

Package ‘openSkies’

April 17, 2021

Type Package

Title An R Package to Retrieve, Analyze and Visualize Air Traffic Data

Version 1.1.1

Date 2021-04-17

Author Rafael Ayala, Daniel Ayala, David Ruiz, Aleix Sellés, Lara Sellés Vidal

Maintainer Rafael Ayala <rafael.ayala@oist.jp>

Description Provides functionalities and data structures to retrieve, analyze and visualize aviation data. It includes a client interface to the 'OpenSky' API <<https://opensky-network.org>>. It allows retrieval of flight information, as well as aircraft state vectors.

Acknowledgements The development of this software is supported by the Spanish Ministry of Science and Innovation (grant code PID2019-105471RB-I00) and the Regional Government of Andalusia (grant code P18-RT-1060).

License CC BY-NC 4.0

Depends grid

Imports httr, ssh, xml2, ggmap, ggplot2, magick, utils, stats, R6, dbscan, cluster

Suggests knitr, BiocStyle, RUnit, BiocGenerics, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Encoding UTF-8

Repository CRAN

Date/Publication 2021-04-17 04:10:02 UTC

R topics documented:

ADSBDecoder	2
clusterRoutes	3
getAircraftFlights	5
getAircraftMetadata	7

getAircraftStateVectorsSeries	8
getAirportArrivals	9
getAirportDepartures	11
getAirportMetadata	12
getIntervalFlights	13
getRouteMetadata	14
getSingleTimeStateVectors	15
getVectorSetFeatures	17
getVectorSetListFeatures	18
openSkiesAircraft	19
openSkiesAirport	21
openSkiesFlight	22
openSkiesRoute	23
openSkiesStateVector	24
openSkiesStateVectorSet	25
plotPlanes	26
plotRoute	27
plotRoutes	28

Index **31**

ADSBDecoder	<i>An object of class adsbDecoder object representing a decoder used to decode ADS-B v2 messages</i>
-------------	--

Description

[R6Class](#) object of class adsbDecoder representing a decoder used to decode ADS-B v2 messages. Provides methods for decoding a single message or a batch. Additionally, it includes methods for transforming hex strings into bits vectors and decoding some individual fields. Decoded messages are returned as lists with each decoded field.

Usage

ADSBDecoder

Fields

`lastOddPosition` Last ground or airborne position message decoded with `decodeGroundPositionMessage` or `decodeAirbornePositionMessage`

`lastEvenPosition` Last ground or airborne position message decoded with `decodeGroundPositionMessage` or `decodeAirbornePositionMessage`

Methods

- `hexToBits(hex)` Transform a hexadecimal string into its corresponding bits representation, with higher bits in the first positions. ,
- `decodeCPR(cprLatEven, cprLonEven, cprLatOdd, cprLonOdd, isAirborne=TRUE)` Decodes a pair of CPR-encoded positions given as longitudes and latitudes, corresponding to a pair of even and odd messages, obtaining the actual positions for both. The `isAirborne` argument indicates whether or not the CPR-encoded positions correspond to `isAirborne` position messages or not (ground position messages). The result is given as a vector with the decoded positions for both the even and odd messages, in the following order: even latitude, even longitude, odd latitude, odd longitude. ,
- `decodeMessage(message)` Decodes a single ADS-B v2 message in the form of a bits vector (higher bits in the first positions). The last even and odd positional messages are cached to decode following positional messages. The following message types are supported: aircraft identification, airborne position, ground position, airborne velocity, and operation status. ,
- `decodeMessages(messages)` Decodes several ADS-B v2 messages in the form of a list of bits vector (higher bits in the first positions). The following message types are supported: aircraft identification, airborne position, ground position, airborne velocity, and operation status.

Examples

```
# Decode three messages, using both individual decoding and batch decoding.
# The two first messages contain the airborne position.
# The third one, the aircraft identification

msg0 <- ADSBDecoder$hexToBits("8D40621D58C386435CC412692AD6")
msg1 <- ADSBDecoder$hexToBits("8D40621D58C382D690C8AC2863A7")
msg2 <- ADSBDecoder$hexToBits("8D4840D6202CC371C32CE0576098")

decoded0 <- ADSBDecoder$decodeMessage(msg0)
decoded1 <- ADSBDecoder$decodeMessage(msg1)
decoded2 <- ADSBDecoder$decodeMessage(msg2)

decodedAll <- ADSBDecoder$decodeMessages(list(msg0, msg1, msg2))
```

clusterRoutes

Cluster aircraft trajectories based on positional features

Description

Performs clustering of aircraft trajectories positional based on their positional features with several available methods. The input should be either a list of `openSkiesStateVectorSet` or an already computed features matrix as returned by `getVectorSetListFeatures`. If the input is a list of vector sets, features will be computed with default settings.

Usage

```
clusterRoutes(input, method="dbscan", eps=0.5, numberClusters=NULL, ...)
```

Arguments

input	input to be clustered, given as either a list of openSkiesStateVectorSet , or a matrix of positional features extracted from a list of openSkiesStateVectorSet objects with getVectorSetListFeatures , that will be used to identify clusters.
method	clustering method that will be applied to the positional features. Accepted methods are: dbscan, kmeans, hclust, fanny, clara, agnes
eps	Size of the epsilon neighborhood to be passed to dbscan . This argument is only used if the selected clustering method is dbscan..
numberClusters	number of expected clusters. If NULL or a value lesser than 2 is passed, the number of clusters will be estimated. This argument is only used if the selected clustering method is kmeans, hclust, fanny, clara, or agnes
...	additional arguments accepted by the selected clustering method.

Value

An object with clustering results, containing at least a "cluster". For additional details, see the documentation of [cluster](#).

