API Developer Notes

Issuing a Ticket on the Apollo CRS

29 June 2012

Version 1.2

Contents

Overview

Ticketing, also known as Document Production, is a complex sequence of events that is influenced by almost all aspects of the booking and mainly applies to air travel. This document explains the basic details of a ticketing approach for online booking applications. It does not cover all aspects of ticketing, as this complicated topic would require much more documentation.

Storing Fares

A properly stored fare sets the stage for the ticketing process, which begins when you store a fare or fares that pertain to air travel. Fares do not need to be stored at the time of booking, and are not required to end transact a PNR. However, stored fares are required for ticketing. Also, because most online applications ticket as soon as practical after booking, it is most efficient to store the fare at the same time as the booking, after selling air segments and requesting seat assignments.

Storing a fare can be a complex process. The various passenger types and modifiers affect the resulting price of a fare, and they must all be correct to obtain the intended price.

Fares are stored in the Automated Ticketing Fare Quote (ATFQ) portions of the PNR. A PNR allows up to eight ATFQs, but the combination of passengers and air segments must be unique for each ATFQ. Most reservations require only one ATFQ. However, there are a number of reasons to have multiple ATFQs in one PNR. For example, multiple ATFQs can occur if different carriers in the itinerary do not have interline e-ticket agreements or have no ticketing agreements at all.

Elements of a Stored Fare

Two processes are invoked when storing a fare in a PNR:

1. A fare quote is generated for the booked itinerary, passengers, and classes of service.

2. The fare quote is saved in the PNR.

Because the stored fare request actually generates a new fare quote, you must verify that the saved fare price matches the expected price, which may have been displayed in an earlier fare quote or in the shopping process. Also, all of the information relevant to the fare must be included in the request, including:

- Descriptions of the passenger types.
- Fare types to be considered (published, private, etc.)
- Private fare information, if needed (Pseudo City Code (PCC), Private Fares account code, etc.)
- Pricing modifiers, such as commission and discounts.

Additional modifiers that do not directly affect the fare price are also required for certain negotiated fares. These modifiers include:

- Ticket designators.
- Endorsement boxes.
- Bulk indicator.
- Net Fare indicator.
- Tour codes.

Simple Stored Fare Format

Use the following format to store a fare if an itinerary is a simple published fare with adult passengers and no other special modifiers:

```
<PNRBFManagement_#>
    <StorePriceMods/>
</ PNRBFManagement_#>
```

Passenger Description

The preceding request assumes that:

1. All of the passengers are adults.
2. Only published fares will be considered.
3. No other modifiers are required.

If an application supports additional passenger types, such as infants, then descriptions of the passengers are required in the request.

The following request includes a passenger description:

```
<StorePriceMods>
    <PassengerType>
        <PsgrAry>
         <Psgr>
            <LNameNum>01</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>01</AbsNameNum>
            <PTC>ADT</PTC>
            <Age><![CDATA[  ]]></Age>
            <PricePTCOnly>N</PricePTCOnly>
         </Psgr>
        </PsgrAry>
    </PassengerType>
</StorePriceMods>
```

Information about each passenger is contained in a<Psgr> element. The system allows passengers to with the same last name to be grouped together; however, from an application standpoint, it is better to treat each passenger independently. To list passengers separately, the <LNameNum> and <AbsNameNum> values increment for each passenger, and the <PsgrNum> is always 01.

To list passengers with the same last name together, the request looks similar to the following request for an adult and an infant without a seat:

```
<StorePriceMods>
    <PassengerType>
        <PsgrAry>
            <Psgr>
                <LNameNum>01</LNameNum>
                <PsgrNum>01</PsgrNum>
                <AbsNameNum>01</AbsNameNum>
                <PTC>ADT</PTC>
                <Age><![CDATA[  ]]></Age>
                <PricePTCOnly>N</PricePTCOnly>
            </Psgr>
            <Psgr>
                <LNameNum>02</LNameNum>
                <PsgrNum>01</PsgrNum>
                <AbsNameNum>02</AbsNameNum>
                <PTC>INF</PTC>
                <Age>01</Age>
                <PricePTCOnly>N</PricePTCOnly>
            </Psgr>
        </PsgrAry>
    </PassengerType>
</StorePriceMods>
```

Used for passenger type code fares.
Y = use PTC filed fares only.

Each passenger type requires an <AssocPsgrs> element, which duplicates the passenger specifications for other processing. As well, if the request contains modifiers, a <PICOptMod> element is required to apply the modifier to the fare. <AssocPsgrs> and <PICOptMod> usually follow the passenger description in the preceding example, and look similar to:

```
<AssocPsgrs>
    <PsgrAry>
        <Psgr>
            <LNameNum>01</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>01</AbsNameNum>
        </Psgr>
    </PsgrAry>
</AssocPsgrs>
<PICOptMod>
    <PIC>ADT</PIC>
</PICOptMod>
```

Adult passenger

```
<AssocPsgrs>
    <PsgrAry>
        <Psgr>
            <LNameNum>02</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>02</AbsNameNum>
        </Psgr>
    </PsgrAry>
</AssocPsgrs>
<PICOptMod>
    <PIC>INF</PIC>
</PICOptMod>
```

Infant passenger

Private Fare Information

Add a <SegSelection> element to the request following the passenger information to quote Private Fares (negotiated fares and Web fares) as well as published fares. <SegSelection> also creates ATFQs for a subset of the entire itinerary. See *Multiple AFTQs,* on *page 11,* for details.

To allow the whole itinerary to use Private Fares (including Web fares), <SegSelection> is formatted similar to:

```
<SegSelection>
    <ReqAirVPFs>Y</ReqAirVPFs>
    <SegRangeAry>
        <SegRange>
            <StartSeg>00</StartSeg>
            <EndSeg>00</EndSeg>
            <FareType>P</FareType>
            <PFQual>
                <CRSInd>1V</CRSInd>
                <PCC>yourPCC</PCC>
                <AirV/>
                <Acct>yourAcctCode</Acct>
                <Contract/>
                <Type>V</Type>
                <PublishedFaresInd>Y</PublishedFaresInd>
                <PFTypeRestrict/>
                <AcctCodeRestrict>N</AcctCodeRestrict>
                <Spare1/>
            </PFQual>
        </SegRange>
    </SegRangeAry>
</SegSelection>
```

00 indicates the entire itinerary

Private Fare information follows

Y = Include published
N = Private Fares only

Private Fare information

Y = all Private Fares
N = Private Fares that match account code only

To apply the Private Fare modifier to the resulting fare, the request must include <PFMod>:

```
<PFMod>
    <PCC>yourPCC</PCC>
    <Acct>yourAcctCode</Acct>
    <Contract/>
</PFMod>
```

To specify the type of fares to be process, the request must include <DocProdFareType>.

```
<DocProdFareType>
    <Type>I</Type>
</DocProdFareType>
```

Possible Values:
I = All available fares.
N = Public fares only.
P = Private fares only (all types).
A = Airline Private Fares only.
G = Agency Private Fares only.

Stored Fare Request

The following example shows a typical stored fare request portion of a PNRBFManagement_# request for an online booking application. The request stores a fare for the booked itinerary with two passengers: one adult and one infant without a seat. It allows both published and Private fares, and expects the entire itinerary fare to be stored in one ATFQ.

