string, optional) = ['Economy', 'Business', 'First'],
  equipment (string, optional): Aircraft equipement code. ,
  aircraftLeaseText (string, optional): Aircraft lease disclosure text. ,
  flight (Flight, optional): Flight details of this segment.
}
FreeBaggageAllowance {
  totalUnits (string, optional): Free Baggage Information Disclosure. ,
  totalWeight (string, optional): Total Weight allowed on Flight. ,
  baggageRestrictions (Array[BaggageRestriction], optional): Baggage Name,
    Number Of Units, Size and Weight Restrictions.
}
ExcessBaggage {
  bagPrice (ApiPrice, optional): Price of the bag/carryOn. ,
  baggageRestriction (BaggageRestriction, optional): Baggage Size and Weight
    descriptions ,
  allowedStatus (string, optional): Returns whether Baggage is allowed for
    free, permitted or not = ['FREE', 'NOT_PERMITTED', 'UNKNOWN']
}
FlightAncillary {
  id (string, optional),
  subCode (string, optional): ATPCO sub code for ancillary.
}
SpecialPreferences {
  mealPreference (string, optional): Choice of Meal ,
  seatPreference (string, optional): Seat preference ,
  specialRequests (Array[string], optional): SpecialRequests selected by
    passenger
}
FrequentFlyer {
  airline (string, optional): Frequentflyer account airline code (IATA code) ,
  number (string, optional): Frequentflyer number ,
```

```
  tierLevel (string, optional): Frequentflyer tier Level
}
BaggageRestriction {
  dimensionDescription (string, optional): Measurment of bag/carryOn in linear
    inches/centimeters with detail description. for eg., UP TO 62 LINEAR
    INCHES/158 LINEAR CENTIMETERS ,
  weightDescription (string, optional): Weight of the bag/carryOn in
    Kilograms/Pounds with detail description. for eg., UP TO 50 POUNDS/23
    KILOGRAMS ,
  allowedUnits (integer, optional): Number of bags/carryOns allowed. ,
  baggageName (string, optional): Baggage like Musical Instrument etc.,.
}
```

Processing Logic

1. The /products/air/search and /products/air services must to be called before calling this service.

Errors

The errors that may be returned by this service are listed below:

HTTP Code	Error Code	Message Code or Text	Type	Cause
400	ERR.SSW.CLIENT.INVALID_REQUEST	conversation '<*id*>' was completed or doesn't exist	Application	Service was called with a non-existent or expired execution Id, or the session has expired. Note that once the /purchase service is called the session and conversation (executionId) end.
400	ERR.SSW.APP.BUSINESS_ERROR	'Reservation' object has to be set first.	BusinessLogic	Trying to perform get on /products/cart before flight has been selected
500	ERR.SSW.INTERNAL_ERROR	<*message*>	Application	The service is unable to complete the request due to a unknown error.
503	ERR.SSW.INTERNAL.SERVICE_UNAVAILABLE	Service is disabled in configuration.	Application	Service is disabled in configuration.