Examples

```
if(interactive()){
# Retrieve series of state vectors for 7 instances of flights between
# Cagliari-Elmas airport and Parma airport

vectors1=getAircraftStateVectorsSeries(aircraft="4d2219",
startTime="2020-11-06 09:20:00", endTime="2020-11-06 10:30:00",
timeZone="Europe/London", timeResolution=300)

vectors2=getAircraftStateVectorsSeries(aircraft="4d226c",
startTime="2020-10-30 09:20:00", endTime="2020-10-30 10:30:00",
timeZone="Europe/London", timeResolution=300)

vectors3=getAircraftStateVectorsSeries(aircraft="4d225b",
startTime="2020-10-29 07:15:00", endTime="2020-10-29 08:25:00",
timeZone="Europe/London", timeResolution=300)

vectors4=getAircraftStateVectorsSeries(aircraft="4d225b",
startTime="2020-10-25 06:25:00", endTime="2020-10-25 07:35:00",
timeZone="Europe/London", timeResolution=300)

vectors5=getAircraftStateVectorsSeries(aircraft="4d224e",
startTime="2020-10-19 09:30:00", endTime="2020-10-19 10:40:00",
timeZone="Europe/London", timeResolution=300)

vectors6=getAircraftStateVectorsSeries(aircraft="4d225b",
startTime="2020-10-16 09:30:00", endTime="2020-10-16 10:30:00",
timeZone="Europe/London", timeResolution=300)

vectors7=getAircraftStateVectorsSeries(aircraft="4d227d",
startTime="2020-10-12 09:30:00", endTime="2020-10-12 10:30:00",
```

```

timeZone="Europe/London", timeResolution=300)

# Retrieve state vectors for an outlier flight, corresponding to a flight
# between the airports of Sevilla and Palma de Mallorca

vectors8=getAircraftStateVectorsSeries(aircraft = "4ca7b3",
startTime="2020-11-04 10:30:00", endTime="2020-11-04 12:00:00",
timeZone="Europe/London", timeResolution=300)

## Group all the openSkiesStateVectorSet objects in a single list

vectors_list=list(vectors1, vectors2, vectors3, vectors4, vectors5, vectors6, vectors7, vectors8)

## Extract the matrix of features

features_matrix=getVectorSetListFeatures(vectors_list, scale=TRUE, useAngles=FALSE)

## Perform clustering

clustering=clusterRoutes(features_matrix, "dbscan", eps=5)

## Display clustering results with flights colored by assigned cluster

plotRoutes(vectors_list, pathColors=clustering$cluster, literalColors=FALSE)
}

```

getAircraftFlights	<i>Retrieve flights performed by a specified aircraft during a time interval</i>
--------------------	--

Description

Retrieves the list of flights registered for a specified aircraft during a given time interval. The aircraft must be specified using its ICAO 24-bit address. Beginning and end times must be specified as date-time strings in any format that can be unambiguously converted to POSIXct (such as YYYY-MM-DD HH:MM:SS).

Usage

```

getAircraftFlights(aircraft, startTime, endTime, timeZone=Sys.timezone(),
                    username=NULL, password=NULL, includeStateVectors=FALSE,
                    timeResolution=NULL, useImpalaShell=FALSE,
                    includeAirportsMetadata=FALSE)

```

Arguments

aircraft	string with the ICAO 24-bit address of an aircraft (for example, <code>"346190"</code> for Air Nostrum EC-NCD (ATR 72-600)).
----------	--

<code>startTime</code>	date-time string indicating the starting time of the interval for which flights should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are <code>"2011-03-27 01:30:00"</code> and <code>"2011/03/27 01:30:00"</code> .
<code>endTime</code>	date-time string indicating the ending time of the interval for which flights should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are <code>"2011-03-28 01:30:00"</code> and <code>"2011/03/28 01:30:00"</code> .
<code>timeZone</code>	string with the name of the time zone for <code>startTime</code> and <code>endTime</code> . For details on supported time zones, see <code>help(timezones)</code> . By default, the system time zone is used.
<code>username</code>	optional string with the username to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>password</code>	optional string with the password to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>includeStateVectors</code>	logical indicating if the set of state vectors for each flight should also be retrieved. By default, state vectors are not retrieved.
<code>timeResolution</code>	time resolution in seconds with which state vectors should be retrieved if <code>includeStateVectors=TRUE</code> .
<code>useImpalaShell</code>	logical indicating whether or not to use the OpenSky impala shell instead of the API to retrieve state vectors if <code>includeStateVectors=TRUE</code> . If used, the provided username and password are used for the ssh connection. By default, the impala shell is not used.
<code>includeAirportsMetadata</code>	logical indicating if the data about the origin and destination airports of each flight should also be retrieved. If not, only the ICAO24 code of the airports will be included.

Value

A list of objects of class `openSkiesFlight`, where each object represents a flight that was performed by the specified aircraft during the specified time interval. See the `openSkiesFlight` documentation for details on the fields for the class.

References

<https://opensky-network.org/apidoc/rest.html>

Examples

```
# Obtain a list with information for all the flights registered for the aircraft
# with ICAO 24-bit address 346190 during the 26th of July, 2019.
```

```
if(interactive()){
  getAircraftFlights("346190", startTime="2019-07-26 00:00:00",
    endTime="2019-07-26 23:59:59", timeZone="Europe/Madrid")
}
```

getAircraftMetadata *Retrieve metadata for a specified aircraft*

Description

Retrieves the available metadata for a specified aircraft. These include the registration ID for the aircraft, as well as information about its manufacturer, owner, operator and country of registration, among others. A single aircraft must be specified using its ICAO 24-bit address.

Usage

```
getAircraftMetadata(aircraft)
```

Arguments

aircraft	string with the ICAO 24-bit address of an aircraft (for example, <code>"3c6444"</code> for Lufthansa D-AIBD (Airbus A319)).
----------	---

Value

An object of class [openSkiesAircraft](#). See the [openSkiesAirport](#) documentation for details on the fields for the class.

References

<https://www.icao.int/publications/doc8643/pages/search.aspx>

<https://www.eurocontrol.int/sites/default/files/content/documents/nm/asterix/archives/asterix-cat021-asterix-ads-b-messages-part-12-v1.4-072009.pdf>

Examples

```
# Obtain metadata for the aircraft with ICAO 24-bit address 3922e2

if(interactive()){
  getAircraftMetadata("3922e2")
}
```

```
getAircraftStateVectorsSeries
```

Retrieve a series of state vectors received from a specified aircraft during a given time interval

Description

Retrieves a time series of state vectors received from a specified aircraft during a given time interval. A state vector is a collection of data elements that characterize the status of an aircraft at a given point during a flight (such as latitude, longitude, altitude, etc.)

