Sabre® APIs

Guide to Accessing and Consuming Orchestrated Sabre APIs

June 2018
Version 3.9.0

Sabre Inc.

3150 Sabre Drive, Southlake, TX 76092

Tel: 682 605 1000

www.sabre.com

Table of Contents

Document Revision Information

Revision number	Date	Revised by	Reason
V3.8.0	5/14/2018	Tomasz Drag	Added new service description ExchangeBooking RQ (pages 40 – 44)
V3.9.0	6/7/2018	Tomasz Drag	Added new section CreatePassengerNameRecordRQ/Post booking validation of …. (page 33)

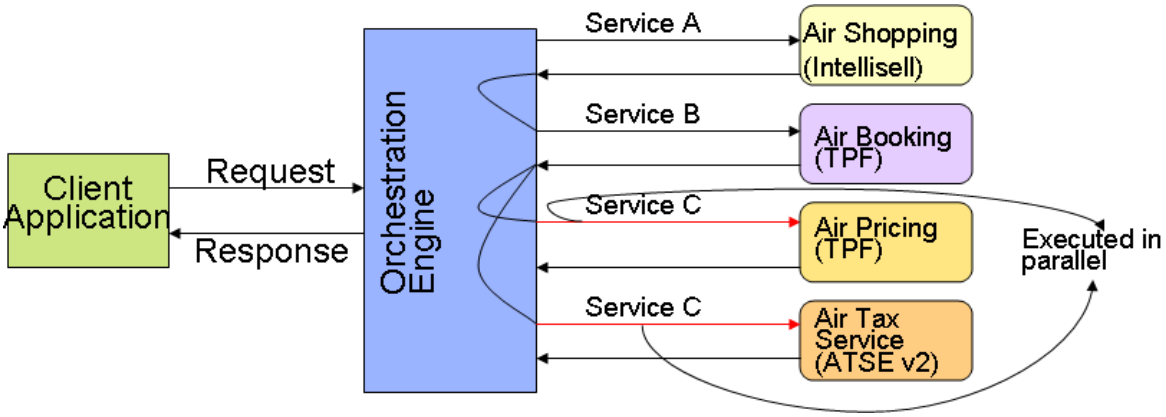
Introduction to Orchestrated *Sabre APIs*

The travel industry has seen a steady growth in ecommerce during the last few years. At the same time, on-line travel agencies, travel suppliers, and traditional agencies with on-line presence are continuing to face several challenges, such as building brand loyalty, high look-to-book ratios, and unsatisfactory customer experience. Web services technology has emerged as a key enabler for application developers in travel agencies, airlines, and providers of related content to deliver products and services to overcome some of these business challenges. *Sabre APIs* offers Orchestrated *Sabre APIs* to help agencies, airlines, and other customers reduce their overall travel content integration costs.

Overview

An orchestrated API easily automates the process of a commonly performed business function or workflow. Sabre's Orchestrated *APIs* bundle several functions/operations into a single Web service call. This type of service consists of transactions that are mapped to individual Web services, and executed either sequentially, in parallel, or both.

An example of how a single orchestrated service request is fulfilled by executing multiple Web service requests is illustrated in the following execution model of an orchestrated service. A single service request call will provide you with at response that retrieves your content and performs the processing for you seamlessly. By using orchestrated services, customers can develop client applications faster and may see an improvement in the overall response time to offer a better customer experience.



Benefits of utilizing Sabre's Orchestrated APIs

- Applications can be deployed to production sooner. By using pre-built workflow functions that have been architected and designed to work together, development time and quality assurance are reduced, shortening the time to place an application into production.
- Integration costs are lowered. Orchestrated services help reduce development complexity, letting clients accomplish more with a single service call.
- Learning curve is shortened. Newcomers to the travel industry or Web services can use these services more easily than the low level services.
- Gain access to content and business logic in multiple Sabre data sources. One orchestrated service may obtain content from multiple sources and orchestrate business logic, shielding clients from making multiple points of entry to various systems.
- Improved response time. By avoiding the round trip latency inherent with multiple individual low level service calls to perform a function, responses are faster.
- Improved bandwidth utilization. Because the HTTP overhead associated with multiple low level Web service calls is avoided, use of bandwidth is improved.
- Improved end-to-end performance. An orchestrated Web service gives clients an opportunity to optimize existing applications.
- Ability to adapt to changing market needs quickly. Clients can build new products and services faster by integrating an orchestrated service versus multiple service calls.
- The majority of the operations available inside an orchestrated service are optional. As such, the client application has the flexibility to control which operations are executed, allowing it to mix and match functions to meet its specific needs.
- The client application that is consuming orchestrated services has the flexibility to control the error/exception processing during the execution of the low level services within the orchestrated service. Client applications have the ability to halt the execution of subsequent service operations by setting "HaltOnError" attribute at the root request element.
- Each 3.x orchestrated request and response is designed so that individual calls are delineated by an "stl:element" element that can be used to associate particular XML nodes to specific low level calls. Client applications can easily identify the offending node using the "stl:element" element returned in the error message.

Orchestrated *APIs* Overview

There are a number of orchestrated *Sabre Web Services* available for consumption:

EnhancedAirBookRQ
PassengerDetailsRQ.
CreatePassengerNameRecordRQ
AirTicketRQ

In terms of feature/function:

- EnhancedAirBookRQ is used to book and price an air itinerary, and retrieve applicable tax-related information.

- PassengerDetailsRQ is used to create a basic Passenger Name Record (PNR).

- CreatePassengerNameRecordRQ combines EnhancedAirBookRQ and PassengerDetailsRQ to create a full Passenger Name Record (PNR)

- AirTicketRQ is used to issue multiple Air Tickets, EMDs or Price Quote Reissue records within a single call.

Please note that services can be used standalone or (as is in the below examples relating to EnhancedAirBookRQ and PassengerDetailsRQ) can be combined with others.

Example 1:
If a client application has the desired air itinerary along with all of the relevant passenger information, the client can invoke:

- EnhancedAirBookRQ – to book and price the air itinerary.
- PassengerDetailsRQ – to add the passenger-related information, associate the passenger-related information to the pricing related information, and end the record.

* If retaining Price Quote records is required, please see Appendix section for instructions.

Example 2:
If a client application has the relevant passenger information, and wants to add it while the customer shops the client can invoke:

- BargainFinderMaxRQ to shop for an air itinerary.
- PassengerDetailsRQ – to add the passenger-related information.
- EnhancedAirBookRQ – to book and price the air itinerary.

- PassengerDetailsRQ – to add any additonal passenger-related information, i.e. seats, SSRs, etc, associate the passenger-related information to the pricing related information, and end the record.

Sessioning Requirements

Below services can use both Session or Sessionless token:

CreatePassengerNameRecordRQ
AirTicketRQ
ExchangeShoppingRQ

Below services require session token:

EnhancedAirBookRQ
PassengerDetailsRQ.

For more details on sessioning requirements please visit Sabre Dev Studio at:
https://developer.sabre.com/resources/getting_started_with_sabre_apis/how_to_get_a_token

Delineating Successful & Failing Transactions

Each orchestrated response message contains several "…/stl:ApplicationResults/STL:Element" occurences that are used to tie a particular operation's response status back to the associated request XML node that caused it to be invoked. This is useful for troubleshooting purposes. If warnings or errors are generated they can quickly be associated back to particular request operations. If an invocation is successful a single stl:Success element is returned.

Example 1:

```
<PassengerDetailsRS xmlns="http://services.sabre.com/sp/pd/v3_1">
   <stl:ApplicationResults xmlns:stl="http://services.sabre.com/STL_Payload/v02_01"
status="Complete">
      <stl:Success timeStamp="2014-08-08T07:13:57.706-05:00"/>
   </stl:ApplicationResults>
         …
</ PassengerDetailsRS>
```

In this example all of the designated request operations succeed.

Example 2:

```
<PassengerDetailsRS xmlns="http://services.sabre.com/sp/pd/v3_1">
   <stl:ApplicationResults xmlns:stl="http://services.sabre.com/STL_Payload/v02_01"
status="NotProcessed">
      <stl:Error type="BusinessLogic" timeStamp="2014-08-08T07:14:33.716-05:00">
         <stl:SystemSpecificResults>
            <stl:Message
code="ERR.SWS.HOST.ERROR_IN_RESPONSE">ERRORTIR</stl:Message>
         </stl:SystemSpecificResults>
      </stl:Error>
   </stl:ApplicationResults>
</PassengerDetailsRS>
```

In this example the request operation failed. …/stl:ApplicationResults/stl:Error/stl:SystemSpecificResults contains details about the error.

Message Processing

EnhancedAirBookRQ and PassengerDetailsRQ offer clients several options for controlling what happens when errors are encountered:

1. …/HaltOnError flag
2. …/IgnoreOnError flag

HaltOnError Flag

Flag defaults to false. The HaltOnError flag take the form of "…/**HaltOnError**" attribute located at the root element of the request message. This attribute controls whether or not the processing of the orchestrated service is stopped if an encountered while an operation processes.

Request example (Child):

```
<PassengerDetailsRQ xmlns="http://services.sabre.com/sp/pd/v3_1"
                HaltOnError="false" IgnoreOnError="false">
   <PostProcessing IgnoreAfter="false" RedisplayReservation="true">
      <EndTransactionRQ>
         <EndTransaction Ind="true"/>
         <Source ReceivedFrom="SWS TESTING"/>
      </EndTransactionRQ>
         …
</ PassengerDetailsRQ>
```

In this request, the client application opted to set "…/HaltOnError" attribute to "true." This means that if any operation fails during invocation the orchestration engine will recognize the failure and halt any subsequent requests/processing.

IgnoreOnError Flag

The client application can utilize the "…/IgnoreOnError" element/attribute pair to ignore the entire transaction if an error is encountered during processing.

```
<PassengerDetailsRQ xmlns="http://services.sabre.com/sp/pd/v3_1"
                HaltOnError="false" IgnoreOnError="false">
         …
</ PassengerDetailsRQ>
```

Please note that the "…/IgnoreOnError" flag works only when the transaction is stopped due to an error, so it usually is combined with the "…/HaltOnError" attribute.

Note, it is possible that the ignore operation could also fail, so the client application needs to check the response to ensure that it succeeded.

IgnoreAfter Flag

Flag defaults to false. When IgnoreAfter flag is set to true, service ignores whole transaction at the end of the flow. Service utilizies ignoreTransactionLLSRQ, when error or warning encountered and they will be merged into application results.

