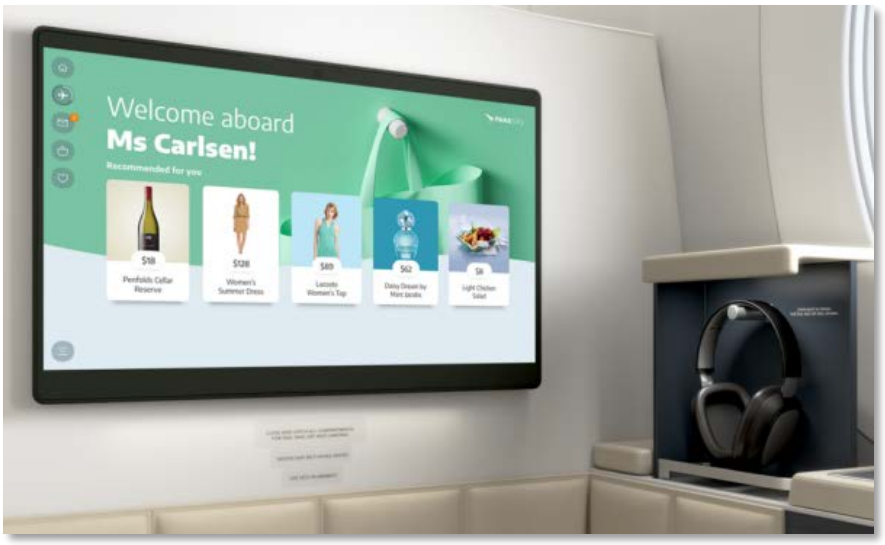
Panasonic

Integrated Personalization Services



Developer Guide for Android Interactives

Document Version: 1.0

Contents

Copyright © 2023 Panasonic Avionics Corporation. All rights reserved. Panasonic is a registered trademark of Panasonic Corporation. Proprietary and Confidential.

Overview

Integrated Personalization Services (IPS) allow the airlines to maintain a profile of each passenger by tracking their preferences and inflight behavior. This allows the airlines to provide tailored services that can boost passenger loyalty and ancillary revenue.

This guide is intended for Interactive developers who work with IPS client APIs through the Android Seatback Application SDK. This supplemental information details the IPS behavior behind the APIs.

Passenger Manifest Data

Passenger manifest data (PMD) is passenger profile information that is transmitted by the customer to the aircraft for each flight. It contains passenger information specific to that flight.

For cell modem transmission, it is typically sent approximately 20 to 40 minutes before the flight departs. For KU transmission, it may be sent during the flight.

Note: The terms PMD and payload are used interchangeably.

Multiple Payloads

Only one payload is stored on the aircraft at any given time. When a new payload arrives, it overwrites the previous payload. IPS does not currently store payloads for future flights.

Active Payload

When a payload arrives to the aircraft, the flight params in the PMD are checked against the flight parameters of the currently open flight.

The three flight parameters are:

- Flight Number – checks only the numeric value
- Flight Origin – consists of the ICAO code
- Flight Destination – consists of ICAO code

All three parameters are checked by default, but this is configurable

Flight Parameters Match

- If the flight parameters match, the new PMD is ingested and the information becomes available to clients.
- The new PMD becomes the active PMD and is the fallback PMD in the event another PMD arrives with flight parameters that do not match.
- The previously active PMD is removed.

Flight Params Mismatch

- If the flight params do not match, the new PMD remains dormant.
- If there was a previously ingested PMD, that PMD continues to be available to clients.
- If there was no previously ingested PMD, the most recently received PMD remains dormant until the flight is opened with matching parameters
- If a third PMD arrives and its flight parameters also do not match the open flight's parameters, it replaces the dormant PMD

User Namespace

The User namespace in the Android Seatback Application SDK supports the personalization features available to the Interactive. The APIs in the User namespace communicate with the IPS Client on the seat.

Develops should use the User v2 APIs which support:

- Class SeatLogin
- Class Playlist, Bookmark, Profile, Preferences, Info and Custom

Class SeatLogin

IPS engagements typically support authentication that restricts all profile fields for a passenger to the client without authentication.

If authentication is not supported, all profile fields for a passenger are available to the client without any authentication.

The following describes the authentication flow and the actions during and after authentication. For additional information on the following APIs, please refer to the SeatLogin Class in the Android Seatback Application SDK.