The time point must be specified as a date-time string in any format that can be unambiguously converted to POSIXct (such as YYYY-MM-DD HH:MM:SS). Time resolution for the time series of state vectors must be specified in seconds. Time resolution is limited to 10 s for anonymous users, and 5 s for registered users.

Usage

```
getAircraftStateVectorsSeries(aircraft, startTime, endTime,
                              timeZone=Sys.timezone(), timeResolution,
                              username=NULL, password=NULL,
                              useImpalaShell=FALSE)
```

Arguments

<code>aircraft</code>	string with the ICAO 24-bit address of an aircraft (for example, <code>"346190"</code> for Air Nostrum EC-NCD (ATR 72-600), or a character vector with multiple ICAO 24-bit addresses. In the default behavior, data is retrieved for any aircraft.
<code>startTime</code>	date-time string indicating the starting time of the interval for which state vectors should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are <code>"2011-03-27 01:30:00"</code> and <code>"2011/03/27 01:30:00"</code> .
<code>endTime</code>	date-time string indicating the ending time of the interval for which state vectors should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are <code>"2011-03-28 01:30:00"</code> and <code>"2011/03/28 01:30:00"</code> .
<code>timeZone</code>	string with the name of the time zone for time. For details on supported time zones, see <code>help(timezones)</code> . By default, the system time zone is used.
<code>timeResolution</code>	time resolution in seconds to be used for the requested timeseries of state vectors. Limited to 5 s for anonymous users and 10 s for registered users.
<code>username</code>	optional string with the username to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>password</code>	optional string with the password to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>useImpalaShell</code>	logical indicating whether or not to use the OpenSky impala shell instead of the API to retrieve state vectors. If used, the provided username and password are used for the ssh connection. By default, the impala shell is not used.

Value

An [openSkiesStateVectorSet](#) object with field `time_series=TRUE`. For details on the information stored in state vectors, see the documentation for [openSkiesStateVector](#) and [openSkiesStateVectorSet](#).

References

<https://opensky-network.org/apidoc/rest.html>

Examples

```
# Obtain a time series of state vectors for the aircraft with ICAO 24-bit
# address 403003 for the 8th of October, 2020 between 16:50 and 16:53 (London
# time), with a time resolution of 1 minute.

if(interactive()){
  getAircraftStateVectorsSeries("403003", startTime = "2020-10-08 16:50:00",
  endTime = "2020-10-08 16:52:00", timeZone="Europe/London", timeResolution=60)
}
```

getAirportArrivals	<i>Retrieve flight arrivals into a specified airport</i>
--------------------	--

Description

Retrieves the list of flights that landed into a specified airport during a certain time interval. The airport must be specified using its ICAO identifier. Beginning and end times must be specified as date-time strings in any format that can be unambiguously converted to POSIXct (such as YYYY-MM-DD HH:MM:SS).

Usage

```
getAirportArrivals(airport, startTime, endTime, timeZone=Sys.timezone(),
  username=NULL, password=NULL, includeStateVectors=FALSE,
  timeResolution=NULL, useImpalaShell=FALSE,
  includeAirportsMetadata=FALSE)
```

Arguments

airport	string with the ICAO identifier of an airport (for example, "EDDF" for Frankfurt International Airport).
startTime	date-time string indicating the starting time of the interval for which arrivals should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are "2011-03-27 01:30:00" and "2011/03/27 01:30:00".
endTime	date-time string indicating the ending time of the interval for which arrivals should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are "2011-03-28 01:30:00" and "2011/03/28 01:30:00".

<code>timeZone</code>	string with the name of the time zone for <code>startTime</code> and <code>endTime</code> . For details on supported time zones, see <code>help(timezones)</code> . By default, the system time zone is used.
<code>username</code>	optional string with the username to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>password</code>	optional string with the password to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>includeStateVectors</code>	logical indicating if the set of state vectors for each flight should also be retrieved. By default, state vectors are not retrieved.
<code>timeResolution</code>	time resolution in seconds with which state vectors should be retrieved if <code>includeStateVectors=TRUE</code> .
<code>useImpalaShell</code>	logical indicating whether or not to use the OpenSky impala shell instead of the API to retrieve state vectors if <code>includeStateVectors=TRUE</code> . If used, the provided username and password are used for the ssh connection. By default, the impala shell is not used.
<code>includeAirportsMetadata</code>	logical indicating if the data about the origin and destination airports of each flight should also be retrieved. If not, only the ICAO24 code of the airports will be included.

Value

A list of objects of class `openSkiesFlight`, where each object represents a flight that landed at the specified airport during the specified time interval. See the `openSkiesFlight` documentation for details on the fields for the class.

References

<https://opensky-network.org/apidoc/rest.html>

Examples

```
# Obtain a list with information for all the flights that landed at Frankfurt
# International Airport on the 29th of January, 2018 between 12 PM and 1 PM,
# local time.

if(interactive()){
  getAirportArrivals(airport="EDDF", startTime="2018-01-29 12:00:00",
    endTime="2018-01-29 13:00:00", timeZone="Europe/Berlin")
}
```

getAirportDepartures *Retrieve flight departures from a specified airport*

Description

Retrieves the list of flights that departed from a specified airport during a certain time interval. The airport must be specified using its ICAO identifier. Beginning and end times must be specified as date-time strings in any format that can be unambiguously converted to POSIXct (such as YYYY-MM-DD HH:MM:SS).