Note that the request does not contain special modifiers, which are discussed in *Fare Modifiers* on *page 8*.

```
<StorePriceMods>
    <PassengerType>
        <PsgrAry>
            <Psgr>
                <LNameNum>01</LNameNum>
                <PsgrNum>01</PsgrNum>
                <AbsNameNum>01</AbsNameNum>
                <PTC>ADT</PTC>
                <Age><![CDATA[  ]]></Age>
                <PricePTCOnly>N</PricePTCOnly>
            </Psgr>
            <Psgr>
                <LNameNum>02</LNameNum>
                <PsgrNum>01</PsgrNum>
                <AbsNameNum>02</AbsNameNum>
                <PTC>INF</PTC>
                <Age>01</Age>
                <PricePTCOnly>N</PricePTCOnly>
            </Psgr>
        </PsgrAry>
```

```
</PassengerType>
<AssocPsgrs>
    <PsgrAry>
        <Psgr>
            <LNameNum>01</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>01</AbsNameNum>
        </Psgr>
    </PsgrAry>
</AssocPsgrs>
<PICOptMod>
    <PIC>ADT</PIC>
</PICOptMod>
<AssocPsgrs>
    <PsgrAry>
        <Psgr>
            <LNameNum>02</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>02</AbsNameNum>
        </Psgr>
    </PsgrAry>
</AssocPsgrs>
<PICOptMod>
    <PIC>INF</PIC>
</PICOptMod>
<SegSelection>
    <ReqAirVPFs>Y</ReqAirVPFs>
    <SegRangeAry>
        <SegRange>
            <StartSeg>00</StartSeg>
            <EndSeg>00</EndSeg>
            <FareType>P</FareType>
            <PFQual>
                <CRSInd>1V</CRSInd>
                <PCC>yourPCC</PCC>
                <AirV/>
                <Acct>yourAcctCode</Acct>
                <Contract/>
                <Type>V</Type>
                <PublishedFaresInd>Y</PublishedFaresInd>
```

```
                    <PFTypeRestrict/>
                    <AcctCodeRestrict>N</AcctCodeRestrict>
                    <Spare1/>
                </PFQual>
            </SegRange>
        </SegRangeAry>
    </SegSelection>
    <PFMod>
        <PCC>yourPCC</PCC>
        <Acct>yourAcctCode</Acct>
        <Contract/>
    </PFMod>
    <PlatingAirVMods>
        <PlatingAirV>UA</PlatingAirV>
    </PlatingAirVMods>
    <DocProdFareType>
            <Type>I</Type>
    </DocProdFareType>
</StorePriceMods>
```

Add this element to specify the plating carrier (required on Galileo)

Fare Modifiers

When carriers provide negotiated fares, they often specify exactly how those fares are to be stored. Usually, one or more modifiers must be added to the stored fare request.

Note: Many modifiers print on the ticket.

Private Fares

Many Terminal formats for storing fares start with a format similar to:

```
T:$B-SS004
```

or

```
T:$B-SS004/:P
```

The **–SS004** portion of the format is an account code that indicates a Private Fare, `SS004` can be inserted into <PFQual> and <PFMod> as shown in the following Structured Data sample:

```
<SegSelection>
    <ReqAirVPFs>Y</ReqAirVPFs>
    <SegRangeAry>
        <SegRange>
            <StartSeg>00</StartSeg>
            <EndSeg>00</EndSeg>
```

```
            <FareType>P</FareType>
            <PFQual>
                <CRSInd>1V</CRSInd>
                <PCC>PCC</PCC>
                <AirV/>
                <Acct>SS004</Acct>
                <Contract/>
                <PublishedFaresInd>Y</PublishedFaresInd>
                <Type>A</Type>
            </PFQual>
        </SegRange>
    </SegRangeAry>
</SegSelection>
<PFMod>
    <Acct>SS004</Acct>
    <PCC>PCC</PCC>
    <Contract/>
</PFMod>
```

IT Tour Code

Tour codes, indicated in terminal formats by the inclusion of **`/IT`*`tourCode`***, can also be added to <StorePriceMods>. This value prints on the ticket. The element for a tour code entry is:

```
<TourCode>
    <Rules>tourCode</Rules>
</TourCode>
```

Commission Modifier

A commission modifier overrides the commission table stored in Apollo and Galileo, or adds a commission when no stored information exists. The commission modifier allows you to add a percentage or dollar amount, and is indicated by **`/Z`** in a Terminal format.

The following sample shows a percentage with the percentage set to zero. In <StorePriceMods>, add:

```
<CommissionMod>
    <Percent>00</Percent>
</CommissionMod>
```

The following sample shows a dollar amount. In <StorePriceMods>, add:

```
<CommissionMod>
    <Amt>10.00</Amt>
</CommissionMod>
```

Generic Modifiers

In addition to the Private Fares, IT Tour Code, and Commission modifiers, a number of additional entries are included in many terminal formats. A **/G.** in Terminal Data indicates a generic ticket modifier. In Terminal Data, multiple generic modifiers are separated by the end item character.

Bulk Ticket

Include the following element in <StorePriceMods> to indicate a bulk ticket (Terminal format `GB`):

```
<BulkTicket/>
```

Endorsement Box

Include the following element in <StorePriceMods> to indicate an endorsement box (Terminal format `EB`):

```
<EndorsementBox>
   <Endors1>FEE@FOR@CHANGE</Endors1>
   <Endors2>CANCEL</Endors2>
   <Endors3>VALID@AA@ONLY</Endors3>
</EndorsementBox>
```

Up to three endorsement box entries can be included, using the Endors1, Endors2 and Endors3 tags. Notice that spaces are replaced by the @ character within an endorsement box entry.

Ticket Designator

Include the following element in <StorePriceMods> to indicate a ticket designator (Terminal format `TD`). <SegNum> indicates the segment the designator is related to, and has a <SegNum> of '0' when not related to a specific segment.

```
<TicketDesignator>
   <SegNumAry>
      <SegNumInfo>
         <SegNum>0</SegNum>
         <TkDesignator>ticketDes</TkDesignator>
      </SegNumInfo>
   </SegNumAry>
</TicketDesignator>
```

Some carriers file their fares using a *Net Fare* format, which is equivalent to `:C` in terminal format. Include the following <NetFaresOnly> element in <StorePriceMods> to store a net fare:

```
<GenQuoteInfo>
   <SellCity/>
   <TktCity/>
   <AltCurrency/>
   <EquivCurrency/>
   <TkDt/>
   <BkDtOverride/>
   <EUROverride/>
```

```
            <LCUOverride/>
            <TkType/>
            <AltCitiesRequired/>
            <AltDatesRequired/>
            <NetFaresOnly>Y</NetFaresOnly>
        </GenQuoteInfo>
```

Passenger Type Code (PTC)

While passenger type codes typically describe the passenger type (adult, child, senior, etc.) some carriers use passenger type codes to file specific fares. In Terminal formats, this usage is indicated by:

```
T:$B*DP8
```

In the preceding example, the passenger type code is **DP8**, which indicates a discount of 8%. This PTC must be used in all of the descriptions of the passengers, and requires multiple PTC fields. If only PTC fares are to be considered, then the <PricePTCOnly> element in the passenger description is also set to 'Y'. In Structured Data, the Passenger Type Code is in <PassengerType>.

```
<PassengerType>
    <PsgrAry>
        <Psgr>
            <LNameNum>01</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>01</AbsNameNum>
            <PTC>DP8</PTC>
            <Age/>
            <PricePTCOnly/>
            <DiscOrIncrInd/>
            <AmtOrPercent/>
            <PersonalGeoType/>
            <PersonalGeoData/>
            <TIC/>
        </Psgr>
    </PsgrAry>
</PassengerType>
```

Multiple AFTQs

As noted previously, up to eight ATFQs can be stored in one PNR. Multiple ATFQs are typically stored because more than one carrier is included in the itinerary, and the specific combination of carriers does not have interline E-ticketing agreements, or does not have ticketing agreements of any kind.