API	Description
public boolean isLoginReady()	Returns whether login is available and the server is ready for user to login.
public boolean isPdiAvailable()	Returns whether server pdi data available.
public LoginState getLoginState()	Returns login state as LoginState
public String getLoginFieldsNeeded()	Returns seat login fields as a JSON string
public void login(Map<String,String> loginFields, boolean logoutPreviousSeat)	Login from the seat with login fields. Must call getLoginFieldsNeeded() to determine what fields were required for login.
public void logout()	Logs out from the seat.
public void resetLoginFieldsNeeded()	Resets login fields needed.
public void setPdiAvailableChangeListener (SeatLogin.PdiAvailableChangeListener pdiAvailableChangeListener)	Sets PDI available change listener

isPdiAvailable

This API returns the state of the pdi_available field. If the field is false, then login is not possible.

isLoginReady

This API reflects two states:

- the state of the auth_available field
- whether the passenger has exceeded the number of allowed failed logins during a specified time interval

If API returns false, then it is not possible to login.

getLoginFieldsNeeded

IPS software is generic for all deployments. In other words, the same software version is deployed for multiple customers.

However, you can customize how the software behaves through the service config that is appended to the payload by PDI Ground. For additional information, please contact your Panasonic focal.

To login, the Interactive must know the fields the server is expecting for authentication. The getLoginFieldsNeeded API provides the information programmatically.

The service config supports the following use cases:

- Different authentication fields depending on whether the passenger is logging in from a Personal Electronic Device (PED) or Seatback Monitor, or whether the passenger is sitting in First Class or Economy.
- An authentication rollover scheme where the expected login fields change depending on the state of the authentication.

An example response from the getLoginFieldsNeeded API follows:

```
{
    "attemptsRemaining": 3,
    "authFields": {
        "fields": [
            "frequent_flyer.frequent_flyer_number_last_4_digits",
            "travel_info.ticket_number_last_4_digits"
        ],
        "operator": "or"
    },
    "authFieldsPed": {
        "fields": [
            "frequent_flyer.frequent_flyer_number",
            "travel_info.ticket_number"
        ],
        "operator": "or"
```

```
    },
    "hashFields": [
        "frequent_flyer.frequent_flyer_number",
        "frequent_flyer.frequent_flyer_number_last_4_digits",
        "travel_info.ticket_number",
        "travel_info.ticket_number_last_4_digits"
    ],
    "loggedIn": false,
    "loginAvailable": true,
    "pdiAvailable": true,
    "timeRemaining": 0
}
```

Recommendations

- The response includes the pdiAvailable and loginAvailable states. You can optionally call this API instead of isPdiAvailable and isLoginReady APIs.
- The response can contain different auth fields for different classes, such as PED vs Seatback Monitor and First Class vs Economy.
- The response can contain information about the server expecting the auth fields to be hashed.
- The fields attemptsRemaining and timeRemaining have significance if the passenger exceeds the number of allowed failed attempts

Note: The IPS client will not reject an auth request if a field in the request is expected to be hashed and the field is not hashed. However, it will log a warning message.

Duplicate Auth Credentials in PMD

It is important to check the login fields expected by the server when the customer allows the passenger to rollover to a different authentication method.

Consider the following use case:

- A customer decides to use email_address or frequent_flyer_number for authentication.
- A family traveling together uses the same email_address when booking their tickets.
    - This results in multiple passengers having the same email_address in the PMD.
    - When a passenger from the family attempts to login using their email_address, the server sees that there are multiple passengers in the PMD with the same email_address and is unable to authenticate the passenger
- In this case, the customer wants the passenger to retry the login but with email_address and the last 4 digits of the frequent_flyer_number
    - This indicates that the login fields expected by the server for that seat has changed because of the duplicate values found in the PMD.
    - This use case is different from the use case where the passenger enters credentials that don't match any user in the PMD.

The "multiple passengers have duplicate auth fields" use case describes a scenario where the internal auth_state has rolled over from the nominal state of 0 to a new state of 1.

It has no relationship to the "login failed due to no matching auth field" use case which uses the resetLoginFieldsNeeded API.

Note: The Interactive should check this API for the login fields expected by the server, especially if the login fields may change.

Login

The API for authentication is login, where the passenger's input is specified through the map loginFields. For example, if the following auth fields are expected:

```
"authFields": {
    "fields": [
        "frequent_flyer.frequent_flyer_number_last_4_digits",
        "travel_info.ticket_number_last_4_digits"
    ],
    "operator": "or"
},
```

The passenger may authenticate with the last 4 digit of their ticket number. In that case, the map loginFields displays as follows:

```
{"travel_info.ticket_number_last_4_digits":"0ffe1abd1a08215353c233d6e009613e95eec4253832a761af28ff37ac5a150c"}
```

Implicit Seat Swap

Implicit seat swap, if allowed by the service config, indicates that a passenger is allowed to login to a seat to which they are not assigned in the PMD. For example, if a passenger is assigned to seat 1A in the PMD, they can instead login to seat 2C. Their profile transfers from seat 1A to seat 2C as part of the login workflow.