Recovery From PREVIOUS ENTRY ACTIVE Error

Sometimes asynchronous processing in Sabre or airline backend systems may not finish in a timely manner. In such situations the host replies with PREVIOUS ENTRY ACTIVE error. Orchestrated services detect this error and retry requests. Three retry attempts are made 1sec apart.

Fatal Errors

There are two kinds of fatal errors which stop the processing regardless of the values set via the "…/HaltOnError" attribute or the "…/IgnoreOnError@Ind" element/attribute:

1. *Timeout:* Each Orchestrated Sabre Web Service is set to timeout at two minutes. The timeout can be decreased by defining, "soap-env:Envelope/soap-env:Header/eb:MessageHeader/eb:MessageData/eb:Timeout" value in the request (in seconds). However, we do not recommend that clients utilize this functionality, because it could cause premature errors if the back-end content systems are slow to respond.

2. *Internal error:* When an error occurs outside of the operations specified in the Orchestrated *API* message it is considered to be an internal error. A good example of this sort of error would be a connection refused error encountered when the orchestration engine is communicating with the back-end content systems.

Internal vs provider error / warning

Whenever a scenario occurs where the application needs to inform the user of any errors or warnings encountered, the processing will be as follows:

- There will always be a single error tag within the response:

```
<Error type="Application" timeStamp="2018-11-07T03:51:24.808-06:00">
        <SystemSpecificResults>
                <Message code="ERR.SP.PROVIDER_ERROR">text</Message>
        </SystemSpecificResults>
</Error>
```

- There can be multiple warning tags:

```
<Warning type="Application" timeStamp="2018-11-07T03:51:24.793-06:00">
        <SystemSpecificResults>
                <Message code="WARN.SP.INTERNAL_ERROR">text</Message>
        </SystemSpecificResults>
</Warning>
<Warning type="Application" timeStamp="2018-11-07T03:51:24.793-06:00">
        <SystemSpecificResults>
                <Message code="WARN.SP.INTERNAL_ERROR">text</Message>
        </SystemSpecificResults>
</Warning>
```

As one can see from the above examples, there can be two types of errors/warnings coming from the application – SP.PROVIDER_ERROR or SP.INTERNAL_ERROR. “Internal error” points to an issue within the SP application itself, whereas “Provider errors” always point to issues from within the low level services that the orchestration calls.

Orchestrated Services Description

PassengerDetailsRQ

https://developer.sabre.com/docs/read/soap_apis/management/itinerary/Passenger_Details

The PassengerDetailsRQ service allows client applications to create shell PNRs containing names, phone numbers, email addresses, customer numbers, passenger types, address information, remarks, and retention segments. The client application has the ability to end the transaction once processing is complete, or to leave the transaction open in the AAA for subsequent processing. If this option is chosen the client application can add additional information into the PNR via existing TPF Connector-based *Sabre Web Service* calls, or other orchestrated service calls before ending the transaction.

PassengerDetailsRQ orchestrates the following operations:

1. SabreCommandLLSRQ (N*(Profile Name)(end-item)NM)
2. TravelItineraryAddInfoLLSRQ (-, 9, PE, DK, PD, W-, 7)
3. MiscSegmentSellLLSRQ (0OTH, 0MCO, 0INS, 0PTA)
4. SpecialServiceLLSRQ (3, 4)
5. AddRemarkLLSRQ (5)
6. AirSeatLLSRQ (4G)
7. ARUNK_LLSRQ (0AA)
8. SabreCommandLLSRQ (PQL(record number)(* or –)(name number)
9. EndTransactionLLSRQ (6, E)
10. QueuePlaceLLSRQ (QP)
11. TravelItineraryReadRQ
12. IgnoreTransactionLLSRQ (I)

Client applications have the ability to pass any of the services contained within the workflow outlined previously, and in terms of responses they can opt to only receive the record locator generated as a result of the process, or to receive the entire PNR generated as a result of the process via the TravelItineraryReadRQ response message.

In case PassengerDetailsRQ requested adding frequent flyer number to PNR and PostProcessing/@RedisplayReservation was set to true the orchestration engine will ensure that frequent flyer information has been added to the PNR before response is returned to customer. In case PassengerDetailsRQ doesn't find frequent flyer information in the displayed PNR it will perform 2 more attempts, each after 1s delay. If frequent flyer data is not present in PNR after 3 attempts then response is returned and warning is added to ApplicationResults: "Missing expected CustLoyalty information". See example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<PassengerDetailsRS xmlns="http://services.sabre.com/sp/pd/v3_1">
   <stl:ApplicationResults xmlns:stl="http://services.sabre.com/STL_Payload/v02_01"
status="Complete">
      <stl:Success timeStamp="2014-08-08T07:13:47.884-05:00"/>
      <stl:Warning type="BusinessLogic" timeStamp="2014-08-21T09:44:44.353-05:00">
         <stl:SystemSpecificResults>
            <stl:Message code="WARN.SP.PROVIDER_WARNING">Missing expected
CustLoyalty information</stl:Message>
         </stl:SystemSpecificResults>
      </stl:Warning>
   </stl:ApplicationResults>
```

Completing a Passenger Name Record (PNR) with single call to PassengerDetailsRQ after call to EnhancedAirBookRQ

The following is an explanation on how a PNR can be completed by performing a call to EnhancedAirBookRQ + a single subsequent PassengerDetailsRQ call, when the client application needs the PNR to store Price Quote records.

Whenever the price 'Retain' flag is set to true (in order to retain the Price Quote records in the PNR) within the EnhancedAirBookRQ service call:

```
<EnhancedAirBookRQ>
<!-- .... -->
<OTA_AirPriceRQ>
  <PriceComparison AmountSpecified="88.2"/>
  <PriceRequestInformation Retain="true">
       <OptionalQualifiers>
          <PricingQualifiers>
            <PassengerType Code="ADT" Quantity="1"/>
            <PassengerType Code="INF" Quantity="1"/>
          </PricingQualifiers>
       </OptionalQualifiers>
  </PriceRequestInformation>
</OTA_AirPriceRQ>
<!-- .... -->
</EnhancedAirBookRQ>
```

And the client application wants to create a PNR as the result of the subsequent PassengerDetailsRQ service call (completing the mandatory PNR information such as contact phone number, ticket time limit, 'received from', passenger name/s - and using

PostProcessing/EndTransactionRQ/EndTransaction Ind="true"), then the same PassengerDetailsRQ request needs to explicitly specify the relationship/link between the passengers and the Price Quote record/s that will be generated and retained in the PNR being built.

The relationship needs to be established using the passenger name number of each passenger and the associated Price Quote record number, considering the passenger type.

As an example, if the call to EnhancedAirBookRQ includes the OTA_PriceRQ section as shown above, the following Price Quote records will be generated:

- Price Quote Record #1 for Adult (ADT) passengers
- Price Quote Record #2 for Infant (INF) passengers

NOTE: when generating and retaining the Price Quote records, the EnhancedAirBookRQ service respects the order in which the PassengerType elements are specified, thus, the order of the associated Price Quote record numbers is the same.

This way the call to PassengerDetailsRQ needs to include:

```
<PriceQuoteInfo>
    <Link NameNumber="1.1" Record="1"/>
    <Link NameNumber="2.1" Record="2"/>
</PriceQuoteInfo>
```

Considering the NameNumber and PassengerType is also specified for each passenger - in the same PassengerDetailsRQ request, as follows:

```
<CustomerInfo>
  <!-- .... -->
  <PersonName NameNumber="1.1" PassengerType="ADT">
      <GivenName>MARIA</GivenName>
      <Surname>MAYERS</Surname>
  </PersonName>
  <PersonName Infant="true" NameNumber="2.1" PassengerType="INF">
      <GivenName>TIM</GivenName>
      <Surname>SMITH</Surname>
  </PersonName>
</CustomerInfo>
```

IMPORTANT: if the price Retain flag is set true in the EnhancedAirBookRQ service call, but no relationship/link to the generated and retained Price Quote record/s is specified in the PassengerDetailsRQ service call, then PassengerDetailsRQ response will return this error:

<Message code="ERR.SWS.HOST.ERROR_IN_RESPONSE">MANUALLY LINK NAMES TO THE CORRECT PQ</Message>

Automated handling of hosted vs. non hosted indicators for SSRs/OSIs

Due to the specifics of Sabre system it is necessary to append information whether the SSR/OSI information is to be passed to a hosted or non-hosted carrier. This is controlled within PassengerDetailsRQ by the below attributes:

/PassengerDetailsRQ/SpecialReqDetails/SpecialService/SpecialServiceInfo/AdvancePassenger/VendorPrefs/Airline/@Hosted
/PassengerDetailsRQ/SpecialReqDetails/SpecialService/SpecialServiceInfo/SecureFlight/VendorPrefs/Airline/@Hosted
/PassengerDetailsRQ/SpecialReqDetails/SpecialService/SpecialServiceInfo/Service/VendorPrefs/Airline/@Hosted

Please note that from version 3.3.0, the API will handle the above flags automatically. Therefore, it is not necessary to include below elements/attributes within the payload:

/PassengerDetailsRQ/SpecialReqDetails/SpecialServiceRQ/SpecialServiceInfo/AdvancePassenger/VendorPrefs
/PassengerDetailsRQ/SpecialReqDetails/SpecialServiceRQ/SpecialServiceInfo/SecureFlight/VendorPrefs
/PassengerDetailsRQ/SpecialReqDetails/SpecialServiceRQ/SpecialServiceInfo/Service/VendorPrefs/Airline/@Hosted

Change Log

V3.3.0:

- Upgrade SpecialServiceLLSRQ to 2.2.0
- Upgrade TravelItineraryAddInfoLLSRQ to 2.1.0
- Upgrade TravelItineraryRead to 3.6.0

v3.2.0:

- Upgraded SpecialServiceLLSRQ to version 2.2.1
- Upgraded SabreCommandLLSRQ to version 1.8.1

v3.1.1:

- SubjectArea in TravelItineraryReadRQ for redisplay reservation changed to FULL
- Traffic on PSS has been reduced by using appropriate SubjectArea in other TravelItineraryReadRQ calls
- All errors from ARUNK_LLSRQ are converted to warnings, regardless of HaltOnError flag
- Upgrade to TravelItineraryAddInfoLLSRQ 2.0.3

v3.1.0:

- Update to TravelItineraryReadRQ v3.5.0
- The TravelItineraryReadRQ request is sent to Open Systems instead of TPF thus returned subset of PNR data may differ
- Support of unmasking credit card information in the TravelItineraryReadRQ response was introduced. When the request contains /PassengerDetailsRQ/PostProcessing/@UnmaskCreditCard='true' and a user has EPR keyword CCVIEW then he will be able to see the credit card information in the response

- TravelItineraryReadLLSRQ has been replaced by TravelItineraryReadRQ which goes to Open System instead of TPF

v3.0.0:

- Service can finish with single success or error and multiple warnings.
- A single HaltOnError flag for whole request. If HaltOnError is false, service merges errors from low level services as warnings and continues processing.
- Updates the service to take advantage of several new, underlying TPF Connector-based service versions
- Fixes logic that reports ProfileRQ successful response as error (SPR-50975).
- Adds configurable delay before TravelItineraryReadLLSRQ

EnhancedAirBookRQ

https://developer.sabre.com/docs/read/soap_apis/air/book/orchestrated_air_booking

The EnhancedAirBookRQ service allows client applications to book and price flight segments via a single Web services call. This service also provides the ability to request air tax information. This is useful for clients that operate their own negotiated or private fares databases since most of the time all that they require in regards to pricing is the relevant tax-related information.

EnhancedAirBookRQ orchestrates the following operations:

1. IgnoreTransactionLLSRQ (I)
2. OTA_AirTaxRQ
3. OTA_AirBookLLSRQ (JA)
4. TravelItineraryReadRQ
5. OTA_AirPriceLLSRQ (WP)
6. TravelItineraryReadRQ
7. IgnoreTransactionLLSRQ (I)

EnhancedAirBookRQ allows a client application to perform an ignore transaction prior to booking to ensure that the AAA is clear. This service also has the ability to control what needs to be done when UC segments are encountered. This service allows client applications to specify a wait interval after booking in order to give carriers a chance to respond with updated segment status. In conjunction with this wait interval client applications can also specify for the system to redisplay the itinerary looking for UC segments up to ten times. In the event that a UC segment is encountered, the client application can specify that the system halt processing for further action from the client application, i.e. a new request utilizing different marriage connection logic, etc...

During the subsequent pricing step, the orchestration engine will also make note of the value contained in ".../OTA_AirPriceRQ/PriceComparison@AmountSpecified" which can then be used to by client applications to compare the actual price being stored during PNR creation against the price gathered during shopping.

After a successful OTA_AirPriceLLSRQ response is received the orchestration engine will extract the value contained in "OTA_AirPriceRS/PricedItineraries/PricedItinerary/AirItineraryPricingInfo/ItinTotalFare/TotalFare@Amount," and return that value along with the initial specified fare amount in the response to allow customers to determine if there was a fare increase between the shopping and booking transaction.

Finally, the client application has the ability to ignore the transaction upon successful processing.

In terms of responses client applications can opt to only receive the flight segments generated as a result of the OTA_AirBookLLSRQ message, or to receive the entire PNR generated as a result of the process via the TravelItineraryRS message.

Segment Status Handling

As mentioned previously, the EnhancedAirBookRQ service has a provision for checking segment status after initial booking to ensure that the air itinerary can be successfully priced. If the segment status result in unconfirmed status code (UC) and only if **@Num Attempts, HaltOnStatus@Code and RetryRebook option="true" is set**, the application attempts to re-book the same O&D in the itinerary with same or different booking class using low fare search with rebooking option (WPNCB). It further re-checks to ensure if any of the segment status equals **HaltOnStatus@Code** and if **@Num Attempts** has been exceeded. Air segments that still result with "UC" or "NN" status cannot be priced.

When "NN" is not specifically mentioned in **HaltOnStatus@Code** it is set by default and the application will Halt. This preventive measure is taken since the segments cannot be priced with "NN" status code.

If only **@Num Attempts** is present in the request, then "NN" is set by default as HaltonStatus code.

To successfully utilize this functionality client applications need to:

1. Set the appropriate segment status codes, i.e. UC, NN, to halt processing via "…/OTA_AirBookRQ/**HaltOnStatus@Code**."
2. Set the appropriate number of times, 1-10, to redisplay the reservation via "…/OTA_AirBookRQ/**RedisplayReservation@NumAttempts**," so that the segment status can be checked. The default wait time between the number of attempts after the first WaitInterval is 1000 millisecond.
3. Set the appropriate wait interval, 0-10000 milliseconds, for the first redisplay via "…/OTA_AirBookRQ/**RedisplayReservation@WaitInterval**," in order to give the carrier an opportunity to respond to the sell message. Some carriers can actually take up to seven seconds to respond to a sell message.
4. If you want the application to rebook the segments to another available class in case of UC in any of the segments, set the value of "…/OTA_AirBookRQ/**RetryRebook@Option**" as "true".
5. Set PostProcessing@IgnoreAfter flag to true if you want service to ignore whole transaction at the end of flow when error or warning encountered. By default value is set to false.

Note: if the carrier responds with "SS" immediately upon initial booking the orchestration engine will override any values set via "…/HaltOnStatus," and "…/RedisplayReservation" and move onto the subsequent operations specified in the request message since SS segments can be priced.

Example:

```
<EnhancedAirBookRQ xmlns="http://services.sabre.com/sp/eab/v3_3">
   <OTA_AirBookRQ>
 <RetryRebook Option="true"/>
      <HaltOnStatus Code="UC"/>
      <OriginDestinationInformation>
```

```
        <FlightSegment ArrivalDateTime="2015-11-13T09:05" DepartureDateTime="2015-11-
13T07:05" FlightNumber="465" NumberInParty="8" ResBookDesigCode="P" Status="NN">
          <DestinationLocation LocationCode="LAS"/>
          <MarketingAirline Code="9W" FlightNumber="1022"/>
          <OriginLocation LocationCode="DFW"/>
        </FlightSegment>
      </OriginDestinationInformation>
      <RedisplayReservation NumAttempts="1" WaitInterval="100"/>
    </OTA_AirBookRQ>
    <OTA_AirPriceRQ>
      <PriceRequestInformation>
        <OptionalQualifiers>
          <PricingQualifiers>
            <PlusUp Amount="100.00"/>
          </PricingQualifiers>
        </OptionalQualifiers>
      </PriceRequestInformation>
    </OTA_AirPriceRQ>
    <PostProcessing IgnoreAfter="false">
      <RedisplayReservation WaitInterval="5"/>
    </PostProcessing>
    <PreProcessing IgnoreBefore="true">
      <UniqueID ID=""/>
    </PreProcessing>
</EnhancedAirBookRQ>
```

In this example the client application has specified to halt subsequent processing if a carrier returns “UC”. Also RetryRebook Option is set as true. For this request, since the sell is not successful (UC), it attempts to re-book the same O&D in the itinerary with same or different booking class using low fare search with rebooking option (WPNCB). The following steps will take place;

- The Application will cancel all the existing Air segments in the itinerary. (If a PNR ID is given in the PreProcessing tag, then the PNR is ignored and retrieved instead of cancellation)
- Rebook all segments in AirBook request with “QF” action code instead of “NN”
- Performs WPNCB entry internally.
- Process the Pricing qualifiers specified by customer in AirPrice request if any.
- Process AirTax request if any.
- Return the response. We can identify if the re-book process was successful by verifying if the value in “…/TravelItineraryReadRS**/RetryRebook@Successful** is “true”. **Note:** This will be available only if the request contains RedisplayReservation tag in the PostProcessing.

Notes:

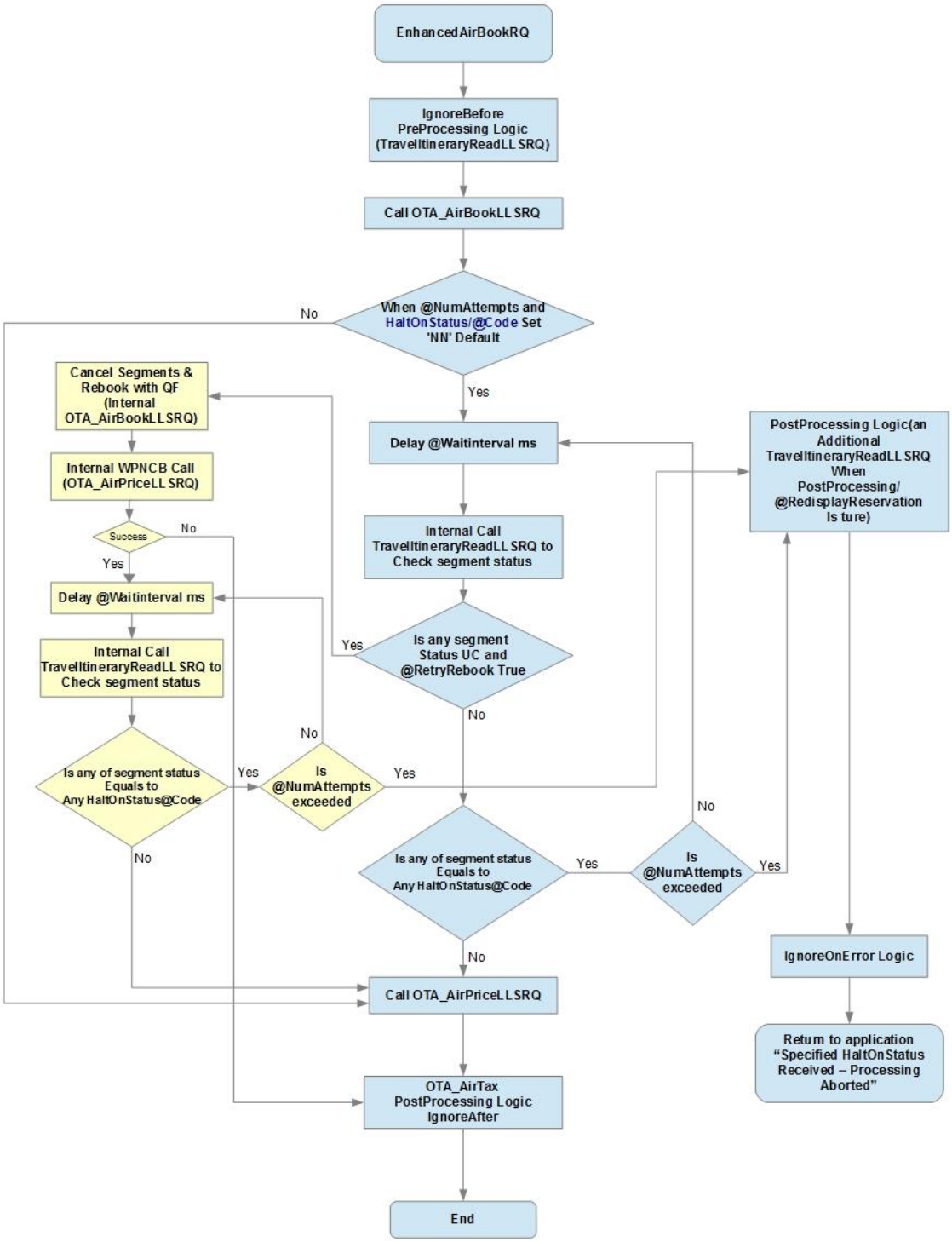
- When IgnoreAfter flag is set to true EnhancedAirBookRQ will ignore segment status passed in the request and sell the segments with QF status. This is a special status that doesn't block airline inventory but it still creates segments in a PNR. This allows pricing to provide a price for itinerary that otherwise could not be booked which could be useful for troubleshooting purposes. It also provides a way to test the service without affecting airlines' inventory.

- If UC is resulted during the five second interval as specified in the above example, it will be rebooked with the lowest fare class available.

- If Pricing (WPNCB) fails for some reason, all segments are returned with QF status code to the customer.
  "…/TravelItineraryReadRS**/RetryRebook@Successful** is "false". **Note:** This will be available only if the request contains RedisplayReservation tag in the PostProcessing.

- If the carrier responds with "SS" at any point during the specified number of seconds (here as 100 in the example) the orchestration engine will override any values set via "…/HaltOnStatus," and "…/RedisplayReservation" and move onto the subsequent operations specified in the request message.

The following diagram shows the control flow for EnhancedAirBookRQ V3.6.0



Support Multiple Pricing

There are scenarios where travel agencies assign different mark-ups or commissions to different fares, depending on the specific passenger type.
For example, an agency is making a booking for one adult (ADT) and one child (CNN). The specific conditions of the fare may allow the agency to assign only 2 percent commission to the child, but no commission to the adult.
EAB currently performs a single OTA_AirPriceLLSRQ call to price and store a fare. In this scenario, agencies may need to call the pricing service twice to price & store two different fares.
In-oder to support this need, EAB has been modified to handle multiple pricing requests and responses, upto a maximum of ten.

Example:

```
<EnhancedAirBookRQ xmlns="http://services.sabre.com/sp/eab/v3_9" IgnoreOnError="true"
HaltOnError="true">
   <OTA_AirBookRQ>
      <OriginDestinationInformation>
         <FlightSegment DepartureDateTime="2018-10-29T12:30:00" FlightNumber="519"
NumberInParty="1" ResBookDesigCode="F" Status="NN">
            <DestinationLocation LocationCode="LAS"/>
            <MarketingAirline Code="AA" FlightNumber="519"/>
            <OriginLocation LocationCode="DFW"/>
         </FlightSegment>
      </OriginDestinationInformation>
   </OTA_AirBookRQ>
   <OTA_AirPriceRQ>
      <PriceComparison AmountSpecified="1400">
           <AcceptablePriceDecrease HaltOnNonAcceptablePrice="true">
            <Amount>20</Amount>
         </AcceptablePriceDecrease>
      </PriceComparison>
      <PriceRequestInformation Retain="true">
         <OptionalQualifiers>
            <FOP_Qualifiers IgnoreStoredFOP="true">
               <BSP_Ticketing>
                  <PayLaterPlan>
                     <Fare Amount="200.00"/>
                     <FOP>
                        <CC_Info>
                           <PaymentCard Code="AX" ExpireDate="2018-11"
ManualApprovalCode="563" Number="372042332191008"/>
                        </CC_Info>
                     </FOP>
                     <Installment Count="03" PayLaterReferenceNumber="XRG065"
Value="10000"/>
                  </PayLaterPlan>
               </BSP_Ticketing>
            </FOP_Qualifiers>
            <MiscQualifiers>
               <MultiTicket Ind="true"/>
            </MiscQualifiers>
            <PricingQualifiers NoDate="true" RoundTheWorld="false">
```

