Sabre® APIs

Guide to Accessing and Consuming
Orchestrated Sabre APIs

March, 2016 v3.1.1

Sabre® APIs: Guide to Accessing and Consuming Orchestrated Sabre APIs, March, 2016 v3.1.1

Sabre Inc.

3150 Sabre Drive, Southlake, TX 76092

Tel: 682 605 1000

www.sabre.com

Table of Contents

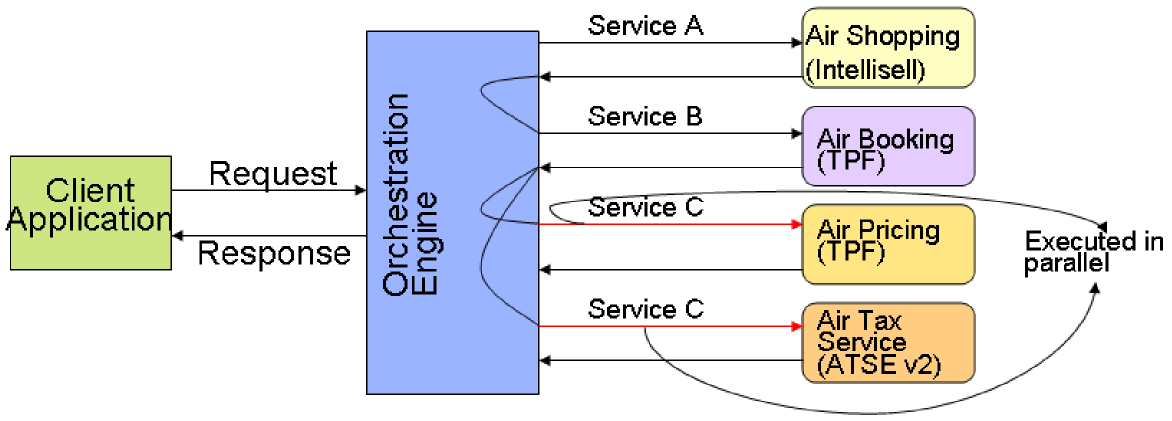
Introduction to Orchestrated *Sabre APIs*

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 presently two 3.x-based orchestrated *Sabre Web Services* available for consumption: EnhancedAirBookRQ and PassengerDetailsRQ.

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).

These services can be utilized together, and they can also be used with existing TPF Connector-based low level services.

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

Sabre's Orchestrated APIs assume that the requesting client application has already authenticated into the *Sabre APIs* infrastructure via a SessionCreateRQ message, and obtained a valid BinarySecurityToken prior to invoking any orchestrated calls.

The consuming application must also be aware that it cannot reuse the BinarySecurityToken for any other request until the orchestrated Web service call completes. If the client application disregards this point and sends requests while the orchestrated service is processing it runs the risk of corrupting the orchestrated transaction and generating errors.

Once the orchestrated call completes the BinarySecurityToken is free to be reused.

Delineating Successful & Failing Transactions

Each 3.x 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

Sabre's Orchestrated APIs 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.

Orchestrated Services Description

PassengerDetailsRQ

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>
```

Change Log

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

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. TravelItineraryReadLLSRQ (JX PNR)
5. OTA_AirPriceLLSRQ (WP)
6. TravelItineraryReadLLSRQ (JX PNR)
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. Air segments that have “UC,” or “NN” status cannot be priced.

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.
3. Set the appropriate wait interval, 0-10000 milliseconds, between redisplays 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. 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_2">
  <OTA_AirBookRQ>
    <HaltOnStatus Code="NN"/>
    <OriginDestinationInformation>
      <FlightSegment DepartureDateTime="2014-06-03T12:30:00" FlightNumber="1022"
NumberInParty="1"
                ResBookDesigCode="F" Status="NN">
        <DestinationLocation LocationCode="LAS"/>
        <MarketingAirline Code="AA" FlightNumber="1022"/>
        <OriginLocation LocationCode="DFW"/>
      </FlightSegment>
    </OriginDestinationInformation>
    <RedisplayReservation NumAttempts="2" WaitInterval="100"/>
  </OTA_AirBookRQ>
</EnhancedAirBookRQ>
```

In this example the client application has specified to halt subsequent processing if a carrier returns “UC,” or “NN” status. The client application has also specified for the system to redisplay the reservation up to four times over the course of six seconds (1500*4) to check segment status.

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 a "UC" is encountered at any point during the six seconds the orchestration engine will immediately halt processing.
- If the carrier responds with "SS" at any point during the six seconds the orchestration engine will override any values set via "…/HaltOnStatus," and "…/RedisplayReservation" and move onto the subsequent operations specified in the request message.
- If the segment status remains "NN" at the end of the six second interval the orchestration engine will halt processing.