However, a passenger may only be logged into one seat at any given time. In the event a passenger is logged into a seat and attempts to login to a different seat, the login API provides a logoutPreviousSeat parameter to logout from the previous seat as part of the login workflow, where:

logoutPreviousSeat	Description
true	Logs user out from prior seat and logs user into new seat
false	Login fails with message `{ "credentials_in_use": true, "logged_in_seat": "1A" }`

An example of how the parameter can be used:

- If the airline wants to auto-logout the passenger from the previous seat without confirmation, the Interactive should always call the login API with logoutPreviousSeat = true.
- If the airline wants to prompt the passenger to logout of the previous seat before doing so, the Interactive should call the login API with logoutPreviousSeat = false.

  On receiving confirmation from the passenger that they wish to logout of the previous seat, call the login API again with logoutPreviousSeat = true

Explicit Seat Swap

An explicit seat swap is initiated from the crew terminal. In this case, the passenger profiles for each seat are swapped. As part of this workflow, both seats are logged out and the passenger will have to login again.

Note: The behavior where the passenger is logged out during an explicit seat swap was changed in Versions 02.01.xx.xx and later.

resetLoginFieldsNeeded

The resetLoginFieldsNeeded API resets internal auth_state to the nominal state 0. This allows the passenger can login with email_address or frequent_flyer_number again. Without this API, there is no method to return the passenger to that nominal state.

If the customer does not use a rollover scheme for duplicate auth fields, then this API is not relevant to the Interactive and should not be used.

This API is not used to reset the attemptsRemaining and timeRemaining fields which trigger when the passenger attempts to login with the incorrect auth fields multiple times. It is not possible to reset these values with an API. The passenger has to wait for the cool down period to expire before attempting login again.

logout

This API is used when the passenger wants to logout.

pdiAvailableChangeListener

Multiple PMDs to the aircraft. When a new PMD arrives, the Interactive should fetch PDI information again, since it is possible that a passenger's profile has changed. Specifying a listener allows the Interactive to monitor for any PDI updates.

Class Playlist, Bookmark, Profile, Preferences, Info, and Custom

These classes allow the Interactive to get or retrieve and set or alter the passenger profile information for the specified groups or combination of groups at the same time. These APIs can used at any time, either before or after login.

Clear Fields versus Sensitive Fields

The passenger profile information consists of Key Value pairs (KVPs). The majority of the information in the PMD is only retrievable after the passenger has successfully authenticated. However, some KVPs in the PMD are retrievable without a login. These KVPs are known as clear fields.

There is a default declaration of whether each KVP is a clear field or a sensitive field. KVPs can be reclassified from a clear field to a sensitive field or from a sensitive field to a clear field.

If a KVP is reclassified, that information is conveyed to the IPS server through the service config from PDI Ground.

- When the Interactive fetches PDI information before the user has logged in, the server only returns clear KVPs
- When the Interactive fetches PDI information after the user has logged in, the server returns both clear KVPs and sensitive KVPs

There is no parameter for the Interactive to specify whether to fetch clear fields or sensitive fields. The server decides which fields to return based on whether the user is authenticated.

When a passenger logs out, IPS client asks the server for PDI information as part of the workflow. The server returns the clear KVPs for that seat number.

Seat Updates as Anonymous User

While the Interactive can only retrieve clear fields from the PMD for that passenger prior to login, it is able use the same set of KVPs to persist the seat state. This means that the passenger can make a number of changes at the seat, such as:

- Change their movie genre preference
- Add items to the playlist
- Bookmark content after watching part of a movie

and the information can be stored on the IPS headend as an anonymous user at the seat.

To persist the seat state at the headend, the Interactive must use the Android Seatback Application SDK setter APIs to send the information to the IPS client.

In the event of a seat reset or reboot, IPS retrieves all previous settings, made as an anonymous user, for the seat and make them available for retrieval even if the user has never logged into the seat.

These settings are treated as clear fields that can be retrieved without a login, since the settings were made as an anonymous user that was tied to the seat number, not to the passenger.

Handling Seat Updates on IPS Login

If the Interactive uses the Android Seatback Application SDK setter APIs to persist seat state prior to login, then it must decide what to do with the state information when the passenger logs in.