Interline ticketing and E-ticketing agreements between carriers vary considerably. Each carrier must be handled separately when storing the fare and ticketing. Essentially, each carrier is stored in a different ATFQ. At ticketing, the ATFQ for each carrier is located and ticketed.

Storing Partial or Multiple ATFQs

Segment selection is used to store an ATFQ for part of the itinerary. or to store multiple ATFQs for a PNR. In the stored price request, the segments that are to be grouped together are specified in the <SegSelection> and <AssocSegs> elements:

```
<SegSelection>
    <ReqAirVPFs>Y</ReqAirVPFs>
    <SegRangeAry>
        <SegRange>
            <StartSeg>01</StartSeg>
            <EndSeg>02</EndSeg>
            <FareType>P</FareType>
            <PFQual>
                <CRSInd>1V</CRSInd>
                <PCC>188I</PCC>
                <AirV/>
                <Acct>yourAcctCode</Acct>
                <Contract/>
                <PublishedFaresInd>Y</PublishedFaresInd>
                <Type>V</Type>
            </PFQual>
        </SegRange>
    </SegRangeAry>
</SegSelection>
<AssocSegs>
    <SegNumAry>
        <SegNum>01</SegNum>
        <SegNum>02</SegNum>
    </SegNumAry>
</AssocSegs>
```

Selects the first two segments of the itinerary

Segment selection is repeated here

If the segments that need to be grouped are not contiguous, multiple entries in the <SegRangeAry> element are needed.

For example, the booked itinerary is:

Outbound:

DL 1790 EWR-ATL and FL 307 ATL-DEN

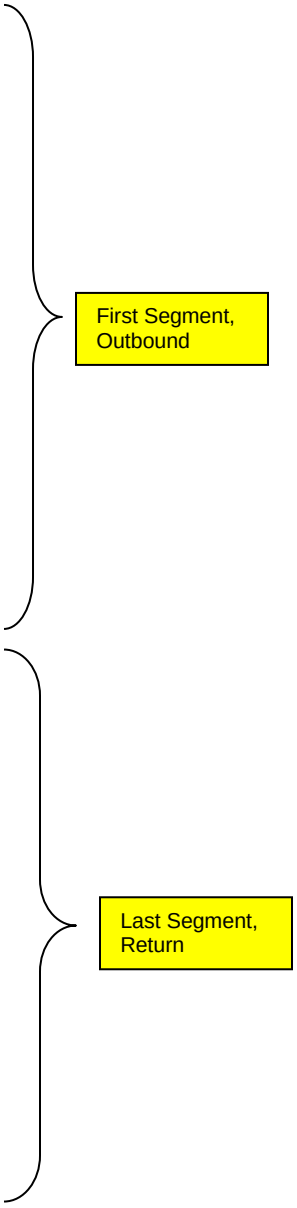
Inbound:

FL 308 DEN-ATL and **DL 1110** ATL-EWR

And the fare is to be stored for the Delta portion of the itinerary (segments 1 and 4). The selection then looks similar to:

```
<SegSelection>
    <ReqAirVPFs>Y</ReqAirVPFs>
    <SegRangeAry>

        <SegRange>
            <StartSeg>01</StartSeg>
            <EndSeg>01</EndSeg>
            <FareType>P</FareType>
            <PFQual>
                <CRSInd>1V</CRSInd>
                <PCC>188I</PCC>
                <AirV/>
                <Acct>yourAcctCode</Acct>
                <Contract/>
                <PublishedFaresInd>Y</PublishedFaresInd>
                <Type>V</Type>
            </PFQual>
        </SegRange>
        <SegRange>
            <StartSeg>04</StartSeg>
            <EndSeg>04</EndSeg>
            <FareType>P</FareType>
            <PFQual>
                <CRSInd>1V</CRSInd>
                <PCC>188I</PCC>
                <AirV/>
                <Acct>yourAcctCode</Acct>
                <Contract/>
                <PublishedFaresInd>Y</PublishedFaresInd>
                <Type>V</Type>
            </PFQual>
        </SegRange>
    </SegRangeAry>
</SegSelection>
<AssocSegs>
    <SegNumAry>
        <SegNum>01</SegNum>
        <SegNum>04</SegNum>
    </SegNumAry>
</AssocSegs>
```



The ATFQ for AirTran (segments 2 and 3) can be stored in the same manner as the Delta example, but storing is not required if a paper ticket is not issued (see *Non-Standard Airline Carriers* on *page 18*).

To verify that the fare has not increased due to the faring of the selected segments, the prices from multiple ATFQs should be totaled and checked against the expected total fare.

Updating a Stored Fare

Ticketing modifiers can be added to an existing stored fare using the DocProdFareRequote_# request. After a PNR is retrieved, modifiers are updated using the appropriate portions of the following request. The DocProdFareRequote_# request adds a commission modifier, a tour code, a ticket designator, endorsement boxes, and a bulk indicator.

```
<DocProdFareRequote_#>
  <UpdateModifiersMods>
    <FareNumInfo>
      <FareNumAry>
        <FareNum>1</FareNum>
      </FareNumAry>
    </FareNumInfo>
    <CommissionMod>
      <Percent>00</Percent>
    </CommissionMod>
    <TourCode>
      <Rules>ABCD</Rules>
    </TourCode>
    <TicketDesignator>
      <SegNumAry>
        <SegNumInfo>
          <SegNum>0</SegNum>
          <TkDesignator>E123</TkDesignator>
        </SegNumInfo>
      </SegNumAry>
    </TicketDesignator>
    <EndorsementBox>
      <Endors1>TEST@FOR@ONETRAVEL</Endors1>
      <Endors2>LINE2</Endors2>
      <Endors3>LINE@THREE@END</Endors3>
    </EndorsementBox>
    <BulkTicket />
  </UpdateModifiersMods>
</DocProdFareRequote_#>
```

The update replaces all of the ticket modifiers. Therefore, to update one modifier element, all of the existing modifiers must be repeated. Any modifier values that that are not repeated will be deleted and replaced with blank values.