Change Log

V3.2.0

- Upgraded OTA_AirPriceLLSRQ to version 2.9
- SubjectArea in TravelItineraryReadRQ for redisplay reservation changed to FULL
- Traffic on PSS has been reduced by using appropriate SubjectArea in other TravelItineraryReadRQ calls

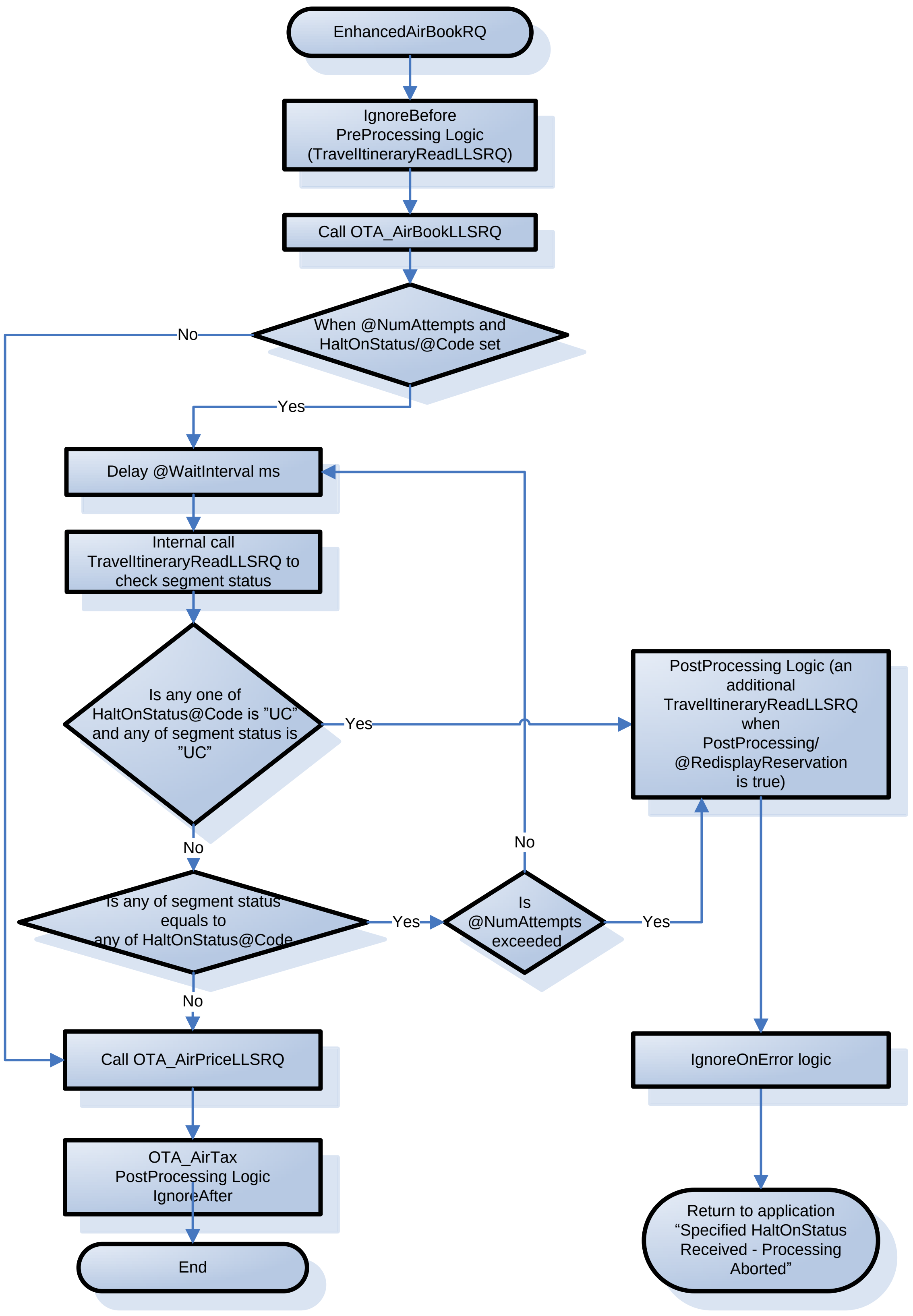
v3.1.0:

- 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 the TravelItineraryRead 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:

Important: In order to invoke this service you must leave /SOAP-ENV:Envelope/SOAP-ENV:Header/eb:Service element empty.

- 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).


EnhancedAirBookRQ
IgnoreBefore PreProcessing Logic (TravelItineraryReadLLSRQ)
Call OTA_AirBookLLSRQ
When @NumAttempts and HaltOnStatus/@Code set
No
Yes
Delay @WaitInterval ms
Internal call TravelItineraryReadLLSRQ to check segment status
Is any one of HaltOnStatus@Code is "UC" and any of segment status is "UC"
Yes
PostProcessing Logic (an additional TravelItineraryReadLLSRQ when PostProcessing/ @RedisplayReservation is true)
No
No
Is any of segment status equals to any of HaltOnStatus@Code
Yes
Is @NumAttempts exceeded
Yes
No
Call OTA_AirPriceLLSRQ
IgnoreOnError logic
OTA_AirTax PostProcessing Logic IgnoreAfter
End
Return to application "Specified HaltOnStatus Received - Processing Aborted"

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.

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>
```