```
            <PassengerType Code="ADT" Quantity="1"/>
            <SpanishLargeFamilyDiscountLevel>1</SpanishLargeFamilyDiscountLevel>
          </PricingQualifiers>
        </OptionalQualifiers>
      </PriceRequestInformation>
    </OTA_AirPriceRQ>
    <OTA_AirPriceRQ>
      <PriceComparison AmountSpecified="1400" HaltOnNonAcceptablePrice="true">>
        <AcceptablePriceIncrease>
          <Amount>20</Amount>
        </AcceptablePriceIncrease>
      </PriceComparison>
      <PriceRequestInformation Retain="true">
        <OptionalQualifiers>
          <FOP_Qualifiers IgnoreStoredFOP="true">
            <BSP_Ticketing>
              <PayLaterPlan>
                <Fare Amount="200.00"/>
                <FOP>
                  <CC_Info>
                    <PaymentCard Code="AX" ExpireDate="2018-11"
ManualApprovalCode="563" Number="372042332191008"/>
                  </CC_Info>
                </FOP>
                <Installment Count="03" PayLaterReferenceNumber="XRG065"
Value="10000"/>
              </PayLaterPlan>
            </BSP_Ticketing>
          </FOP_Qualifiers>
          <MiscQualifiers>
            <MultiTicket Ind="true"/>
          </MiscQualifiers>
          <PricingQualifiers NoDate="true" RoundTheWorld="false">
            <PassengerType Code="ADT" Quantity="1"/>
            <SpanishLargeFamilyDiscountLevel>1</SpanishLargeFamilyDiscountLevel>
          </PricingQualifiers>
        </OptionalQualifiers>
      </PriceRequestInformation>
    </OTA_AirPriceRQ>
    <PostProcessing IgnoreAfter="true">
      <RedisplayReservation WaitInterval="100" UnmaskCreditCard="true"/>
    </PostProcessing>
    <PreProcessing IgnoreBefore="true">
      <UniqueID ID=""/>
    </PreProcessing>
</EnhancedAirBookRQ>
```

Agencies may assign a mark-up or commission tied to a specific fare, if the specific fare they end up pricing with CreatePassengerNameRecord or EnhancedAirBook is lower than what was anticipated. In this case, the mark-up or commission assignment may be incorrect, and therefore agencies need to be able to stop processing when this occurs, as they would need to recalculate or reassign a new mark-up or commission value to the new fare.

In the above example, in EnhancedAirBookRQ,

- When **AcceptablePriceIncrease**/@HaltOnNonAcceptablePrice=" true", EAB halts processing, when the price is higher than the user expected price.

- When **AcceptablePriceDecrease**/@HaltOnNonAcceptablePrice=" true", EAB halts processing, when the price is lower than the user expected price.

Note:

The difference between EnhancedAirBookRS/PriceComparison/@**AmountReturned** and EnhancedAirBookRS/ PriceComparison**/@AmountSpecified** is compared against **EnhancedAirBookRQ/AcceptablePriceIncrease/**@**Amount** (@**Percent**) or **EnhancedAirBookRQ/AcceptablePriceDecrease/**@**Amount** to determine if the price is higher or lower that the user anticipated price.

Interpretation of the Baggage Info

The below sections of the service response pass details pertaining to baggage information.

… /OTA_AirPriceRS/PriceQuote/MiscInformation/BaggageInfo
…/OTA_AirPriceRS/PriceQuote/PricedItinerary/AirItineraryPricingInfo/BaggageProvisions

By linking

…/PriceQuote/MiscInformation/BaggageInfo/SubCodeProperties/@RPH

to the

…/PriceQuote/PricedItinerary/AirItineraryPricingInfo/BaggageProvisions/SubCodeInfo/SubCodeForChargesOthers

one can build the baggage correspondence piece concept vs. weight and how it relates to the segment/leg.

Provision types (*ProvisionType)*:

A - Checked Baggage Allowance
C - Day of Check-in Charges
B - Carry-on Baggage Allowance
CC - Carry-on Baggage Charges
E - Baggage Embargo
P - Prepaid Checked Baggage Charges
EE - Generic Embargo: No Excess Permitted

Example 1:

Here one can see the dimensions of the baggage:

<SubCodeProperties SolutionSequenceNmbr="1" RPH="1">
<AncillaryFeeGroupCode>BG</AncillaryFeeGroupCode>
<BookingMethod>01</BookingMethod>
<CommercialNameofBaggageItemType>UPTO50LB 23KG AND62LI 158LCM</CommercialNameofBaggageItemType>
<DescriptionOne Code="23">
<Text>UP TO 50 POUNDS/23 KILOGRAMS</Text>
</DescriptionOne>
<DescriptionTwo Code="6U">
<Text>UP TO 62 LINEAR INCHES/158 LINEAR CENTIMETERS</Text>
</DescriptionTwo>
<EMD_Type>2</EMD_Type>
<ExtendedSubCodeKey>0GOACLH</ExtendedSubCodeKey>
<RFIC>C</RFIC>
<SizeWeightInfo>
<MaximumSizeInAlternate Units="C">158</MaximumSizeInAlternate>
<MaximumSize Units="I">62</MaximumSize>

<MaximumWeightInAlternate Units="K">23</MaximumWeightInAlternate>
<MaximumWeight Units="L">50</MaximumWeight>
</SizeWeightInfo>
<SSR_Code>XBAG</SSR_Code>
</SubCodeProperties>

Here you see the provision type and segment/leg it relates to. This example the baggage is chargeable and not free allowance

<BaggageProvisions RPH="2">
<Associations>
<CarrierCode RPH="1">LH</CarrierCode>
<CountForSegmentAssociatedID>1</CountForSegmentAssociatedID>
<DepartureDate RPH="1">2016-09-20</DepartureDate>
<DestinationLocation LocationCode="FRA" RPH="1"/>
<FlightNumber RPH="1">921</FlightNumber>
<OriginLocation LocationCode="LHR" RPH="1"/>
<PNR_Segment RPH="1">3</PNR_Segment>
<ResBookDesigCode RPH="1">T</ResBookDesigCode>
<StatusCode RPH="1">SS</StatusCode>
</Associations>
<CarrierWhoseBaggageProvisionsApply>LH</CarrierWhoseBaggageProvisionsApply>
<Commissionable>N</Commissionable>
<FeeApplicationIndicator>4</FeeApplicationIndicator>
<FeeNotGuaranteedIndicator>N</FeeNotGuaranteedIndicator>
<FirstOccurrence>1</FirstOccurrence>
<Interlineable>Y</Interlineable>
<LastOccurrence>1</LastOccurrence>
<PassengerType Code="ADT"/>
<PriceInformation>
<Base Amount="15.00" CurrencyCode="EUR"/>
<Equiv Amount="12.00" CurrencyCode="GBP"/>
<Total>12.00</Total>
</PriceInformation>
<ProvisionType>C</ProvisionType>
<RefundReissue>N</RefundReissue>
<SubCodeInfo>
<SubCodeForChargesOthers>0GOACLH</SubCodeForChargesOthers>
</SubCodeInfo>
</BaggageProvisions>

Example 2:

Here one can see the free baggage allowance

<SubCodeProperties SolutionSequenceNmbr="1" RPH="4">
<AncillaryFeeGroupCode>BG</AncillaryFeeGroupCode>
<CommercialNameofBaggageItemType>FREE BAGGAGE ALLOWANCE</CommercialNameofBaggageItemType>
<EMD_Type>4</EMD_Type>
<ExtendedSubCodeKey>0DFAALH</ExtendedSubCodeKey>

<RFIC>C</RFIC>
</SubCodeProperties>

But based on the provision type A, 0 (zero) is the allowance :

<BaggageProvisions RPH="1">
<Associations>
<CarrierCode RPH="1">LH</CarrierCode>
<CountForSegmentAssociatedID>1</CountForSegmentAssociatedID>
<DepartureDate RPH="1">2016-09-20</DepartureDate>
<DestinationLocation LocationCode="FRA" RPH="1"/>
<FlightNumber RPH="1">921</FlightNumber>
<OriginLocation LocationCode="LHR" RPH="1"/>
<PNR_Segment RPH="1">3</PNR_Segment>
<ResBookDesigCode RPH="1">T</ResBookDesigCode>
<StatusCode RPH="1">SS</StatusCode>
</Associations>
<CarrierWhoseBaggageProvisionsApply>LH</CarrierWhoseBaggageProvisionsApply>
<NumPiecesBDI>0</NumPiecesBDI>
<ProvisionType>A</ProvisionType>
<SubCodeInfo>
<SubCodeForChargesOthers>0DFAALH</SubCodeForChargesOthers>
</SubCodeInfo>
</BaggageProvisions>

Example 3:

Carrier allowed free baggage allowance, number of pieces and the weight/size details

<SubCodeProperties SolutionSequenceNmbr="1" RPH="3">
<AncillaryFeeGroupCode>BG</AncillaryFeeGroupCode>
<CommercialNameofBaggageItemType>FREE BAGGAGE ALLOWANCE</CommercialNameofBaggageItemType>
<EMD_Type>4</EMD_Type>
<ExtendedSubCodeKey>0DFAABA</ExtendedSubCodeKey>
</SubCodeProperties>

<BaggageProvisions RPH="1">
<Associations>
<CarrierCode RPH="1">BA</CarrierCode>
<CountForSegmentAssociatedID>1</CountForSegmentAssociatedID>
<DepartureDate RPH="1">2016-08-20</DepartureDate>
<DestinationLocation LocationCode="DFW" RPH="1"/>
<FlightNumber RPH="1">193</FlightNumber>
<OriginLocation LocationCode="LHR" RPH="1"/>
<PNR_Segment RPH="1">2</PNR_Segment>
<ResBookDesigCode RPH="1">Y</ResBookDesigCode>
<StatusCode RPH="1">SS</StatusCode>
</Associations>
<CarrierWhoseBaggageProvisionsApply>BA</CarrierWhoseBaggageProvisionsApply>
<NumPiecesBDI>1</NumPiecesBDI>

<NumPiecesITR>1</NumPiecesITR>
<ProvisionType>A</ProvisionType>
<SubCodeInfo>
<SubCodeForAllowance RPH="1">0IZACBA</SubCodeForAllowance>
<SubCodeForChargesOthers>0DFAABA</SubCodeForChargesOthers>
</SubCodeInfo>
</BaggageProvisions>

<SubCodeProperties SolutionSequenceNmbr="1" RPH="9">
<AncillaryFeeGroupCode>BG</AncillaryFeeGroupCode>
<BookingMethod>01</BookingMethod>
<CommercialNameofBaggageItemType>BAG MAX 23KG 51LB 208LCM
81LI</CommercialNameofBaggageItemType>
<DescriptionOne Code="23">
<Text>UP TO 50 POUNDS/23 KILOGRAMS</Text>
</DescriptionOne>
<DescriptionTwo Code="6C">
<Text>UP TO 81 LINEAR INCHES/208 LINEAR CENTIMETERS</Text>
</DescriptionTwo>
<EMD_Type>4</EMD_Type>
<ExtendedSubCodeKey>0IZACBA</ExtendedSubCodeKey>
<RFIC>C</RFIC>
<SizeWeightInfo>
<MaximumSizeInAlternate Units="C">208</MaximumSizeInAlternate>
<MaximumSize Units="I">81</MaximumSize>
<MaximumWeightInAlternate Units="K">23</MaximumWeightInAlternate>
<MaximumWeight Units="L">50</MaximumWeight>
</SizeWeightInfo>
<SSR_Code>XBAG</SSR_Code>
</SubCodeProperties>

Change Log

v3.9.0:

- Upgrade OTA_AirPriceLLSRQ to 2.16.0
- Halt EAB when price drops from a specified value
- Support Multiple Pricing Requests/Responses within a single EAB request
- BugFix to not to convert the segment status to “QF” by default, when InstantPurchase=”true” and IgnoreAfter=”true”

v3.8.0:

- Upgrade TravelItineraryReadRQ to 3.8.0
- New feature to book API carriers

v3.7.0:

- Upgrade AirPriceRQ to 2.13.0

v3.6.0

- Upgrade TravelItineraryReadQ to 3.6.0
- Upgrade AirBookRQ to 2.1.0
- MarriageGroup changed from boolean attribute in RQ. New value I/O translated to true/false
- Retry Rebook Feature
- Acceptable Price logic
- Redisplay reservation will be done for pricing errors even when HaltonError is true.

v3.5.0:

- Upgrade to OTA_AirPriceLLSRQ 2.11.0 for the following project:
    - ABACUS-WP WITH SPECIFIC FARE BASIS CODE

v3.2.0

- Upgrade to OTA_AirPriceLLSRQ 2.8
- SubjectArea in TravelItineraryReadRQ for redisplay reservation changed to FULL