Canceling a Stored Fare

To replace a stored fare with a new fare, the existing stored fare must be deleted from the PNR before the new fare is stored. The request to cancel the existing fare is:

```
<PNRBFManagement_#>
    <CancelStoredFareMods>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>01</FareNum>
            </FareNumAry>
        </FareNumInfo>
    </CancelStoredFareMods>
</PNRBFManagement_#>
```

A successful response looks similar to:

```
<PNRBFManagement_#>
    <CancelStoredFare>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
            </FareNumAry>
        </FareNumInfo>
        <DPOK>
        </DPOK>
    </CancelStoredFare>
</PNRBFManagement_#>
```

This indicates success

Canceling Multiple Fares

If a PNR contains more than one fare, all fares can be canceled at the same time with this request:

```
<PNRBFManagement_#>
    <CancelStoredFareMods>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
                <FareNum>2</FareNum>
            </FareNumAry>
        </FareNumInfo>
```

```
    </CancelStoredFareMods>
</PNRBFManagement_#>
```

A successful response to a multiple cancel looks similar to:

```
<PNRBFManagement_#>
    <CancelStoredFare>
      <FareNumInfo>
        <FareNumAry>
          <FareNum>1</FareNum>
        </FareNumAry>
      </FareNumInfo>
      <DPOK>
      </DPOK>
      <FareNumInfo>
        <FareNumAry>
          <FareNum>2</FareNum>
        </FareNumAry>
      </FareNumInfo>
      <DPOK>
      </DPOK>
    </CancelStoredFare>
</PNRBFManagement_#>
```

After the existing fare or fares have been deleted, the new fare is stored as described in *Updating a Stored Fare* on *page* 16.

Verifying a Successful Stored Fare

The response to a stored fare request is the entire set of fare information. Within this response is an indicator that tells you that the fare was stored properly (assuming the fare quote did not generate an error). A value of 'Y' in <FareFiledOKInd> indicates that the fare was successfully stored.

If the fare is not stored successfully, additional indicators within the <FilingStatus> element can help to identify the cause of a stored fare failure. For example:

```
<FilingStatus>
    <UniqueKey>0000</UniqueKey>
    <PsgrMismatchInd>N</PsgrMismatchInd>
    <NeedPlatingCarrierInd>Y</NeedPlatingCarrierInd>
    <NoNamesInd>N</NoNamesInd>
    <OpenSegInd>N</OpenSegInd>
    <TkDtInd>N</TkDtInd>
    <ClassOverrideInd>N</ClassOverrideInd>
    <RetTripInd>N</RetTripInd>
    <NeedRebookInd>N</NeedRebookInd>
    <DecMismatchInd>N</DecMismatchInd>
    <CurrencyMismatchInd>N</CurrencyMismatchInd>
    <AmendedItinInd>N</AmendedItinInd>
    <PseudoItinInd>N</PseudoItinInd>
    <TooManyTaxesInd>N</TooManyTaxesInd>
    <BaseFareTooBigInd>N</BaseFareTooBigInd>
    <BookingDtInd>N</BookingDtInd>
    <PFMismatchInd>N</PFMismatchInd>
    <NotFullGuarInd>N</NotFullGuarInd>
    <FareFiledOKInd>N</FareFiledOKInd>
    <DocProdErrTextInd>N</DocProdErrTextInd>
    <Spare1>NNNNN</Spare1>
    <Spare2>NNNNNNNN</Spare2>
</FilingStatus>
```

Plating carrier needed

Fare did not store

Non-Standard Airline Carriers

Non-standard carriers (“ticketless” carriers and guaranteed payment carriers) do not use the normal ATFQ and ticketing process:

- An ATFQ cannot be stored for any ticketless carrier.
- Most guaranteed payment carriers cannot store an ATFQ; however, if the default plating carrier logic is turned on (`PLAT-Y`), an agent can store an ATFQ for a guaranteed payment carrier.

And, although an ATFQ can be stored for some guaranteed payment carriers, this process fails for other non-standard carriers.

Therefore, the recommended approach for non-standard carriers is to:

- Store ATFQs only for standard carriers.
- Store fare information for non-standard carriers in a remark to facilitate invoice generation and customer service.

Note: Find information for a specific carrier using the **`S*AIR/`*`cc`** format in Focalpoint, where *cc* is the carrier code.

To use the ticketless form of booking for ticketless carriers, the form of payment must be included in an SSR (Special Service Request) associated with a carrier segment in the PNR. The SSR is an item in the Secondary PNR Data section <PNRBFSecondaryBldChgMods>, and looks similar to:

```
<Item>
    <DataBlkInd>S</DataBlkInd>
    <SSRQual>
        <EditTypeInd>A</EditTypeInd>
        <AddQual>
            <SSRCode>OTHS</SSRCode>
            <LNameNum>01</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>01</AbsNameNum>
            <FltNum>0308</FltNum>
            <OpSuf/>
            <AirV>XX</AirV>
            <Dt>20070306</Dt>
            <BIC>R</BIC>
            <StartAirp>DEN</StartAirp>
            <EndAirp>ATL</EndAirp>
            <Text>VI4444333322221111/D1207/TEST JOE</Text>
        </AddQual>
    </SSRQual>
</Item>
```

Must be OTHS

Passenger reference

Carrier

Passenger name

Card number

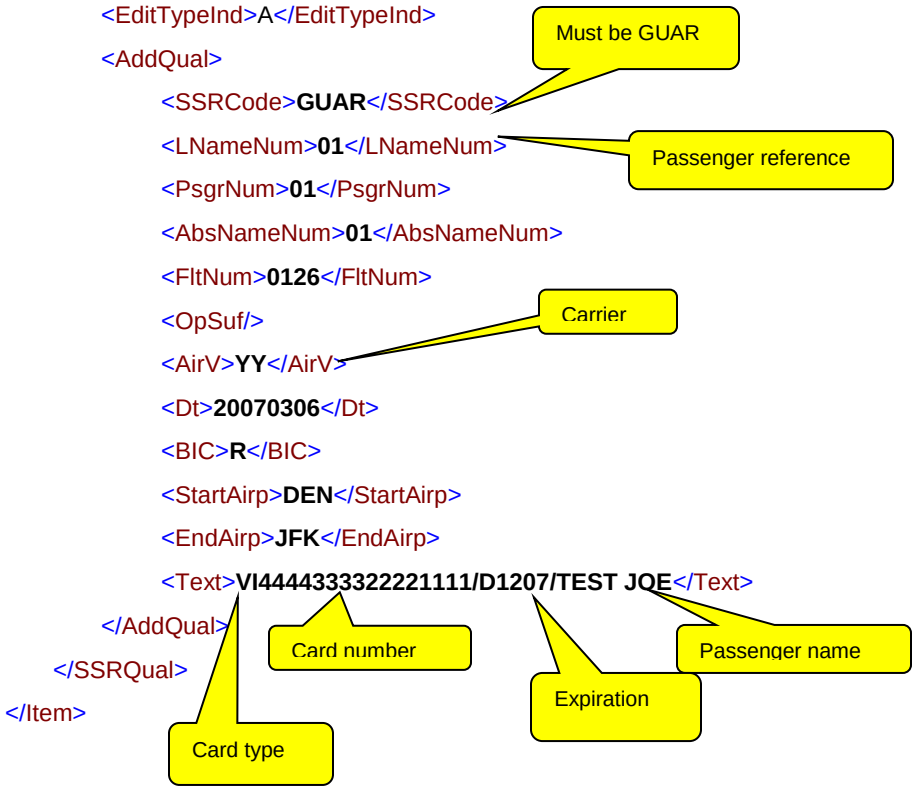
Expiration

Card type

You can store the fare with an itinerary all on one carrier (e.g., XX in the preceding sample), but the form of payment is always the SSR shown above for a ticketless transaction. For carriers that are considered ticketless, there is no need to specifically issue a ticket.