Usage

```
getAirportDepartures(airport, startTime, endTime, timeZone=Sys.timezone(),
                    username=NULL, password=NULL, includeStateVectors=FALSE,
                    timeResolution=NULL, useImpalaShell=FALSE,
                    includeAirportsMetadata=FALSE)
```

Arguments

airport	string with the ICAO identifier of an airport (for example, "LEZL" for Seville Airport).
startTime	date-time string indicating the starting time of the interval for which departures should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are "2011-03-27 01:30:00" and "2011/03/27 01:30:00".
endTime	date-time string indicating the ending time of the interval for which departures should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are "2011-03-28 01:30:00" and "2011/03/28 01:30:00".
timeZone	string with the name of the time zone for startTime and endTime. For details on supported time zones, see help(timezones). By default, the system time zone is used.
username	optional string with the username to use for authentication for the OpenSky API. By default, no authentication is performed.
password	optional string with the password to use for authentication for the OpenSky API. By default, no authentication is performed.
includeStateVectors	logical indicating if the set of state vectors for each flight should also be retrieved. By default, state vectors are not retrieved.
timeResolution	time resolution in seconds with which state vectors should be retrieved if includeStateVectors=TRUE.
useImpalaShell	logical indicating whether or not to use the OpenSky impala shell instead of the API to retrieve state vectors if includeStateVectors=TRUE. If used, the provided username and password are used for the ssh connection. By default, the impala shell is not used.

includeAirportsMetadata

logical indicating if the data about the origin and destination airports of each flight should also be retrieved. If not, only the ICAO24 code of the airports will be included.

Value

A list of objects of class [openSkiesFlight](#), where each object represents a flight that that departed from the specified airport during the specified time interval. See the [openSkiesFlight](#) documentation for details on the fields for the class.

References

<https://opensky-network.org/apidoc/rest.html>

Examples

```
# Obtain a list with information for all the flights that departed from Seville
# Airport on the 25th of July, 2019 between 9 AM and 11 AM, local time.

if(interactive()){
  getAirportDepartures(airport="LEZL", startTime="2019-07-25 09:00:00",
    endTime="2019-07-25 11:00:00", timeZone="Europe/Madrid")
}
```

getAirportMetadata	<i>Retrieve metadata for a specified airport</i>
--------------------	--

Description

Retrieves the available metadata for a specified airport. These include its IATA code, common name and location, among others. A single airport must be specified using its ICAO code.

Usage

```
getAirportMetadata(airport)
```

Arguments

airport	string with the ICAO 4-letter code of an airport (for example, "LEZL" for Sevilla Airport).
---------	---

Value

An object of class [openSkiesAirport](#). See the [openSkiesAirport](#) documentation for details on the fields for the class.

References

https://en.wikipedia.org/wiki/List_of_airports_by_IATA_and_ICAO_code

Examples

```
# Obtain metadata for the airport with ICAO code LEZL

if(interactive()){
  getAirportMetadata("LEZL")
}
```

getIntervalFlights *Retrieve all flights registered during a time interval*

Description

Retrieves the list of all flights registered for any aircraft during a given time interval. Beginning and end times must be specified as date-time strings in any format that can be unambiguously converted to POSIXct (such as YYYY-MM-DD HH:MM:SS).

Usage

```
getIntervalFlights(startTime, endTime, timeZone=Sys.timezone(), username=NULL,
                    password=NULL, includeStateVectors=FALSE,
                    timeResolution=NULL, useImpalaShell=FALSE,
                    includeAirportsMetadata=FALSE)
```

Arguments

startTime	date-time string indicating the starting time of the interval for which flights should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are <code>"2011-03-27 01:30:00"</code> and <code>"2011/03/27 01:30:00"</code> .
endTime	date-time string indicating the ending time of the interval for which flights should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are <code>"2011-03-28 01:30:00"</code> and <code>"2011/03/28 01:30:00"</code> .
timeZone	string with the name of the time zone for startTime and endTime. For details on supported time zones, see <code>help(timezones)</code> . By default, the system time zone is used.
username	optional string with the username to use for authentication for the OpenSky API. By default, no authentication is performed.
password	optional string with the password to use for authentication for the OpenSky API. By default, no authentication is performed.
includeStateVectors	logical indicating if the set of state vectors for each flight should also be retrieved. By default, state vectors are not retrieved.
timeResolution	time resolution in seconds with which state vectors should be retrieved if includeStateVectors=TRUE.

`useImpalaShell` logical indicating whether or not to use the OpenSky impala shell instead of the API to retrieve state vectors if `includeStateVectors=TRUE`. If used, the provided username and password are used for the ssh connection. By default, the impala shell is not used.

`includeAirportsMetadata` logical indicating if the data about the origin and destination airports of each flight should also be retrieved. If not, only the ICAO24 code of the airports will be included.

Value

A list of objects of class `openSkiesFlight`, where each object represents a flight that was registered during the specified time interval. See the `openSkiesFlight` documentation for details on the fields for the class.

References

<https://opensky-network.org/apidoc/rest.html>

Examples

```
# Obtain a list with information for all the flights registered during the 16th
# of November, 2019 between 9 AM and 10 AM, London time.
```

```
if(interactive()){
  flights <- getIntervalFlights(startTime="2019-11-16 09:00:00",
                                endTime="2019-11-16 10:00:00", timeZone="Europe/London")
```

```
# Count the number of registered flights.
```

```
length(flights)
}
```

<code>getRouteMetadata</code>	<i>Retrieve metadata for a specified route</i>
-------------------------------	--

Description

Retrieves the available metadata for a specified flight route. These include the airports of origin and destination, the operator IATA code and the flight number.

Usage

```
getRouteMetadata(route, includeAirportsMetadata=FALSE)
```

Arguments

route	string with the call sign of a route (for example, "AAL683" for American Airlines flight number 683).
includeAirportsMetadata	logical indicating if the data about the origin and destination airports of the route should also be retrieved. If not, only the ICAO24 code of the airports will be included.

Value

An object of class `openSkiesRoute`. See the `openSkiesRoute` documentation for details on the fields for the class.

References

https://en.wikipedia.org/wiki/Flight_number

Examples

```
# Obtain metadata for the route with call sign AAL683

if(interactive()){
  getRouteMetadata("AAL683")
}
```

`getSingleTimeStateVectors`

Retrieve all state vectors received at a given time point

Description

Retrieves the list of all state vectors received from any or specified aircrafts at a single time point. A state vector is a collection of data elements that characterize the status of an aircraft at a given point during a flight (such as latitude, longitude, altitude, etc.)