- Traffic on PSS has been reduced by using appropriate SubjectArea in other TravelItineraryReadRQ calls

v3.1.0:

- Upgrade to OTA_AirPriceLLSRQ 2.7 for Pricing Round the World Fares with ATPCO (P_88998)
- Update to TravelItineraryReadRQ v3.5
- The TravelItineraryRead request is sent to Open Systems instead of TPF thus returned subset of PNR data may differ.
- Support of unmasking credit card information in theTravelItineraryRead response was introduced. When the request contains /EnhancedAirBookRQ/PostProcessing/RedisplayReservation/@UnmaskCreditCard='true' and a user has EPR keyword CCVIEW then he will be able to see the credit card information in the response

v3.0.0:

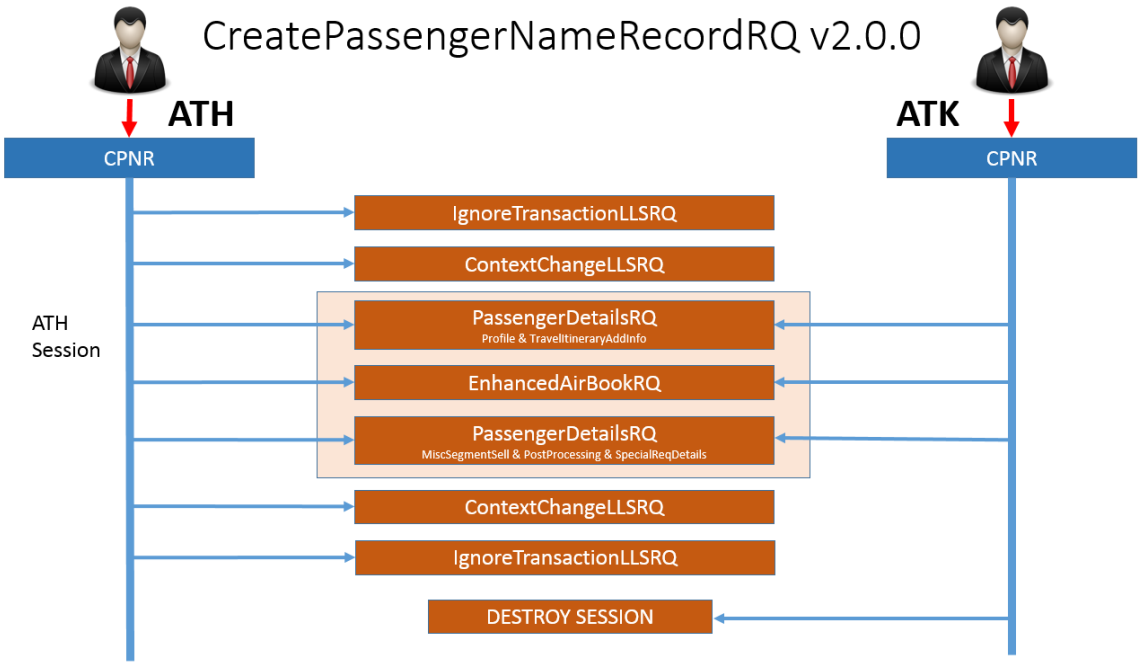
- Service can finish with single success or error and multiple warnings.
- A single HaltOnError flag for whole request. If HaltOnError is false, service merges errors from low level services as warnings and continues processing.
- Updates the service to take advantage of several new, underlying TPF Connector-based service versions
- Fixes logic that reports ProfileRQ successful response as error (SPR-50975).

CreatePassengerNameRecordRQ

https://developer.sabre.com/docs/read/soap_apis/air/book/create_passenger_name_record

The CreatePassengerNameRecordRQ service allows client applications to combine PassengerDetailsRQ and EnhancedAirBookRQ services in one, single call. Its main purpose is to create a full reservation i.e add agency or customer details then perform booking, pricing and finally add any special requests (remarks, SSRs or seats).
It also allows to change target city before operation.

Service flow can be illustrated as below



As you can see the API performs three main steps to complete reservation:

- Add Profile, Passenger, Agency details – 1st call to PassengerDetailsRQ
- Perform booking and pricing – EnhancedAirBookRQ
- Add miscelleaneous segment information, special service details (SSRs, Remarks, etc) and finalize the transaction – 2nd call to PassengerDetailsRQ

HaltOnAirPriceError Flag

Flag defaults to false. The HaltOnAirPriceError attribute controls whether or not the processing of the orchestrated service is stopped if an encountered while a pricing error processes.

Automated handling of hosted vs. non hosted indicators for SSRs/OSIs

Due to the specifics of Sabre system it is necessary to append information whether the SSR/OSI information is to be passed to a hosted or non-hosted carrier. This is controlled within CreatePassengerNameRecord by the below attributes:

/CreatePassengerNameRecordRQ/SpecialReqDetails/SpecialService/SpecialServiceInfo/AdvancePassenger/VendorPrefs/Airline/@Hosted
/CreatePassengerNameRecordRQ/SpecialReqDetails/SpecialService/SpecialServiceInfo/SecureFlight/VendorPrefs/Airline/@Hosted
/CreatePassengerNameRecordRQ/SpecialReqDetails/SpecialService/SpecialServiceInfo/Service/VendorPrefs/Airline/@Hosted

Please note that from version 2.0.0, the API will handle the above flags automatically. Therefore, it is not necessary to include below elements/attributes within the payload:

/CreatePassengerNameRecordRQ/SpecialReqDetails/SpecialServiceRQ/SpecialServiceInfo/AdvancePassenger/VendorPrefs
/CreatePassengerNameRecordRQ/SpecialReqDetails/SpecialServiceRQ/SpecialServiceInfo/SecureFlight/VendorPrefs
/CreatePassengerNameRecordRQ/SpecialReqDetails/SpecialServiceRQ/SpecialServiceInfo/Service/VendorPrefs/Airline/@Hosted

Post booking validation of Airline Record locator and/or HK status for flight segments.

Starting from version 2.1.0, the API is now able to validate airline record locators returned after endtransaction and/or checking if the HK status for flight segments for codeshare flights was modified by carriers after end transaction step.

/CreatePassengerNameRecordRQ/PostProcessing/PostBookingHKValidation requires setting two attributes (*waitinterval* and *numAttempts*) which control the number of times the API will attempt to redisplay reservation after PNR locator has been generated. During each redisplay the API will check if the existing “HK” status for all codeshare flight segments (flights where marketing carrier is different than operating carrier) has changed. Once all redisplay attempts have been completed and at least one flight segment modified its “HK” status, the API will return a warning stating e.g.:

"Flight segment status changed for .../TravelItineraryRead/TravelItinerary/ItineraryInfo/ReservationItems/Item[n]/FlightSegment/@FlightNumber="1234" to "UC".”

/CreatePassengerNameRecordRQ/PostProcessing/WaitForAirlineRecLoc requires setting two attributes (*waitinterval* and *numAttempts*) which control the number of times the API will attempt to redisplay reservation after PNR locator has been generated. During each redisplay the API will check if the airline record locator was added by a carrier. Once all redisplay attempts have been completed and at least one flight segment does not contain full airline record locator, the API will return a warning stating e.g.:

"Missing airline record locators for .../TravelItineraryRead/TravelItinerary/ItineraryInfo/ReservationItems/Item/FlightSegment/@FlightNumber="1234".”

Change Log

V2.1.0

- Added ability to specify multiple pricing instructions (e.g to provide different commission amounts per PQ).
- Added ability to pass Credit Card billing information.
- Added ability to validate airline record locator returned after ending the transaction.
- Added ability to validate whether HK status for flight segments was modified by the airline after ending transaction.
- Updated schema based on the changes within the latest version of EnhancedAirBookRQv3.9.0
- Service also available via REST

V2.0.0

- Introduced REST version
- Added HaltOnAirPriceError flag
- Added automated handling of hosted vs. non hosted indicators

V1.0.0:

- Initial version of service

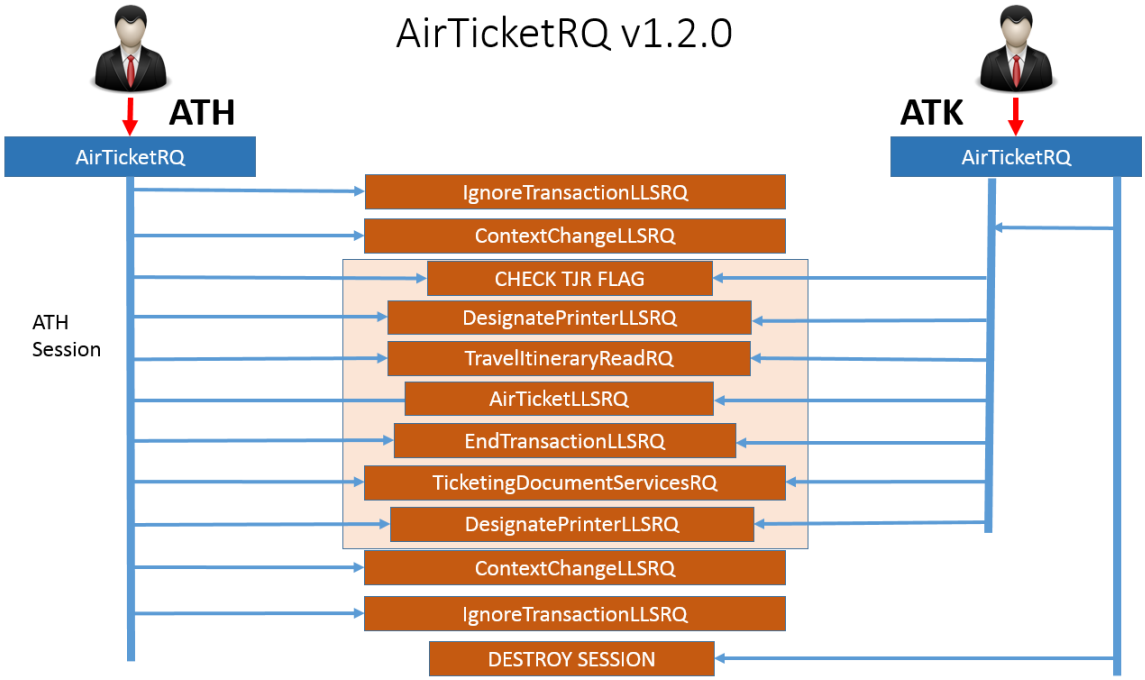
AirTicketRQ

https://developer.sabre.com/docs/read/soap_apis/air/fulfill/orchestrated_air_ticket

The AirTicketRQ service allows client applications to:

- issue multiple Air Tickets and EMDs within a single call,
- issue multiple PQRs in one call,
- Issue more than one PTC in the same transaction with installments,
- integrate printer address designation/un-designation,
- manage sessions on behalf of the client application,
- manage error handling to ensure the successful issuance of an Air Ticket,
- return newly issued ticket numbers together with additional details pertaining to specific documents,
- handle Context change/AAA (modify target city),
- Delete air accounting lines prior to ticketing.

Service flow can be illustrated as below:



As you can see, the API performs a number of steps to issue a single ticket or multiple tickets:

- perform an ignore transaction to ensure that the AAA is clear,
- handle Context change/AAA (modify target city),
- validate the existence of AUTOEND/AUTOER TJR flag,
- designate printers,
- retrieve the existing reservation using record locator provided within the payload

- delete Accounting lines prior to ticketing,
- issue a single ticket (ETR/EMD/PQR) or multiple tickets,
- end transaction – commit the tickets to the face of the PNR (this step is omitted for the users who have AUTOEND/AUTOER TJR flag present)
- retrieve newly generated ticket details,
- undesignate printers,
- perform ignore transaction after the transaction to ensure that the AAA is clear.

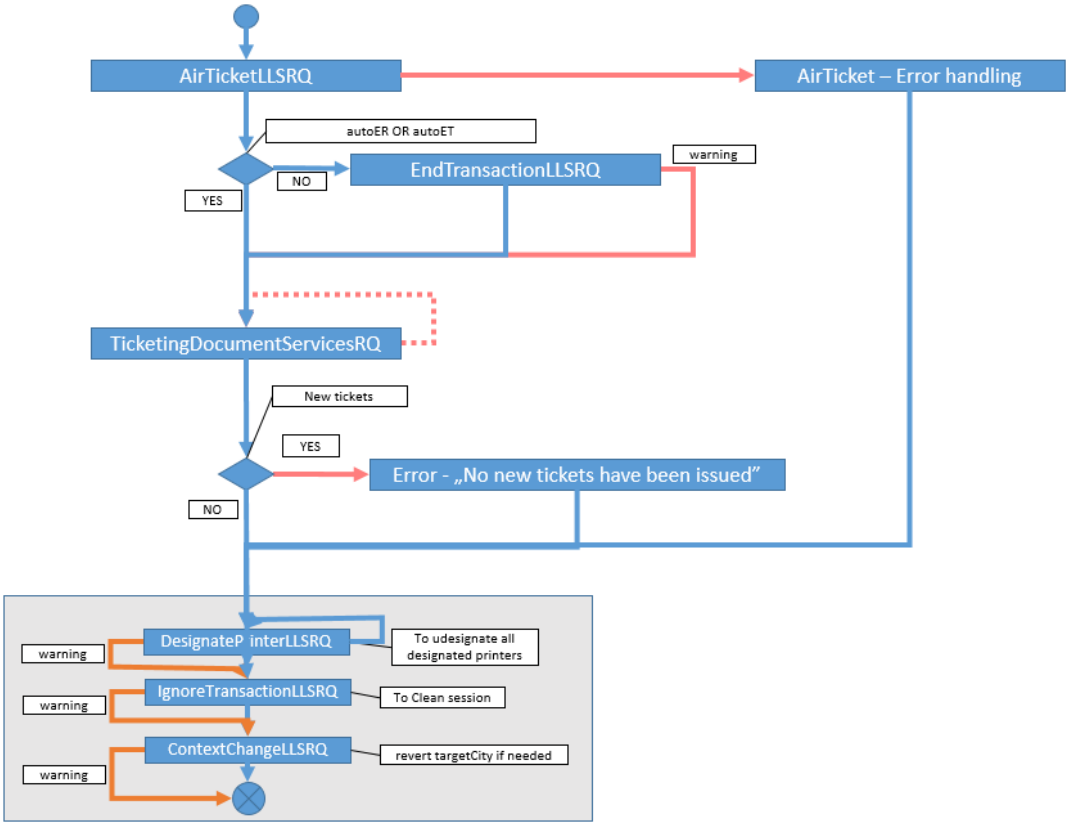
Single call strategy

A single ticketing instruction (single call to AirTicketLLSRQ) is generated by sending a single instance of */AirTicketRQ/Ticketing*.

The API follows below flow:

- single call to low level service AirTicketLLSRQ,
- If the user has below TJR setting on (AUTOEND/AUTOER), issued ticket will
  be automatically commited to the PNR, hence the EndtransactionLLSRQ call
  is omitted,
- API will then attempt to retrieve the newly issued ticket details by calling a low level API TicketingDocumentServicesRQ,
- When all the ticket details have been collected, API will attempt to undesignate all previously designated printers, ignore transaction (to clean the session)
  and revert target city if previously specified within the request payload.

See below diagram for more details:



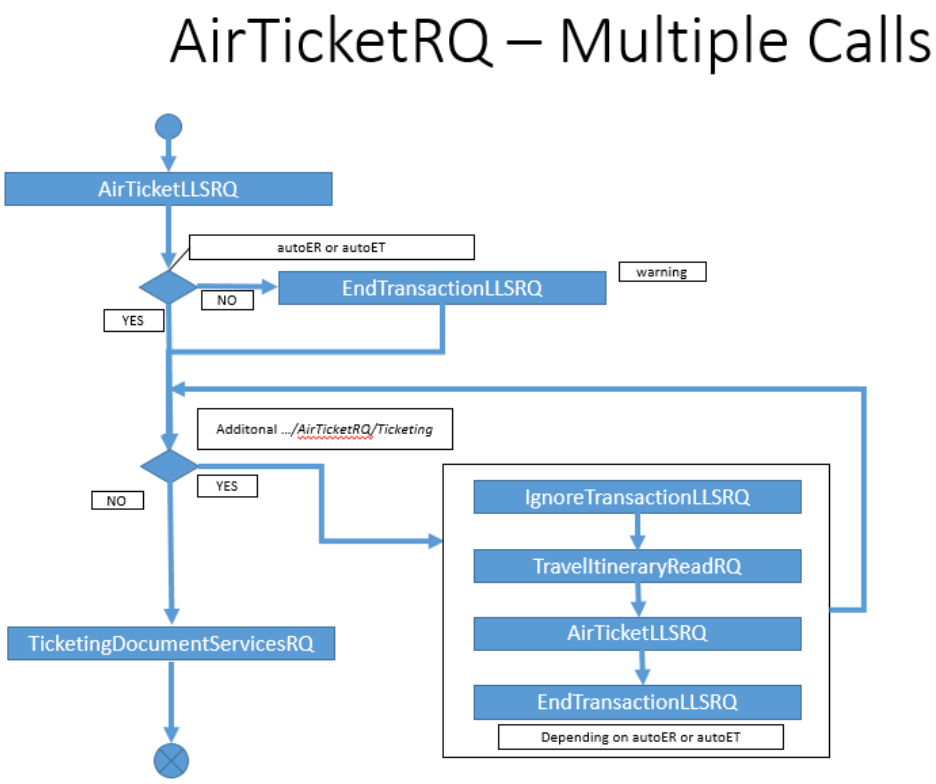
Issuing multiple tickets

When a user wants to issue multiple tickets/ticket types, …*/AirTicketRQ/Ticketing* node needs to be sent within the request payload as many times as many tickets are to be generated. API will then perform a single ticket issuance (as decribed in the above paragraph) and attempt to perform below steps:

- ignore transaction in order to clear the session data,
- retrieve the reservation,
- perform a single call to low level service AirTicketLLSRQ,
- if the user has below TJR setting on (AUTOEND/AUTOER), issued ticket will be automatically commited to the PNR, hence the EndtransactionLLSRQ call will be omitted.

If there are more tickets to be issued, the above steps will be repeated as many times as many …*/AirTicketRQ/Ticketing* are present within the request payload. When the whole process is completed, API will attempt to retrieve the newly issued ticket details by calling TicketingDocumentServicesRQ.

See below diagram for more details:



Handling mask scenarios

There are a number of so-called “mask scenarios” that can be triggered during ticket issuance. The ones that are handled by orchestrated Air Ticket service are:

UNABLE TO TICKET STORED FARE - PQ EXPIRED – handled by */AirTicketRQ/PostProcessing/@actionOnPQExpired*

VERIFY TKT TTL (*) TICKET – handled by */AirTicketRQ/PostProcessing/@acceptPriceChanges*

UNABLE TO TICKET STORED FARE - NEGOTIATED FARE STORED – handled by */AirTicketRQ/PostProcessing/@acceptNegotiatedFare*

Error Handling

- Orchestrated Air ticket will attempt to recover from minor errors but when it encounters a critical error it will throw the below message:

```
<Error type="Application" timeStamp="2018-11-07T03:51:24.808-06:00">
        <SystemSpecificResults>
```

```
                <Message code="ERR.SP.PROVIDER_ERROR">No new tickets have been issued</Message>
                </SystemSpecificResults>
        </Error>
```

The message means that all of the low level AirTicketLLSRQ calls (or one in case of a single ticket transaction) failed – the application was not able to issue any tickets.

Additionally, the orchestration engine will collect error messages coming from low level services to indicate which specific ticketing transaction (a specific instance of */AirTicketRQ/Ticketing*) failed and which low level service triggered it:

```
<AirTicketRS xmlns="http://services.sabre.com/sp/air/ticket/v1">
        <ApplicationResults xmlns="http://services.sabre.com/STL_Payload/v02_01" status="Incomplete">
                <Error type="Application" timeStamp="2018-11-07T01:56:16.357-06:00">
                        <SystemSpecificResults>
                                <Message code="ERR.SP.PROVIDER_ERROR">No new tickets have been
issued</Message>
                        </SystemSpecificResults>
                </Error>
                <Warning type="Application" timeStamp="2018-11-07T01:56:10.731-06:00">
                        <SystemSpecificResults>
                                <Message code="WARN.SP.PROVIDER_WARNING">AirTicketLLS failed for
/Ticketing[1] with Cause: AirTicketLLSRQ: TKT PRT NOT ASSIGNED/USE W*-1496</Message>
                        </SystemSpecificResults>
                </Warning>
        </ApplicationResults>
</AirTicketRS>
```

- There may be another, rare, scenario where all AirTicketLLSRQ calls were successful (all tickets were issued) but the application was unable to receive a successful response from TicketingDocumentServicesRQ (newly created ticket details could not be retrieved). In such case, API will throw below message:

```
        <Error type="Application" timeStamp="2018-11-07T03:51:24.808-06:00">
                <SystemSpecificResults>
                <Message code="WARN.SP.PROVIDER_ERROR">Ticketing was successful but application was
unable to retrieve new ticket details</Message>
                </SystemSpecificResults>
        </Error>
```

The solution for that scenario would be to retrieve reservation using UpdateReservationRQ to check if the new tickets are within the PNR.

Finally, if there were errors coming from low level services that did not prevent the API to collect new ticket details, these errors will be passed within the <*Warning*> tags.

Understanding “Simultaneous Changes” message

It may happen that during the commit stage (when newly issued ticket is in the process of being pushed onto the face of PNR) a warning message “SIMULTANEOUS CHANGES TO PNR - USE IR TO IGNORE AND RETRIEVE PNR” appears. In such a scenario the system does not immediately update the PNR with the ticket data. Yet, if the system is able to retain it, it updates the PNR with the ticket information approximately 15 minutes after the message occurred. Therefore, it is recommended reviewing the agency audit trail (using DailySalesReportLLSRQ) and validating PNR

information (using GetReservationRQ). If the ticket number does not appear in the GetReservationRQ response but it does in the agency audit trail (DailySalesReportLLSRQ response), it is recommended voiding and reissuing the ticket.

Change Log

V1.2.0

- Introduced REST version
- Modified service response schema to pass full ticket details
- Updated to AirTicketLLSRQv2.10.0

V1.1.0

- Modified request schema to simplify process of issuing multiple tickets within one service call

V1.0.0

- Initial version of service

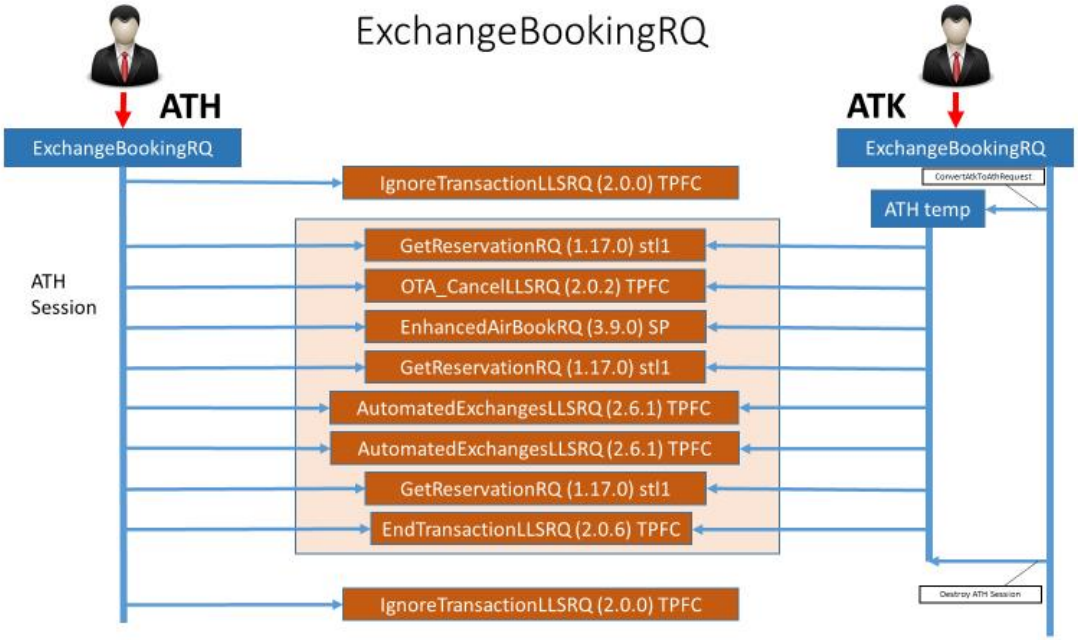
ExchangeBookingRQ

https://developer.sabre.com/docs/soap_apis/air/fulfill/exchange_booking/

Features in the ExchangeBookingRQ service include:

- capability to update the itinerary and create a Price Quote Reissue Record for a ticket exchange in a single API call,
- capability to specify a price tolerance threshold for the Automated Exchanges Comparison response,
- returning details of the generated Price Quote Reissue record,
- managing sessions on behalf of the client application,
- error handling ensuring the successful issuance of a PQR,
- handling Context change/AAA (modify target city),

Service flow can be illustrated as below:



As you can see, the API performs a number of steps to issue a PQR:

- perform an ignore transaction to ensure that the AAA is clear,
- handle Context change/AAA (modify target city),
- retrieve the existing reservation using record locator provided within the payload
- cancel specified PNR air segments,
- book new segments,
- perform automated exchanges comparison call,
- perform automated exchanges confirmation call
- commit newly generated PQR into the PNR
- depending on user input, API can retrieve PQR details and/or retrieve full reservation details,

- perform ignore transaction after the transaction to ensure that the AAA is clear.

Single PQR strategy

A single PQR is generated by sending a single instance of */ExchangeBookingRQ/AutomatedExchanges*.

Depending on the user's choice the API will:

- cancel specific air segments *(/ExchangeBookingRQ/Cancel*),
- add new segments to the reservation (*/ExchangeBookingRQ/AirBook)*
- keep specific segments unchanged *(/ExchangeBookingRQ/Itinerary/SegmentPricing*)

The API will then automatically select segments to be passed onto the Automated Exchanges comparison step using the below logic:

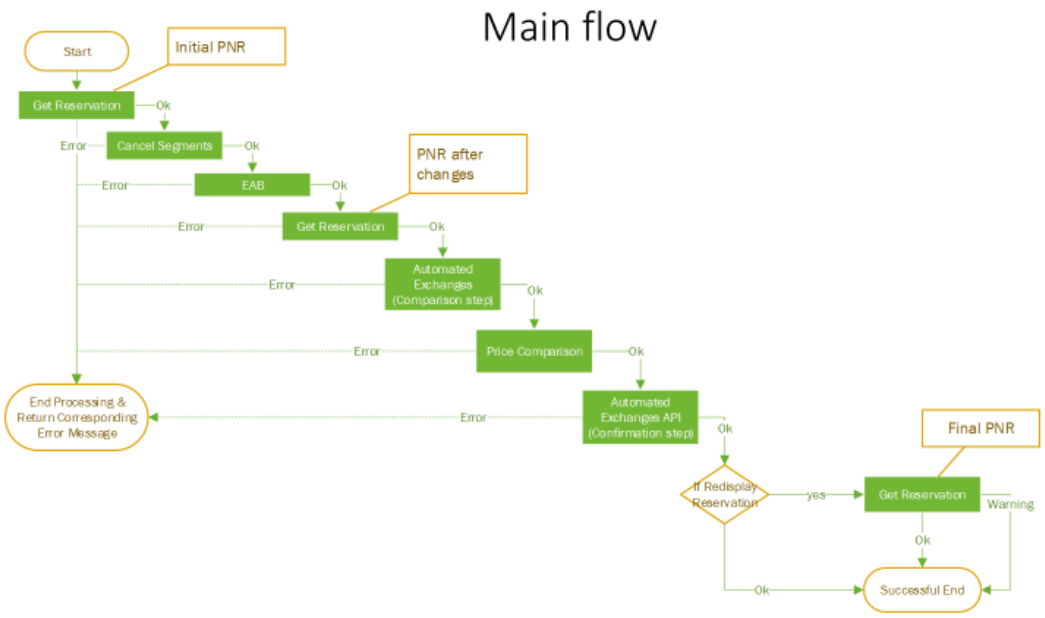
- all newly created air segments during secondary call to reservation service (performed after cancel and airbook steps)
- all segments specified by the user at */ExchangeBookingRQ/Itinerary/SegmentPricing*

If user wishes to validate that the total amount for an exchange matches specific price threshold (based for instance on amounts passed by ExchangeShopping response), such values can be specified at *…/AutomatedExchanges/PriceComparison*.
If the acceptable threshold is not met, ExchangeBookingRQ will stop processing and return a corresponding error message (see Error Handling section on page 42).
If the amount matches the specified threshold, the API will move on to Automated Exchanges confirmation step in order to complete the exchange. Finally, the PQR will be commited to the face of PNR by means of an end transaction call.

Service response will contain the generated PQR number as a default: *(/ExchangeBookingRS/ExchangeConfirmation/@PQR_Number*).

Depending on the user choice, the service response may include full PQR details: *(/ExchangeBookingRQ/PostProcessing/@returnPQRInfo="true")*
or full reservation details
(/ExchangeBookingRQ/PostProcessing/@redisplayReservation="true")

See below diagram, which illustrates the current workflow:



Error Handling

The below table illustrates currently implemented error handling for ExchangeBookingRQ:

Location	Scenario	Error Message
Schema RQ	Customer request schema validation has failed, customer provided an invalid input.	SOAP Fault in response with schema validation details
Request validation	Same segment numbers in SegmentPricing/SegmentSelect/@number and Cancel/Segment/@Number, customer wants to price and cancel the same itinerary segment	ExchangeBookingRS with Business error: ***„/ExchangeBookingRQ/Itinerary/SegmentPricing/SegmentSelect/@Number and /ExchangeBookingRQ/Cancel/Segment/@Number cannot be the same”***
GetReservation – initial call	Error in response or connection issue	ExchangeBookingRS with Business error: ***„Unable to retrieve reservation – see below messages for details”*** *Warnings (Provider error): with details, GetReservation error as warning*
GetReservation – initial call	client chose to price non-air segments	ExchangeBookingRS with Business error: ***„/ExchangeBookingRQ/Itinerary/SegmentPricing/SegmentSelect/@Number=<NUMBER> is not an air segment”***

GetReservation – after cancel and book steps	Error in response or connection issue	ExchangeBookingRS with Business error: *„**Unable to retrieve reservation details after cancel and booking steps – see below messages for details**”* *Warnings (Provider error): with details, GetReservation error as warning*
GetReservation – after cancel and book steps	no new flight segments to price.	ExchangeBookingRS with Business error: *„**No newly created flight segments available for Automated Exchanges pricing after Cancel and AirBook steps**”*
GetReservation – final optional call	Error in response or connection issue	ExchangeBookingRS with Business warning: *„**Unable to redisplay reservation after PQR creation – see below messages for details**”* *Warnings (Provider error): with details, GetReservation error as warning*
Price Comparison	Exchange cost value not returned from AutomatedExchangesLLSRQ Comparison step response.	ExchangeBookingRS with Business error: *„**Unable to perform price comparison - total amount for exchange was not returned by the low level service.**”*
Price Comparison	Price above the threshold specified by customer when haltOnNonAcceptablePrice is **true**	ExchangeBookingRS with Business error: *„**Automated Exchanges comparison TotalRefund amount <VALUE> is over the specified limit for .../ExchangeBookingRQ/AutomatedExchanges[<VALUE>]/PriceComparison**”*
Price Comparison	Price above the threshold specified by customer when haltOnNonAcceptablePrice is **false**	ExchangeBookingRS with Business warning: *„**Automated Exchanges comparison TotalRefund amount <VALUE> is over the specified limit for .../ExchangeBookingRQ/AutomatedExchanges[<VALUE>]/PriceComparison**”*
Price Comparison	Price below the threshold specified by customer when haltOnNonAcceptablePrice is **true**	ExchangeBookingRS with Business error: *„**Automated Exchanges comparison TotalRefund amount <VALUE> is under the specified limit for .../ExchangeBookingRQ/AutomatedExchanges[<VALUE>]/PriceComparison**”*
Price Comparison	Price below the threshold specified by customer when haltOnNonAcceptablePrice is **false**	ExchangeBookingRS with Business warning: *„**Automated Exchanges comparison TotalRefund amount <VALUE> is under the specified limit for .../ExchangeBookingRQ/AutomatedExchanges[<VALUE>]/PriceComparison**”*
AutomatedExchanges – confirmation call	When **PQR_Number** attribute value is not returned by AutomatedExchangesLLSRQ Comparison step response.	ExchangeBookingRS with Business error: *„**Unable to perform automated exchanges confirmation step. Comparison step did not generate PQR Number**”*

Retrieval of Price Quote details	DisplayPriceQuoteLLSRQ error in response or connection issue	ExchangeBookingRS with Business warning: „ “***Unable to retrieve PQR information - see below messages for details***”

Change Log

V1.0.0

- Initial version of service

  Schema includes a number of elements that are not currently supported.

  - */ExchangeBookingRQ/CreditVerificationRQ*
  - Only single repetition of */ExchangeBookingRQ/AutomatedExchanges* is supported
  - */ExchangeBookingRQ/MiscSegmentSell*
  - */ExchangeBookingRQ/SpecialService*
  - */ExchangeBookingRQ/DeleteAccountingLines*
  - */ExchangeBookingRQ/DeletePriceQuote*
  - */ExchangeBookingRQ/PostProcessing/@acceptIncompleteTransactions*

  The content will be available in future versions of the application.

Tools and Artifacts

WSDL and Schema Documentation

Each Sabre SOAP API has a WSDL, schema, and design documents available on Sabre Dev Studio.

Customers can download the latest WSDL and schema documents via Sabre Dev Studio, located at https://developer.sabre.com.

Supporting Documentation

Additional supporting documentation on Sabre Dev Studio includes SOAP API descriptions, sample request and response design XML documents, sample request and response payloads, as well as other service-specific documents.

When a new SOAP API release is deployed to certification or production, updated supporting documentation is also made available via Sabre Dev Studio.

During the customer acceptance testing phase, customers can refer to these documents to make the necessary client code updates to take advantage of the new services/enhancements.

Technical Support

If you have any questions or need assistance, please contact our *Sabre API* Global Customer Support Center.

Telephone:

When reporting production or other critical/time sensitive issues, please contact us via the telephone:

- **USA**: 800-678-9460
- **Canada**: 682-605-5570
- **International**: 598-2-518-6020, or your regional Sabre Software help desk.

Email:

Email is monitored 24 x 7 with a response within 24 hours or less:

- webservices.support@sabre.com

Providing the support desk with the necessary files at the time of initial contact improves our ability to troubleshoot and provide a timely resolution.

In order to better serve you please note the following:

- Please include the Sabre Pseudo City Code (PCC) or Domain where the issue is occurring.
- When reporting an issue with Sabre API Support, input and output payloads are required. Please attach the payloads as separate files, and name them clearly.
- To help ensure that our environment is free of viruses, our policy mandates that all messages received by Sabre from external sources follow special file name guidelines. File names must end in ".sabre.zip" or the zipped attachment will be removed by the e-mail server (for example, “docs.zip” would need to be renamed to “docs.sabre.zip”).
- If your correspondence is regarding a previously reported issue, please include the service incident (“SI”) number in the subject line of your message.