To properly book itineraries on other carriers *before the PNR is end transacted,* a form of payment must be included in an SSR associated with the carrier's segment. The SSR is an item in the PNR Secondary Data section, and looks similar to:

```
<Item>
    <DataBlkInd>S</DataBlkInd>
    <SSRQual>
        <EditTypeInd>A</EditTypeInd>
        <AddQual>
            <SSRCode>GUAR</SSRCode>
            <LNameNum>01</LNameNum>
            <PsgrNum>01</PsgrNum>
            <AbsNameNum>01</AbsNameNum>
            <FltNum>0126</FltNum>
            <OpSuf/>
            <AirV>YY</AirV>
            <Dt>20070306</Dt>
            <BIC>R</BIC>
            <StartAirp>DEN</StartAirp>
            <EndAirp>JFK</EndAirp>
            <Text>VI4444333322221111/D1207/TEST JOE</Text>
        </AddQual>
    </SSRQual>
</Item>
```



Also, it is best not to store a fare for these itineraries, as the stored price request will return a *NO VALIDATING AIRLINE FOUND* error if an agency does not have plating carrier logic (PLAT-N) turned on. If an agency is turned on to plating carrier logic (PLAT-Y), and the itinerary includes only a guaranteed payment carrier, the stored price request returns a fare quote. Ticketing is not allowed.

Ticketing

If the fares are stored properly in a PNR, ticketing is a relatively simple process, although there are numerous opportunities for ticketing errors. The potential for errors makes it difficult to develop a ticketing application that can handle every possible ticketing scenario. The typical approach for an online application is to handle the bulk of the ticketing for the agency environment, and to send ticketing failures to a queue for agent intervention. The sophistication of what can be ticketed depends to a large degree on the sophistication of the logic that stores the fares using the methods described in preceding sections.

The Ticketing Sequence

Ticketing an existing PNR is a multi-step process that requires careful application development. The steps to ticket a PNR are:

1. Start a Host (CRS) session.
2. Retrieve the PNR.
3. Verify that the PNR is ready to ticket.
4. Link to one or more printers.
5. Ticket the PNR.

Ticketing is done within a Host session, and a PNR must be ready for ticketing before a ticketing attempt is made. At a minimum, the PNR must have:

- At least one air segment.
- A valid stored fare.
- A form of payment (FOP).

In addition:

- The PNR must have been end transacted.
- The PNR must not have been ticketed already.
- The PNR must not be in use by any other application.
- A valid ticket printer must be online and ready.

Starting a Host Session

Start a Host session by calling the *BeginSession* method of the XML Select Web Services (XWS). Your Host Access Profile (HAP) is required to access XWS. The *BeginSession* method returns a session token, which is then used for the subsequent session requests (see the Galileo API Developer Notes: *Galileo Web Services Host Sessions* for details).

The SOAP body for this request looks similar to:

```
<soap:Body>
    <BeginSession xmlns="http://webservices.galileo.com">
```

```
        <Profile>yourHAP</Profile>
    </BeginSession>
</soap:Body>
```

If you have established a Web Services proxy stub by using SOAP tools for your environment, the call to that stub looks similar to:

yourStubName.BeginSession(***yourHAP***);

The response is a very long string that is the session token. The SOAP body of the response looks similar to:

```
<soap:Body>
    <BeginSessionResponse xmlns="http://webservices.galileo.com">
        <BeginSessionResult>veryLongTokenStringHere</BeginSessionResult>
    </BeginSessionResponse>
</soap:Body>
```

Your application must save this token for use in additional session requests.

Retrieving the PNR

After starting a Host session, you can work on items stored in a session, such as a new or retrieved PNR. For example, to retrieve a PNR into a session, use the following XML request:

```
<PNRBFManagement_#>
    <PNRBFRetrieveMods>
        <PNRAddr>
            <FileAddr/>
            <CodeCheck/>
            <RecLoc>recordLocator</RecLoc>
        </PNRAddr>
    </PNRBFRetrieveMods>
    <FareRedisplayMods>
        <DisplayAction>
            <Action>I</Action>
        </DisplayAction>
    </FareRedisplayMods>
</PNRBFManagement_#>
```

Retrieves the PNR

This optional section retrieves fare information in the PNR.

Use **I** to retrieve all ATFOs

The PNRBFManagement_# request uses the *SubmitXmlOnSession* method of XWS, not to the non-sessioned *SubmitXML*. The SOAP body for this request is:

```
<soap:Body>
    <SubmitXmlOnSession xmlns="http://webservices.galileo.com">
        <Token> veryLongTokenStringHere </Token>
        <Request>XmlRequestFormatShownAbove</Request>
```

```
        <Filter><_ xmlns=""/></Filter>
    </SubmitXmlOnSession>
</soap:Body>
```

The PNRBFManagement_# request returns the XML representation of the PNR and the ATFQs. The PNR now resides in the session and can be operated on.

If you are using a proxy stub, your method call looks similar to:

yourStubName.SubmitXmlOnSession (***veryLongTokenStringHere,***

XmlRequestFormatShownAbove,

"<_ xmlns=""/>***"***);

All follow on requests also use the *SubmitXmlOnSession* call.

Verifying a PNR is Ready For Ticketing

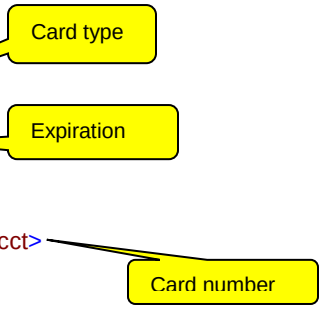
After the PNR is retrieved, it should be examined to verify that it is ready to be ticketed. Before ticketing, check the PNR to confirm:

- A form of payment (FOP).
- A stored fare.
- That the PNR has not been ticketed.
- That the PNR is not in use.
- Vendor confirmations.

Check the PNR for a Form of Payment (FOP)

A PNR cannot be ticketed without a form of payment. The FOP for an online application is usually the traveler's credit card. If an agency is acting as the merchant of record, the form of payment is usually a check. The FOP is commonly entered during the booking process as an item in the Secondary Data section of the PNR. The XML for entering an FOP item is:

```
<Item>
    <DataBlkInd>F</DataBlkInd>
    <FOPQual>
        <EditTypeInd>A</EditTypeInd>
        <AddChgQual>
            <TypeInd>2</TypeInd>
            <CCQual>
                <CC>VI</CC>
                <ExpDt>1206</ExpDt>
                <ExtTxt/>
                <Acct>4444333322221111</Acct>
            </CCQual>
```



```
            </AddChgQual>
        </FOPQual>
    </Item>
```

After retrieving a PNR, the form of payment is contained in either the <CreditCardFOP> element or the <CheckFOP> element. If both of these elements are null, then a form of payment has not been included in the PNR and ticketing will fail.

Check the PNR for a Stored Fare

The PNR must contain a stored fare. To determine if the PNR contains a stored fare, examine the value of the <FareDataExistsInd> element within the <GenPNRInfo> element. A value of 'Y' indicates that at least one fare was stored.

Note: A value of 'Y' does not necessarily mean that PNR is ready to be ticketed, but a value of 'N' indicates that the PNR is **not ready** to be ticketed (see *Fare Verification on page 24* for details).

Confirm that the PNR Was Not Previously Ticketed

There are two elements that indicate a PNR has already been ticketed:

<TkNumExistInd>

<PNRBFTicketedInd>

If the value of these elements is not 'N', which indicates that no ticketing information exists, it is best to end transact the PNR and send it to a queue for manual processing. Do not attempt to continue the ticketing process if either of the two elements listed above are set to 'Y', as the ticketing process will also fail.