The time point must be specified as a date-time string in any format that can be unambiguously converted to POSIXct (such as YYYY-MM-DD HH:MM:SS). Results can be filtered to specific ranges of latitudes and/or longitudes. The extent of the data than can be accessed varies depending on if login details are provided: * For anonymous users: + If no aircraft is specified or multiple aircrafts are specified: historical data cannot be retrieved. If a time point was specified, it will be ignored, and data for current time will be returned. + If a single aircraft is specified, historical data can be retrieved. * For registered users: + If no aircraft is specified or multiple aircrafts are specified: historical data of up to 1 hour ago from current time can be retrieved. + If a single aircraft is specified, historical data can be retrieved.

Usage

```
getSingleTimeStateVectors(aircraft=NULL, time=NULL, timeZone=Sys.timezone(),
                           minLatitude=NULL, maxLatitude=NULL, minLongitude=NULL,
                           maxLongitude=NULL, username=NULL, password=NULL,
                           useImpalaShell=FALSE)
```

Arguments

<code>aircraft</code>	string with the ICAO 24-bit address of an aircraft (for example, <code>"346190"</code> for Air Nostrum EC-NCD (ATR 72-600), or a character vector with multiple ICAO 24-bit addresses. In the default behavior, data is retrieved for any aircraft.
<code>time</code>	date-time string indicating the time point for which state vectors should be retrieved. Must be in a format that can be unambiguously converted into POSIXct time. Valid examples are <code>"2011-03-28 01:30:00"</code> and <code>"2011/03/28 01:30:00"</code> . If no time point is specified, data is retrieved for current time.
<code>timeZone</code>	string with the name of the time zone for time. For details on supported time zones, see <code>help(timezones)</code> . By default, the system time zone is used.
<code>minLatitude</code>	minimum latitude to filter the retrieved state vectors. Must be a value between -180 and 180. Negative values denote south latitudes, and positive values denote north latitudes. By default, no filtering based on location is performed.
<code>maxLatitude</code>	maximum latitude to filter the retrieved state vectors. Must be a value between -180 and 180. Negative values denote south latitudes, and positive values denote north latitudes. By default, no filtering based on location is performed.
<code>minLongitude</code>	minimum longitude to filter the retrieved state vectors. Must be a value between -180 and 180. Negative values denote west longitudes, and positive values denote east longitudes. By default, no filtering based on location is performed.
<code>maxLongitude</code>	maximum longitude to filter the retrieved state vectors. Must be a value between -180 and 180. Negative values denote west longitudes, and positive values denote east longitudes. By default, no filtering based on location is performed.
<code>username</code>	optional string with the username to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>password</code>	optional string with the password to use for authentication for the OpenSky API. By default, no authentication is performed.
<code>useImpalaShell</code>	logical indicating whether or not to use the OpenSky impala shell instead of the API to retrieve state vectors. If used, the provided username and password are used for the ssh connection. By default, the impala shell is not used.

Value

If a single state vector matching the query conditions is found, an `openSkiesStateVector` object. If multiple state vectors matching the query conditions are found, an `openSkiesStateVectorSet` object with field `time_series=FALSE`. For details on the information stored in state vectors, see the documentation for `openSkiesStateVector` and `openSkiesStateVectorSet`.

References

<https://opensky-network.org/apidoc/rest.html>

Examples

```
# Obtain a list with the state vectors for all aircrafts currently flying over
# an area covering Switzerland.

getSingleTimeStateVectors(minLatitude=45.8389, maxLatitude=47.8229,
minLongitude=5.9962, maxLongitude=10.5226)

# Obtain the state vector for aircraft with ICAO 24-bit address 403003 for
# the 8th of October, 2020 at 16:50 London time.

if(interactive()){
getSingleTimeStateVectors(aircraft="403003", time="2020-10-08 16:50:00",
timeZone="Europe/London")
}
```

getVectorSetFeatures *Get positional features of an openSkiesStateVectorSet object*

Description

Retrieves positional features of an [openSkiesStateVectorSet](#) object. Features will be uniformly interpolated from the observed values.

Usage

```
getVectorSetFeatures(stateVectorSet, resamplingSize=15, method="fmm", useAngles=FALSE)
```

Arguments

stateVectorSet	object of class openSkiesStateVectorSet for which positional features should be extracted.
resamplingSize	number of uniformly separated interpolation points at which the values of the position features should be calculated.
method	method to be used for interpolation. "linear" will result in linear interpolation, while "fmm", "periodic", or "natural" will result in different types of spline interpolation.
useAngles	logical indicating if the angle of the trajectory of the aircraft should be included in the extracted positional features.

Value

A vector with positional features of the provided [openSkiesStateVectorSet](#) object. The vector alternates values of longitude and latitude at each interpolated point, unless useAngles=TRUE, in which case values of longitude, latitude and trajectory angle are alternated, in this order.

Examples

```
# Extract positional features for a time series of state vectors for the
# aircraft with ICAO 24-bit address 403003 for the 8th of October, 2020 between
# 16:50 and 16:53 (London time), with a time resolution of 1 minute.

if(interactive()){
  vectors <- getAircraftStateVectorsSeries(aircraft="4ca7b3",
  startTime="2020-11-04 10:30:00", endTime="2020-11-04 12:00:00",
  timeZone="Europe/London", timeResolution=300)

  features <- getVectorSetFeatures(vectors)
}
```

```
getVectorSetListFeatures
```

Get positional features of a list of openSkiesStateVectorSet objects

Description

Retrieves positional features of a list of [openSkiesStateVectorSet](#) objects. Features will be uniformly interpolated from the observed values for all the [openSkiesStateVectorSet](#) objects.

