NOI A.G. / S.p.A.
Roberto Cavaliere
r.cavaliere@noi.bz.it
T +39 0471 066 676

SkyAlps Data Collector

v2.1, 16.12.2022

Preliminary notes

SkyAlps is an Italian airline operator managing flights at the Bolzano airport in South Tyrol. Thanks to the support of NOI, SkyAlps has initiated an innovation process that aims to share the data of the air services offered.

The first set of data which is shared is related to the planned timetable of the flights offered, which is made available through a machine-readable API, i.e. through the AeroCRS hub, to which SkyAlps is connected. The reference methods that will be used are the following:

- **Planned data**: https://docs.aerocrs.com/reference/getschedule
- **Fare data**: https://docs.aerocrs.com/reference/getfares

The access credentials have been made available by SkyAlps. After a more detailed evaluation of the data retrieved by the getschedule service, it has been noted that in case of changes in the schedules these are visible only through the SSIM format. Therefore, the data should be requested in this way and not in the JSON format. From a correct decoding of the SSIM format, please check this specification: https://www.slots-austria.com/jart/prj3/sca/uploads/data-uploads/downloads/e)%20Miscellanious/overall/SCR,%20SIR%20Quick%20Guide.pdf

As far as the **real-time data** is concerned, an additional API has been made available. The end-point is:

https://dataprovider.ifly.aero:8443/fidsdataproviderproxy/dataProvider.ashx

The access to the IP is filtered on the base of the source IP, and the access to the data also requires a token in the HTTP parameter. The API provides real-time and scheduled data, but since reference scheduled data are retrieved from the AeroCRS API, only real-time data is considered from this method.

SSIM format decoding specifications

A single data record is characterized by a pattern as follows:

```
3 BN 19520101J10NOV2210NOV22   4    BZO09000900+0100  DUS10451045+0100  DH4
BN 1952                           Y76                   000003
```

The main fields to be considered are:

- **flight_number**: in the example “BN 1952”
- **date_of_operation**: in the example “10NOV22” (the corresponding week day is provided as number, in this case ‘4’, since November 10th 2022 is a Thursday; the week days not considered are not set)
- **departure_airport_code**: in the example “BZO”
- **departure_time**: in the example “0900”, meaning 09:00. Please note that times are provided in UTC format
- **arrival_airport_code**: in the example “DUS”
- **departure_time**: in the example “1045”, meaning 10:45. Please note that times are provided in UTC format

The data should be retrieved from the API so that for each calendar data the single flights are returned. If a certain time interval is requested, than the API could return the different days in which a flight is scheduled, for example:

```
3 BN 19500201J16NOV2225NOV22 3 5  BZO08000800+0100 BER09500950+0100 DH4                                BN
1950              Y76           000024
```

In this case there are two flights to be considered, namely November 16th (Wednesday) and November 25th (Friday). In other word, a separate flight information should be considered for explicit week day reported (in this case associated to ‘3’ and ‘5’). In order to avoid this complex mapping, the solution is to interrogate the API for each calendar day.

Fare data information

As already mentioned, the fare data can be retrieved through a separate service, documented at https://docs.aer-ocrs.com/reference/getfares

Basically, by giving in input a certain time interval, it is possible to get all prices associated to the routes planned in the given time interval. The prices are not associated to specific flights, but to the routes, i.e. to a combination of airport A to airport B (which could also include transfer flights!). These prices have to be intended as fare indications, so not as exact prices associated to the flights.

The list of fields provided are:

Parameter	Type	Description
count	Number	number of fares found
airlineDesignator	String	Airline Designator
airlineICAOcode	String	Airline 3 letter ICAO code
airlinename	String	Airline name
fromCode	String	Destination FROM code
toCode	String	Destination TO code
fromDate	String	Fare flight date range start
toDate	String	Fare flight date range end
classes	Array	List of class codes and the quantity available for each one
adultFareRT	Money	Fare for Adult (Round trip)
childFareRT	Money	Fare for Child (Round trip)
infantFareRT	Money	Fare for Infant (Round trip)
tax1RT	Money	Tax (Round trip)
tax2RT	Money	Tax (Round trip)
tax3RT	Money	Tax (Round trip)
tax4RT	Money	Tax (Round trip)
adultFareOW	Money	Fare for Adult (One Way)
childFareOW	Money	Fare for Child (One Way)
infantFareOW	Money	Fare for Infant (One Way)
tax1OW	Money	Tax (One Way)
tax2OW	Money	Tax (One Way)
tax3OW	Money	Tax (One Way)
tax4OW	Money	Tax (One Way)
chargeTaxOnReturnTrip	String	When searching for a RT, this will state if we charge the tax on both legs, first leg only or second leg only
notification	String	Fare notification