Confirm that the PNR is Not in Use

Check that the value of the <InUseInd> element within the <GenPNRInfo> element is 'N' to ensure other systems (typically the carriers involved in the itinerary) are not currently modifying the PNR. If the value of <InUseInd> is not 'N', do not proceed with ticketing.

Check for Vendor Confirmations

Although vendor confirmations are not strictly required for ticketing, issuing a ticket without proper vendor confirmations can result in debit memos from some carriers. Vendor confirmations are located in an element that looks similar to:

```
<VndRecLocs>
    <RecLocInfoAry>
        <RecLocInfo>
            <TmStamp>1735</TmStamp>
            <DtStamp>20061128</DtStamp>
            <Vnd>AA</Vnd>
            <RecLoc>BDIHHR</RecLoc>
        </RecLocInfo>
    </RecLocInfoAry>
</VndRecLocs>
```

Depending on the itinerary, there can be more than one confirmation. The carrier code (<Vnd>**AA**</Vnd>) determines the carrier providing the confirmation. If you do not have a vendor confirmation, it is best not to ticket. You may not have a vendor confirmation because:

- The link was down when the ticket was booked.
- The vendor has not yet built the PNR.
- The ticket was booked with a passive segment, and there was no passive segment notification to the carrier.

Fare Verification

If a PNR is ticketed soon after booking (within approximately 15 minutes), then the fare that was stored is not likely to have changed, and fare verification is not needed. If a substantial amount of time has passed between booking and ticketing, you must ensure that the fare is still valid before proceeding to ticketing by verifying:

- The number of ATFQs.
- Each ATFQ.

Verify the Number of ATFQs

If the itinerary in a PNR has changed after the initial booking, it is possible one or more of the ATFQs is no longer valid. It is also possible that not all segments have stored fares.

To verify that all segments are covered by an ATFQ, the air segments in the PNR must be compared with the segments in the ATFQs. This comparison requires two steps:

1. Count the number of <AirSeg> elements in the PNR.
2. For each ATFQ, in the <DocProdDisplayStoredQuote> element, look in the <AssocSegs> child element to see which segments are included in the ATFQ. Verify that each air segment is included in an ATFQ.

<AssocSegs> looks like:

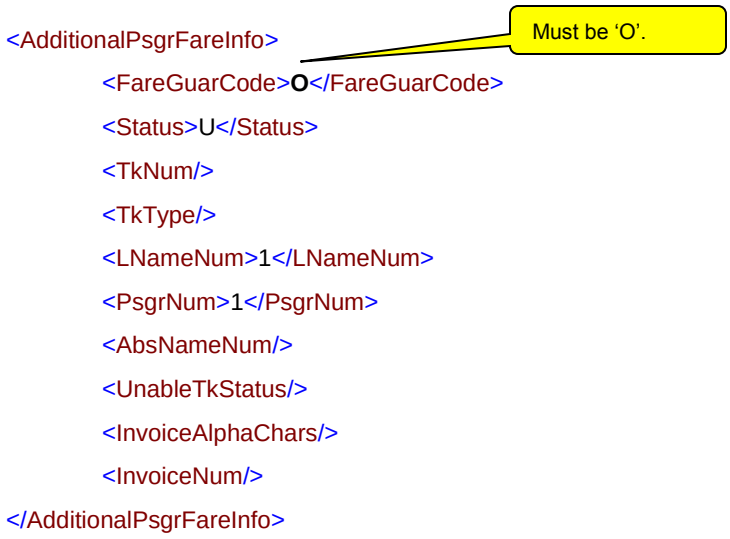
```
<AssocSegs>
    <SegNumAry>
        <SegNum>1</SegNum>
        <SegNum>2</SegNum>
    </SegNumAry>
</AssocSegs>
```

If any segment is missing from the ATFQs, the PNR is not ready to be ticketed. A fare quote for the missing segment is needed, and the new fare must be stored. Because a missing segment in the ATFQs can indicate a more serious problem with the PNR, it is best to send the PNR to a queue for agent intervention.

Verify Each ATFQ

After the PNR has been retrieved, the <FareGuarCode> element for each fare must be reviewed to verify it is set to 'O' for all ATFQs. 'O' indicates all stored fares are still valid and the PNR can be ticketed.

<FareGuarCode> is a child element of <AdditionalPsgrFareInfo> in <DocProdDisplayStoredQuote>. It looks similar to:



A potentially invalid ATFQ is indicated if <FareGuarCode> is not 'O'. An invalid ATFQ can be a sign of a more serious problem with the PNR that requires agent intervention. In the GWS Help, see *XML Select Service* > *Transactions* > *DocProdFareManipulation_#* and in the XML Select Help, see *Transactions* > *DocProdFareManipulation_#.* Review the <FareGuarCode> field description in the response.

If you want to proceed, you can request that the system verify each of the ATFQs. The verify request updates the stored fare with a new price, if the price has changed. It is a good practice to record the existing fare prices before issuing the verify requests, so that you can compare the results after verification.

Note that invalid ATFQs can be caused by a change in the itinerary, e.g., one segment can change from class V to class H. Therefore, an itinerary change can result in a substantially different fare price.

The request looks similar to:

```
<DocProdFareRequote_#>
<DocProdFareRequoteMods>
   <FareNumInfo>
      <FareNumAry>
         <FareNum>001</FareNum>
      </FareNumAry>
   </FareNumInfo>
</DocProdFareRequoteMods>
</DocProdFareRequote_#>
```

If there are multiple ATFQs in the PNR, one request is issued for each fare, even though it looks like multiple <FareNum> entries are allowed in the <FareNumAry> array. A successful response to a verify request looks similar to:

```
<DocProdFareRequote_# xmlns="">
  <DocProdFareRequote>
    <FareNumInfo>
      <FareNumAry>
        <FareNum>1</FareNum>
      </FareNumAry>
    </FareNumInfo>
    <DPOK>
    </DPOK>
  </DocProdFareRequote>
</DocProdFareRequote_#>
```

If a verify request fails, the response looks similar to the following, although many variations of the error message can be returned.

```
<DocProdFareRequote_# xmlns="">
    <DocProdFareRequote>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
            </FareNumAry>
        </FareNumInfo>
        <ErrText>
            <Err>D0000265</Err>
            <KlrInErr>0000</KlrInErr>
            <InsertedTextAry>
    </InsertedTextAry>
            <Text>UNABLE TO UPDATE FARE</Text>
        </ErrText>
        <ErrText>
            <Err>D0000039</Err>
            <KlrInErr>0000</KlrInErr>
            <InsertedTextAry>
    </InsertedTextAry>
            <Text>APPLICATION ERROR</Text>
        </ErrText>
    </DocProdFareRequote>
</DocProdFareRequote_#>
```

If the ATFQ verify is successful, retrieve the PNR again and compare the new fare prices in the ATFQs with the original prices. If the prices are different, it is best to ignore the changes and queue the PNR for agent intervention.

Note: If the stored fare has a <FareGuarCode> value of 'O', but the fare is no longer available or no longer valid, the PNR cannot be ticketed and must be queued for agent intervention.

Printer Linkage

Whether you issue E-tickets or paper tickets, your application must link to a printer. You can display the printers linked to your set or Pseudo City Code (PCC) using the **HMLD*gtid*** or **HMLD*pcc*** terminal formats. To link a ticket printer to the application, issue the TicketPrinterLinkage_# request.