Usage

```
getVectorSetListFeatures(stateVectorSetList, resamplingSize=15, method="fmm",
  scale=TRUE, useAngles=FALSE)
```

Arguments

stateVectorSetList	list of objects of class openSkiesStateVectorSet for which positional features should be extracted.
resamplingSize	number of uniformly separated interpolation points at which the values of the position features should be calculated.
method	method to be used for interpolation. "linear" will result in linear interpolation, while "fmm", "periodic", or "natural" will result in different types of spline interpolation.
scale	logical indicating if the matrix of features should be scaled by applying the scale function. This can be desirable if the features are going to be used for clustering.
useAngles	logical indicating if the angle of the trajectory of the aircraft should be included in the extracted positional features.

Value

A matrix with positional features of the provided list of [openSkiesStateVectorSet](#) objects. Each row of the matrix represents a vector of features for each of the [openSkiesStateVectorSet](#) objects. Each vector alternates values of longitude and latitude at each interpolated point, unless `useAngles=TRUE`, in which case values of longitude, latitude and trajectory angle are alternated, in this order.

Examples

```
# Extract positional features for a time series of state vectors for the
# aircraft with ICAO 24-bit address 403003 for the 8th of October, 2020 between
# 16:50 and 16:53 (London time), with a time resolution of 1 minute.

if(interactive()){
  vectors1=getAircraftStateVectorsSeries(aircraft="345107",
    startTime="2020-11-04 11:55:00", endTime="2020-11-04 13:10:00",
    timeZone="Europe/London", timeResolution=300)

  vectors2=getAircraftStateVectorsSeries(aircraft = "4ca7b3",
    startTime="2020-11-04 10:30:00", endTime="2020-11-04 12:00:00",
    timeZone="Europe/London", timeResolution=300)

  vectors_list=list(vectors1, vectors2)

  features_matrix=getVectorSetListFeatures(vectors_list, scale=FALSE, useAngles=TRUE)
}
```

openSkiesAircraft *An [R6Class](#) object representing an aircraft*

Description

[R6Class](#) object representing an aircraft. Contains information about the ICAO 24-bit code of the aircraft, its registration code, its country of origin, its manufacturer and its operator. New instances can be manually created by providing values for at least the ICAO24 field. Alternatively, [getAircraftMetadata](#) will return an [openSkiesAirport](#) object corresponding to the airport with the provided ICAO 24-bit code.

Usage

```
openSkiesAircraft
```

Fields

ICA024 String with the ICAO 24-bit aircraft address associated to the aircraft in hexadecimal format
 registration String with the aircraft registration code, also called tail number
 origin_country String with the country where the aircraft is registered

`last_state_vector` An object of class `openSkiesStateVector` representing the last known state vector for the aircraft

`state_vector_history` An object of class `openSkiesStateVectorSet` representing the history of all known state vectors for the aircraft

`manufacturer_name` String with the name of the manufacturer of the aircraft

`manufacturer_ICAO` String with the ICAO code of the manufacturer of the aircraft

`model` String with the aircraft model

`serial_number` String with the serial number of the aircraft

`line_number` String with the line number of the aircraft. Usually only provided for Boeing aircrafts. Line numbers specify the order in which airframes of a particular product line were assembled.

`ICAO_type_code` String with the ICAO code for the model of aircraft

`ICAO_aircraft_class` String with the ICAO code for the type of aircraft. ICAO aircraft classes provide more general groups than ICAO type codes

`owner` String with the name of the registered aircraft owner

`operator` String with the name of the aircraft operator

`operator_call_sign` String with the callsign of the aircraft operator

`operator_ICAO` String with the ICAO code of the aircraft operator

`operator_IATA` String with the IATA code of the aircraft operator

`first_flight_date` String with the date when the first flight for the aircraft was registered. This information is usually not available when retrieving information from the OpenSky API

`category_description` String with physical information about the aircraft provided by the ADS-B emitter unit

Methods

`get_values(field, removeNAs=FALSE)` This method retrieves the value of `field` for all the state vectors stored in the `openSkiesStateVectorSet` object. If `removeNAs=TRUE` (by default, `removeNAs=FALSE`), missing values are removed from the output. Otherwise, NA is returned in place of missing items.

Examples

```
# Create an openSkiesAircraft object corresponding to the aircraft with
# ICAO 24-bit address 3922e2

if(interactive()){
  test_aircraft <- getAircraftMetadata("3922e2")
  test_aircraft
}
```

openSkiesAirport *An R6Class object representing an airport*

Description

[R6Class](#) object representing an airport. Contains information about the name of the airport, its IATA and ICAO codes, and its location. New instances can be manually created by providing values for at least the fields `name`, `city`, `country`, `longitude` and `latitude`. Alternatively, [getAirportMetadata](#) will return an `openSkiesAirport` object corresponding to the airport with the provided ICAO code.

Usage

```
openSkiesAirport
```

Fields

`name` String with the name of the airport
`ICAO` String with the ICAO code of the airport
`IATA` String with the IATA code of the airport
`longitude` Longitude of the position of the airport
`latitude` Latitude of the position of the airport
`altitude` Altitude of the position of the airport
`city` String with the name of the city where the airport is located
`municipality` String with the ISO 3166-2 code where the airport is located
`region` String with the name of the region where the airport is located
`country` String with the ISO 3166-1 alpha-2 code of the country where the airport is located
`continent` String with the ISO 3166-1 alpha-2 code of the continent where the airport is located
`type` String with information about the type of airport
`website` String with the URL for the website of the airport
`wikipedia_entry` String with the URL for the Wikipedia entry of the airport
`reliable_position` Logical value indicating if the position of the airport is reliable
`GPS_code` String with the GPS code of the airport

Examples

```
# Create an openSkiesAirport object corresponding to Sevilla Airport

if(interactive()){
  test_airport <- getAirportMetadata("LEZL")
  test_airport
}
```

openSkiesFlight *An R6Class object representing a specific flight*

Description

[R6Class](#) object representing a specific flight performed by a certain aircraft. Contains information about the aircraft that performed the flight, the airports of origin and destination, the times of departure and arrival and the callsign under which the flight was performed. New instances can be manually created by providing values for at least the fields `ICAO24`, `departure_time` and `arrival_time`. Alternatively, [getAircraftFlights](#), [getAirportDepartures](#), [getAirportArrivals](#) [getIntervalFlights](#) will all return lists of `openSkiesFlight` objects corresponding to the flights that match the query conditions.