Real-time data information

The API provides a JSON file with a list of flights divided into departures (DEP) and arrivals (ARR). Actually the API is not very self-descriptive, and the provided fields need an explanation. For the departures, following fields are provided:

- F = Flight Code
- EX = Expected Time
- SC = Scheduled Time
- D = Destinations List (multilanguage list)
- A = Airline Code
- S = Status Code (see explanation below)
- C = Checkin Code
- G = Gate Code
- GI = Gate information Time
- T = Terminal
- DC = Flag of delay. 0 = no flag. 1 = early flight, 2 = delayed flight

For the arrivals, following fields are provided:

- F = Flight Code
- EX = Expected Time
- SC = Scheduled Time
- D = Destinations List (multilanguage list)
- A = Airline Code
- S = Status Code (see explanation below)
- B = Belt Code
- T = Terminal
- Flag of delay. 0 = no flag. 1 = early flight, 2 = delayed flight

The status codes follow this convention:

- B = Boarding
- U = Last Call
- Z = Boarding Closed
- C = Check-In Opened
- D = Departed
- L = Landed
- G = Gate Number
- Y = Diverted
- X = Cancelled
- R = Baggage claim
- K = Check-In Closed

Integration in the Open Data Hub as Data Collector

METADATA

The proposal is to insert all this static data in the “station” table.

Web-service fields	Open Data Hub parameters
flight_number_date_of_operation	stationcode
departure_airport_code - arrival_airport_code	name

Table 1: Mapping between main web-service and Open Data Hub fields (reference: “station” table).

The following specifications have to be also considered:

- the Open Data Hub field **origin** is to set as **SKYALPS**.
- the Open Data Hub field **stationtype** is to set as **Flight**
- since the API does not provide any information about the positions of the aiports, the proposal is to hardcode this information for all flights, assigning the position of the airport of Bolzano
  - **latitude**: 46.46248
  - **longitude**: 11.32985
- the Open Data Hub field stationcode is created on top of two fields, flight_number and date_of_operation (e.g. ‘BN 1952_10NOV22’). In this way each single flight flying at a certain day will be considered as an Open Data Hub station. This means that a certain flight number will have multiple entries in the stations table, one for each day in which the flight is scheduled.
- the other fields provided by the web-service (i.e.: airlinename, airlinedesignator, airlineid, std, sta, weekdaysun, weekdaymon, weekdaytue, weekdaywed, weekdaythu, weekdayfri, weekdaysat, accode, fltsfromperiod, flt-stoperiod) have to be considered as **metadata** (i.e. will be stored in the “metadata” table of the Open Data Hub-database). The entire SSIM message could be saved as a separate field in the metadata;
- two specific fields are added, namely **arrival_timestamp** and **departure_timestamp**, which provide departing and arrival date and time of the flights, respectively as UNIX timestamps. Thanks to these fields it is possible to filter through the Open Data Hub API all flights in a certain time interval.

METADATA (FARES)

The proposal is to foresee a separate call to the fare method and to append additional fare information as additional metadata, as follows:

- separate sub-structure called “**Fares**” with all the above fields (except the first one, count);
- additional field “**minimumPriceRoundTrip**”, mapped with the API field adultFareRT;
- additional field “**minimumPriceOneWay**”, mapped with the API field adultFareOW;

The latter two values aim to provide the information that the fares are indicative, and they should be treated as such by 3rd parties. On the other side, we provide the full raw datasets retrieved by AeroCRS.

DATA

The proposal is to associate the real-time data to the corresponding stations and to store it in the **measurementjson (measurementjsonhistory)** table, so that the entire set of information is saved as provided. The proposal is to map the codes used in the API with the corresponding descriptions provided above, so that the data structure can also be easily understood by a human being.

An important task that needs to be checked is the mapping between real-time and already available planned data. This should be implemented through the Flight Code (F) attribute, which should not stored again since it is already present as metadata. During implementation it is to be understood the temporal scope of the flights returned, the assumption here is that basically only the flights on the given day are returned, making this mapping quite simple to implement.