Printer Link Request

```
<TicketPrinterLinkage_#>
    <LinkageUpdateMods>
        <PrinterParameters>
            <LNIATA>yourPrinter#</LNIATA>
            <Type>T</Type>
        </PrinterParameters>
    </LinkageUpdateMods>
</TicketPrinterLinkage_#>
```

Printer Link Response

A successful printer link response looks like the following example, with your LNIATA numbers:

```
<TicketPrinterLinkage_# xmlns="">
    <LinkageUpdate>
        <LNIATAInfo>
            <LNIATA>5EA121</LNIATA>
        </LNIATAInfo>
        <PrinterParameters>
            <LNIATA>F52201</LNIATA>
            <PrinterMode>D</PrinterMode>
            <StockNum/>
            <Type>T</Type>
            <ElectronicPrintInd/>
            <Status>U</Status>
            <ValidationInd/>
            <PCC/>
        </PrinterParameters>
    </LinkageUpdate>
</TicketPrinterLinkage_#>
```

U=Printer up
D=Printer down
B=Printer busy

After you link your PCC to a printer, it should remain linked. However, the link request can be issued without error even if you are already linked. If multiple printers are needed, they can be linked in one request, as shown:

```
<TicketPrinterLinkage_#>
    <LinkageUpdateMods>
        <PrinterParameters>
            <LNIATA>F12345</LNIATA>
            <Type>T</Type>
        </PrinterParameters>
        <PrinterParameters>
            <LNIATA>C12345</LNIATA>
            <Type>I</Type>
        </PrinterParameters>
    </LinkageUpdateMods>
</TicketPrinterLinkage_#>
```

Ticket Printer

Invoice Printer

Changing the Printer Status

If the printer returns a status of ‘D’ (Down), a request can be sent to change the printer status to ‘U’ (Up):

```
<TicketPrinterLinkage_#>
      <LinkageDefinitionMods>
          <ModifyPrinterStatus>
              <LNIATA>024F88</LNIATA>
              <Modifier>U</Modifier>
          </ModifyPrinterStatus>
      </LinkageDefinitionMods>
</TicketPrinterLinkage_#>
```

Changes printer status to Up.

The response should be:

```
<TicketPrinterLinkage_# xmlns="">
      <LinkageDefinition>
          <DPOK>
          </DPOK>
      </LinkageDefinition>
</TicketPrinterLinkage_#>
```

DPOK indicates success.

Issuing Tickets

If the PNR is ready and any printers are linked and ready to print, it is time to ticket. The ticketing request must be issued within the same session that the PNR was retrieved.

Ticketing Request

The DocProdFareManipulation_# request issues a ticket for the first ATFQ in the PNR:

```
<DocProdFareManipulation_#>
    <TicketingMods>
        <ElectronicTicketFailed>
            <CancelInd>Y</CancelInd>
            <IssuePaperTkInd>N</IssuePaperTkInd>
            <IssuePaperTkToSTP/>
            <STPlocation/>
        </ElectronicTicketFailed>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
            </FareNumAry>
        </FareNumInfo>
        <TicketingControl>
            <TransType>TK</TransType>
        </TicketingControl>
    </TicketingMods>
</DocProdFareManipulation_#>
```

Cancel ticket transaction if E-ticket fails – 'Y' or 'N'.

Issue paper ticket if E-ticket fails – 'Y' or 'N'.

First ATFQ.

Ticketing Response

A successful ticketing response looks similar to:

```
<DocProdFareManipulation_# xmlns="">
    <Ticketing>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
            </FareNumAry>
        </FareNumInfo>
        <RecordLocator>
            <RecLoc>yourRecordLocator</RecLoc>
          </RecordLocator>
```

```
        <TextMsg>
            <Txt>ELECTRONIC MESSAGE DELIVERED</Txt>
        </TextMsg>
        <TextMsg>
            <Txt><![CDATA[TKT ISSUED TTL FARE  USD   429.63]]></Txt>
        </TextMsg>
        <TextMsg>
            <Txt>PRICE APPLIES IF TICKETED BY: 10AUG05</Txt>
        </TextMsg>
    </Ticketing>
</DocProdFareManipulation_#>
```

If you have multiple ATFQs in the same PNR, you must ticket each one individually.

Ticketing Error Responses

The ticketing request can fail for any number of reasons, such as printer linkage or processing errors. The first example indicates the error by including the <TransactionErrorCode> element. The last two examples do not include this element, but are still ticketing failures. The key phrase to look for is ELECTRONIC TICKETING TRANSACTION CANCELED (or CANCELLED).

Need Printer Linkage

```
<DocProdFareManipulation_# xmlns="">
    <TransactionErrorCode>
        <Domain>AppErrorSeverityLevel</Domain>
        <Code>1</Code>
    </TransactionErrorCode>
    <Ticketing>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
            </FareNumAry>
        </FareNumInfo>
        <ErrText>
            <Err>D0000042</Err>
            <KlrInErr>0000</KlrInErr>
            <InsertedTextAry/>
            <Text>INVALID ND LINKAGE</Text>
        </ErrText>
    </Ticketing>
</DocProdFareManipulation_#>
```

Unable to Process

```
<DocProdFareManipulation_# xmlns="">
    <Ticketing>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
            </FareNumAry>
        </FareNumInfo>
        <TextMsg>
            <Txt><![CDATA[  VENDOR UNABLE TO PROCESS ETKT - 912-DL]]></Txt>
        </TextMsg>
        <RecordLocator>
            <RecLoc> |*NPB</RecLoc>
        </RecordLocator>
        <TextMsg>
            <Txt>ELECTRONIC TICKETING TRANSACTION CANCELED</Txt>
        </TextMsg>
    </Ticketing>
</DocProdFareManipulation_#>
```

Duplicate Names

```
<DocProdFareManipulation_# xmlns="">
    <Ticketing>
        <FareNumInfo>
            <FareNumAry>
                <FareNum>1</FareNum>
            </FareNumAry>
        </FareNumInfo>
        <TextMsg>
            <Txt><![CDATA[  DUPLICATE NAMES IN PNR-MX]]></Txt>
        </TextMsg>
        <RecordLocator>
            <RecLoc> |*QJ1</RecLoc>
        </RecordLocator>
        <TextMsg>
            <Txt>ELECTRONIC TICKETING TRANSACTION CANCELED</Txt>
        </TextMsg>
    </Ticketing>
</DocProdFareManipulation_#>
```

Ending the Ticketing Process

When all the session transactions are complete, you should end the session. Ending the session releases the GTID that was reserved when the session was started. If modifications were made to a PNR during the session, be sure to End Transact (Finish) the PNR before ending the session.

Ending a Host Session

Use the *EndSession* method of XWS with the session token to end a session. The SOAP body for this method is:

```
<soap:Body>
   <EndSession xmlns="http://webservices.galileo.com">
      <Token> veryLongTokenStringHere </Token>
   </EndSession>
</soap:Body>
```

The response is:

```
<soap:Body>
   <EndSessionResponse xmlns="http://webservices.galileo.com"/>
</soap:Body>
```

Voiding a Ticket

A ticket for a PNR can be voided if a ticket void request is sent after the PNR was ticketed. The sequence to void a ticket is:

1. Start a Host session.
2. Retrieve the PNR.
3. Retrieve the ticket information for the PNR.
4. Request a ticket void for each ticket in the PNR.
5. End the Host session.