If the Interactive does not use the Android Seatback Application SDK setter APIs to persist seat state prior to login, then this section is not relevant.

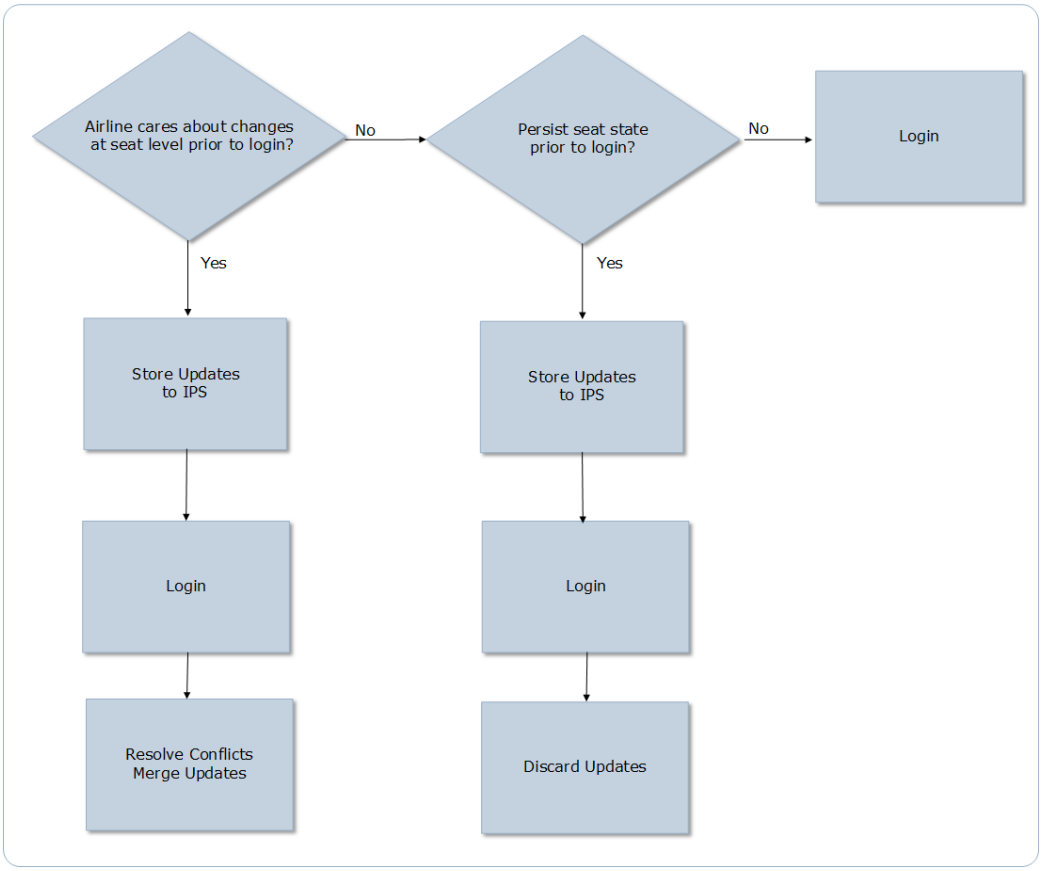
If the changes made at the seat prior to login are not necessary, then:

- It is not necessary to persist seat state. This means no IPS APIs are invoked prior to login.
- or, if seat state is persisted, to discard all changes made at the seat upon IPS login.

If the changes made at the seat prior to login are necessary, then:

- Combine the updates made at the seat prior to login with the passenger profile from the PMD
- In the event of a conflict, there are two choices:
    - Always take the latest update which is the updates prior to login, or
    - Ask the user to decide

The following diagram describes how the INT should function based on the above input



The SDK supports both methods through the ConflictStrategy enum, which allows the Interactive to specify

- KEEP_LATEST
- USER_DEFINED

Playlist and Bookmarks

A playlist is a list of content. A playlist may be created by the passenger or curated by the airline. The following shows a playlist with 1 item.

```
{
  "playlist": {
    "pac_favorites": {
      "items": {
        "ts152200612959410": {
          "uri": "media:3a34492e87db327182e50cee998d1e41",
          "ct": "tvepisode",
          "uri_parent": "media:f3b41392e8d33285a079365941e08968",
          "ts": 1522006129594
        }
      }
    }
  }
}
```

Where

Key	Description
uir	Media URI identifier
ct	Identifies playlist entry content type as tvepisode, movie or track
uri_parent	Media URI identifier for album containing the track or TV series for the episode
ts	Timestamp of when media was added to the playlist. Epoch Time in milliseconds.