Usage

```
openSkiesFlight
```

Fields

`ICAO24` String with the ICAO 24-bit aircraft address associated to the aircraft in hexadecimal format
`call_sign` String with callsign under which the flight was performed
`state_vectors` Object of class [openSkiesStateVectorSet](#) with field `time_series = TRUE` containing the state vectors received from the aircraft during the flight
`origin_airport` String with the ICAO 4-letter code of the airport of origin
`destination_airport` String with the ICAO 4-letter code of the destination airport
`departure_time` String with the date and time at which the aircraft took off
`arrival_time` String with the date and time at which the aircraft arrived at its destination

Methods

`get_moment_state_vector(time, includeFuture = TRUE)` This method retrieves the state vector closest with the timestamp closest to the provided time, which must be supplied as a date-time string. In the default behaviour, `includeFuture=TRUE` and the retrieved vector will be the one with the closest timestamp, regardless of if this is earlier or later than the provided time. If `includeFuture=FALSE`, the closest earlier state vector will be retrieved. ,

`get_duration()` This method returns the duration of the flight in seconds ,

`distance_to_flight(flight, numberSamples=15, samplesAggregationMethod="concatenated", method="euclidean")`
 This method calculates the distance to the provided flight, which must be another object of class `openSkiesFlight`. Both `openSkiesFlight` objects will be resampled to the number of points specified by `numberSamples`. If `samplesAggregationMethod="concatenated"`, a vector of values indicating the distance between the flights at each point is returned. If `samplesAggregationMethod="average"`, the average distance is returned. By default, `method="euclidean"` and euclidean distances are calculated. Other possible values of `method` are all values accepted by [dist](#). By default, `useAngles=FALSE`, and track angles are not included in the features vectors used to calculate distances.

Examples

```
# Create a list of openSkiesFlight objects corresponding to all the flights that
# landed at Frankfurt International Airport on the 29th of January, 2018 between
# 12 PM and 1 PM

if(interactive()){
  test_flights <- getAirportArrivals(airport="EDDF", startTime="2018-01-29 12:00:00",
  endTime="2018-01-29 13:00:00", timeZone="Europe/Berlin")
  test_flights
}
```

openSkiesRoute	An R6Class object representing a flight route
----------------	---

Description

[R6Class](#) object representing a flight route, usually operated by a commercial airline. Contains information about the callsign under which the route is operated, the operator itself and the airports of origin and destination. New instances can be manually created by providing values for at least the fields `call_sign`, `origin_airport` and `destination_airport`. Alternatively, [getRouteMetadata](#) will return an `openSkiesRoute` object corresponding to the route with the provided callsign.

Usage

```
openSkiesRoute
```

Fields

`call_sign` String with callsign of the route
`origin_airport` String with the ICAO 4-letter code of the airport of origin
`destination_airport` String with the ICAO 4-letter code of the destination airport
`operator_IATA` String with the IATA code for the operator of the route
`flight_number` String with the flight number for the route. The callsign is usually composed of an airline identifier and the flight number

Examples

```
# Create an openSkiesRoute object corresponding to the American Airlines route
# with callsign AAL683

if(interactive()){
  test_route <- getRouteMetadata("AAL683")
  test_route
}
```

openSkiesStateVector An [R6Class](#) object representing an aircraft state vector

Description

[R6Class](#) object representing an aircraft state vector. Contains information about status at a given timepoint of an aircraft, including its position, altitude and velocity. New instances can be manually created by providing values for at least the `ICA024`, `longitude` and `latitude` fields. Alternatively, [getSingleTimeStateVectors](#) will return an `openSkiesStateVector` object if a single state vector matching the query conditions is found.

Usage

```
openSkiesStateVector
```

Fields

`ICA024` String with the ICAO 24-bit aircraft address associated to the aircraft in hexadecimal format

`call_sign` String with the callsign assigned to the aircraft

`origin_country` String with the country where the aircraft is registered

`requested_time` String with the time point for which the state vector was requested

`last_position_update_time` String with the time at which the last position update for the aircraft was received, or NULL if no position update had been received in the past 15 s

`last_any_update_time` String with the time at which the last update (of any type) for the aircraft was received

`longitude` Longitude value for the position of the aircraft

`latitude` Latitude value for the position of the aircraft

`baro_altitude` Barometric altitude of the aircraft in meters

`geo_altitude` Geometric altitude of the aircraft in meters

`on_ground` Logical indicating if the aircraft is at a surface position

`velocity` velocity of the aircraft over the ground in meters/second

`true_track` True track angle in degrees of the current aircraft course. Measured clockwise from the North (0°)

`vertical_rate` Vertical movement rate of the aircraft in meters/second. Positive means the aircraft is climbing, and negative means it is descending

`squawk` String with the squawk code for the aircraft transponder

`special_purpose_indicator` Logical indicating if the transponder of the aircraft has emitted a Special Purpose Indicator pulse

`position_source` String with the source of the position information for this state vector. Can be ADS-B (Automatic Dependent Surveillance–Broadcast), ASTERIX (All Purpose Structured Eurocontrol Surveillance Information Exchange) or MLA (Multilateration)

Examples

```
# Obtain the state vector for aircraft with ICAO 24-bit address 403003 for
# the 8th of October, 2020 at 16:50 London time.

if(interactive()){
  test_stateVector <- getSingleTimeStateVectors(aircraft="403003",
  time="2020-10-08 16:50:00", timeZone="Europe/London")
  test_stateVector
}
```

openSkiesStateVectorSet

An R6Class object representing an ensemble of aircraft state vectors

Description

[R6Class](#) object representing an ensemble of aircraft state vectors. Contains a list of objects of class [openSkiesStateVector](#). The ensemble can either represent a time series of state vectors of a single aircraft, or represent state vectors corresponding to multiple different aircrafts. For details on the information stored in each state vector, see the documentation for the [openSkiesStateVector](#) class. New instances can be manually created by providing a list of [openSkiesStateVector](#) objects. Alternatively, [getSingleTimeStateVectors](#) and [getAircraftStateVectorsSeries](#) will return an [openSkiesStateVectorSet](#) object if multiple state vectors matching the query conditions are found.