Starting a Host Session

Start a Host session by calling the *BeginSession* method of the XML Select Web Services (XWS), as described previously. Again, this request returns a very long string, which is the session token. Your application must save this token to use in additional session requests.

Retrieving the PNR

After you start a Host session, you can retrieve the PNR for which you want to void the ticket. Use the following XML request format to retrieve a PNR into a session:

```
<PNRBFManagement_#>
    <PNRBFRetrieveMods>
        <PNRAddr>
            <FileAddr/>
            <CodeCheck/>
            <RecLoc>recordLocator</RecLoc>
        </PNRAddr>
    </PNRBFRetrieveMods>
</PNRBFManagement_#>
```

Retrieving the PNR Ticket Information

You must know the ticket number or numbers associated with the PNR to void a ticket. To retrieve the ticket information, use the DocProdFareManipulation_# request:

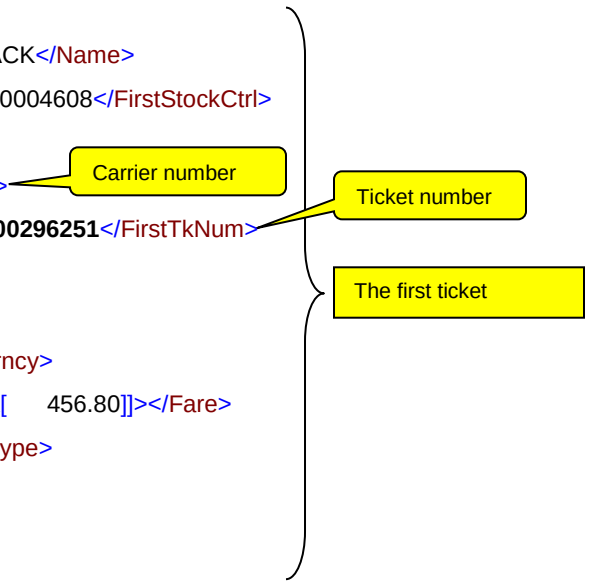
```
<DocProdFareManipulation_#>
    <TicketNumbersMods/>
</DocProdFareManipulation_#>
```

Because you are in a Host session, you do not need to specify the PNR record locator.

The response to the DocProdFareManipulation_# request contains a section for each ticket in the PNR. This section shows the ticket number and a numeric code for the airline, similar to:

```
<DocProdFareManipulation_# xmlns="">
```

```
<TicketNumberData>
  <ETicketNum>
    <Name>TEST/JACK</Name>
    <FirstStockCtrl>00004608</FirstStockCtrl>
    <LastStockCtrl />
    <AirV>422</AirV>
    <FirstTkNum>9900296251</FirstTkNum>
    <LastTkNum />
    <ItinInvNum />
    <Crncy>USD</Crncy>
    <Fare><![CDATA[      456.80]]></Fare>
    <TkType>E</TkType>
    <Dt>07FEB</Dt>
    <Tm>1646</Tm>
  </ETicketNum>
  <ETicketNum>
    <Name>TEST/JILL</Name>
    <FirstStockCtrl>00004609</FirstStockCtrl>
    <LastStockCtrl />
    <AirV>422</AirV>
    <FirstTkNum>9900296252</FirstTkNum>
    <LastTkNum />
    <ItinInvNum />
    <Crncy>USD</Crncy>
    <Fare><![CDATA[      456.80]]></Fare>
    <TkType>E</TkType>
    <Dt>07FEB</Dt>
    <Tm>1646</Tm>
  </ETicketNum>
</TicketNumberData>
</DocProdFareManipulation_#>
```



If the PNR has not been ticketed, the response looks similar to the following code, which indicates that ticket information cannot be returned:

```
<DocProdFareManipulation_# xmlns="">
  <TransactionErrorCode>
    <Domain>AppErrorSeverityLevel</Domain>
    <Code>1</Code>
  </TransactionErrorCode>
  <TicketNumberData>
```

```
      <ErrText>
        <Err>D0000042</Err>
        <KlrInErr>0000</KlrInErr>
        <InsertedTextAry>
        </InsertedTextAry>
        <Text>NO TIN REMARKS EXIST</Text>
      </ErrText>
   </TicketNumberData>
</DocProdFareManipulation_#>
```

Requesting a Void for Each Ticket in the PNR

After you retrieve the ticket information, you can send a void request for each ticket number returned in the response. The request to void the first ticket, based on the example in *Retrieving the PNR Ticket Information,* on *page 33,* is:

```
<TicketVoid_#>
    <VoidTicketMods>
        <TicketNumberRange>
            <AirNumeric>422</AirNumeric>
            <TkStockNum>9900296251</TkStockNum>
        </TicketNumberRange>
    </VoidTicketMods>
</TicketVoid_#>
```

A successful void looks similar to:

```
<TicketVoid_1_0 xmlns="">
  <VoidTicket>
    <DPOK>
    </DPOK>
  </VoidTicket>
</TicketVoid_1_0>
```

DPOK indicates success

To continue with the example in *Retrieving the PNR Ticket Information*, a second void request is needed for the second ticket. The format is the same as above, with a different value for <TkStockNum>.

Voiding Error Responses

If the void is not successful, several errors can be returned.

Ticket Already Voided

```
<TicketVoid_# xmlns="">
   <TransactionErrorCode>
      <Domain>AppErrorSeverityLevel</Domain>
      <Code>1</Code>
```

```
  </TransactionErrorCode>
  <VoidTicket>
    <ErrText>
      <Err>D0008750</Err>
      <KlrInErr>0000</KlrInErr>
      <InsertedTextAry>
      </InsertedTextAry>
      <Text>VOID NOT ACCEPTED TKT ALREADY VOIDED</Text>
    </ErrText>
  </VoidTicket>
</TicketVoid_#>
```

Incorrect Ticket Number or Incorrect Airline Number

```
<TicketVoid_# xmlns="">
  <TransactionErrorCode>
    <Domain>AppErrorSeverityLevel</Domain>
    <Code>1</Code>
  </TransactionErrorCode>
  <VoidTicket>
    <ErrText>
      <Err>D0008720</Err>
      <KlrInErr>0000</KlrInErr>
      <InsertedTextAry>
      </InsertedTextAry>
      <Text>VOID NOT ACCEPTED 13 DIGIT TKT NBR</Text>
    </ErrText>
  </VoidTicket>
</TicketVoid_#>
```

or

```
<TicketVoid_# xmlns="">
  <TransactionErrorCode>
    <Domain>AppErrorSeverityLevel</Domain>
    <Code>1</Code>
  </TransactionErrorCode>
  <VoidTicket>
    <ErrText>
      <Err>D0008722</Err>
      <KlrInErr>0000</KlrInErr>
      <InsertedTextAry>
      </InsertedTextAry>
```

```
        <Text>VOID NOT ACCEPTED TICKET NUMBER NOT FOUND</Text>
      </ErrText>
    </VoidTicket>
</TicketVoid_#>
```

Ending a Host Session

When all the tickets have been voided, you must end the session to releases the GTID that was reserved when the session was started. There is no need to End Transact (Finish) the PNR before ending the session. In GWS, use the *EndSession* method of XWS, with the session token as previously described, to end the session.