A bookmark is a representation of the elapsed time for a previously played content that has not been fully consumed. It has the following information:

```
{
  "bookmarks": {
    "media:c867eff8a67630c7992ecc07e8a868de": [
      {
        "uri_parent": "media:3a34492e87db327182e50cee998d1e41",
        "soundtrack_language": "eng",
        "soundtrack_type": 0,
        "subtitle_language": "eng",
        "subtitle_type": 0,
        "contentType": "movie",
        "last_updated": 1540711750578,
        "elapsed_time": 1059
      }
    ]
  }
}
```

Where

Key	Description
uri_parent	Media URI identifier for album containing the track or TV series for the episode
soundtrack_language	Language of the media soundtrack ISO 639-2 Code
soundtrack_type	Language type of the media soundtrack Values are: 1=commentary 2=dialog 3=description
subtitle_language	Language of the media subtitles ISO 639-2 Code
subtitle_type	Language type of the media subtitles Values are: 1=closed caption

Key	Description
	2=subtitle 3=both 4=embedded 5=none
contentType	Type of content
last_updated	Timestamp of latest bookmark update. Epoch Time in milliseconds.
elapsed_time	Elapsed bookmark time of the media in seconds
cid	Media category from where the bookmarked media was added.

There are maximum number of allowed playlist items and bookmarks. The nominal values are:

- bookmarks: 50
- playlist:
  - movie: 50
  - tvepisode: 100
  - track: 200

But these values can be configured. Older bookmarks or playlist items are pruned when the number exceeds the maximum. However, they are not removed or a deletion of those items in the offload.

When a PMD arrives to the aircraft with playlist or bookmark items, the server scans the bookmarks and playlist as the user logs in.

If the item is not found in the media kit for that flight or if it is not available for that seat class, then it is pruned from the list before it is returned to the caller. The item is only removed, not deleted. This behavior can be disabled through a config.

The effect of not deleting the bookmark is that there is a possibility that older bookmarks or playlist items remain indefinitely in the airline's database for that passenger, if the Interactive does not warn the passenger about an excessive number of items and provide a way for the passenger to delete them.

These older items are uploaded to the aircraft each time the passenger travels again, which increases the size of the payload. Ideally, the airline should delete playlist and bookmark items from passenger lists after a period of time, such as 1 year, since media content is not available indefinitely on future flights.

Recommendation Namespace

The recommendation service is a separate service from IPS, but it is packaged with the IPS loadable. As a result, it can operate independently of IPS and/or make recommendations based on IPS information.

Class Recommendation

The following APIs are available through the Recommendation class

API	Description
getSimilarMedia	Returns items similar to the URIs or MIDs you directly provide to it as parameters.
getTrendingMedia	Returns the items currently/recently watched by passengers on the flight
getRecommendations	Returns similar items based on your viewing history/playlist, which Recommender pulls from IPS service
getEnsembleMedia	Returns the getRecommendaitons and getTrendingMedia items. This call saves the network from multiple calls and avoids duplicate items. Items do not display for both recommended and trending.

Recommendation Algorithm Details

Recommender uses content based recommendation by:

- Gathering all movies that the passenger has watched or added to a playlist
- Weighing their effect on the score based on recency+percentage watched
- Building a content feature map for the passenger, which has a mapping of features to weights.

Example: If a passenger watches a lot of Bill Murray movies, that passenger's content feature map displays a large weight for actor:bill_murray, genre:comedy, country_of_origin:usa

Best Practice

Authentication

- **isPdiAvailable** and **isLoginReady**

  The Interactive should hide or disable the login screen to the passenger when isPdiAvailable or isLoginReady is false.

  This prevents the passenger from entering credentials when the system is not ready to accept them.

- **Auth Failures**

  When the passenger exceeds the number of allowed failed attempts, the server rejects all subsequent login requests for a cool down period. This occurs even if the auth fields are correct.

  Ideally, the Interactive should disable the login screen to the passenger during the cool down period. This prevents the passenger from repeatedly trying to login.

  At a minimum, the Interactive should indicate to the passenger why their login is rejected. Example: wrong auth field values versus locked out period.

  The number of failed attempts and the duration of the cool down period are configurable parameters.

Persistent Store

Decide up front whether your program intends to support persistent store.

Based on that decision, you may need to setup additional prompts to the user on login or just keep latest changes as the default.

If you use persistent store, but do not resolve conflicts on login, the user will not be able to save or make changes after login.