Usage

```
openSkiesStateVectorSet
```

Fields

`state_vectors` List of [openSkiesStateVector](#) objects

`time_series` Logical indicating if the [openSkiesStateVectorSet](#) object represents a time series of state vectors of a single aircraft.

Methods

`add_state_vector(state_vector)` This method adds a new state vector to the `openSkiesAircraft` object, which will be set as the new `last_state_vector` and will be added to `state_vector_history`. `state_vector` should be an [openSkiesStateVector](#) object

`get_uniform_interpolation(n, fields, method="fmm")` This method obtains a data frame with an interpolation of the specified fields along the route represented by the state vector set across `n` evenly distributed points. `fields` should be a character vector with the name of the fields that will be interpolated. Only numeric fields are accepted. `method` represents the interpolation method. "linear" will result in linear interpolation, while "fmm", "periodic", or "natural" will result in different types of spline interpolation.

`get_time_points_interpolation(fields, time_field, timestamps, method="fmm")` This method obtains a data frame with an interpolation of the specified fields along the route represented by the state vector set across the specified timestamps. `fields` should be a character vector with the name of the fields that will be interpolated. Only numeric fields are accepted. `method` represents the interpolation method. "linear" will result in linear interpolation, while "fmm", "periodic", or "natural" will result in different types of spline interpolation. `time_field` indicates the name of the field of the `openSkiesStateVectorSet` object from which the timestamps of the original state vectors will be retrieved. Possible values are `c("requested_time", "last_position_update_time", "1")`. The time points at which the interpolations should be calculated should be provided as a vector through the `timestamps` argument.

`sort_by_field(field, decreasing=FALSE)` This method sorts the state vectors of the `openSkiesStateVectorSet` object by the values of the field provided through `field`. By default, `decreasing=FALSE` and the state vectors will be sorted by increasing order of field. If `decreasing=TRUE`, decreasing order will be used.

`split_into_flights(timeOnLandThreshold=300, timeDiffThreshold=1800)` This method automatically detects different flights contained in the `openSkiesStateVectorSet` object and returns a list of objects of class `openSkiesFlight`. Separate flights are detected based on two conditions: either the aircraft staying on ground for a given amount of time, or the aircraft not sending any status update for a given period. The thresholds are controlled, respectively, through the `timeOnLandThreshold` and `timeDiffThreshold` arguments. In both cases, the value should be provided in seconds.

Examples

```
# Obtain a time series of state vectors for the aircraft with ICAO 24-bit
# address 403003 for the 8th of October, 2020 between 16:50 and 16:53 (London
# time), with a time resolution of 1 minute.

if(interactive()){
  test_stateVectorSet <- getAircraftStateVectorsSeries("403003", startTime="2020-10-08 16:50:00",
  endTime="2020-10-08 16:52:00", timeZone="Europe/London", timeResolution=60)
  test_stateVectorSet
}
```

plotPlanes

Plot the location of a set of aircrafts

Description

Draws the location of a set of aircrafts given in their state vectors on a `ggmap` object. The planes will be oriented according to the path they are following.

Usage

```
plotPlanes(stateVectors, ggmapObject=NULL, plotResult=TRUE, paddingFactor=0.2,
           iconSize=1)
```

Arguments

stateVectors	list of state vectors. Each state vector must be represented by a list with, at least, fields "longitude", "latitude" and "trueTrack".
ggmapObject	optional ggmap object on which the route will be drawn. By default, a new ggmap object will be created, covering the necessary space plus an amount of padding determined by the paddingFactor argument.
plotResult	whether or not the resulting ggmap object with the added route should be plotted.
paddingFactor	amount of padding to add to the map if no ggmap object is provided. The added padding will be equal to paddingFactor multiplied by the height and width of the map (determined by the difference between the maximum and minimum longitudes/latitudes).
iconSize	scaling factor for the size of the plane icons.

Value

A ggmap object with added paths and points representing the route.

Examples

```
# Plot the position of aircrafts currently flying over an area covering
# Switzerland.

if(interactive()){
  vectors <- getSingleTimeStateVectors(minLatitude=45.8389, maxLatitude=47.8229,
                                       minLongitude=5.9962, maxLongitude=10.5226)

  plotPlanes(vectors)
}
```

plotRoute

Plot a single aircraft route

Description

Draws a given route on a ggmap object. The route must be given as an object of class `openSkiesStateVectorSet` with field `time_series=TRUE`.

Usage

```
plotRoute(stateVectorSet, pathColor="blue", ggmapObject=NULL, plotResult=TRUE,
          paddingFactor=0.2, lineSize=1, lineAlpha=0.5, pointSize=0.3,
          pointAlpha=0.8, arrowLength=0.3)
```

Arguments

stateVectorSet	object of class openSkiesStateVectorSet with field <code>time_series=TRUE</code> with positional information of an aircraft along different timepoints.
pathColor	color of the path and points that will be used to draw the route. Must be a value accepted by ggmap's color attributes.
ggmapObject	optional ggmap object on which the route will be drawn. By default, a new ggmap object will be created, covering the necessary space plus an amount of padding determined by the <code>paddingFactor</code> argument.
plotResult	whether or not the resulting ggmap object with the added route should be plotted.
paddingFactor	amount of padding to add to the map if no ggmap object is provided. The added padding will be equal to <code>paddingFactor</code> multiplied by the height and width of the map (determined by the difference between the maximum and minimum longitudes/latitudes).
lineSize	width of the line that connects the points of the route in the plot.
lineAlpha	opacity of the line that connects the points of the route in the plot.
pointSize	size of the points of the route in the plot.
pointAlpha	opacity of the points of the route in the plot.
arrowLength	length of the segment arrows in centimeters.

Value

A ggmap object with added paths and points representing the route.

Examples

```
# Plot the route followed by the aircraft with ICAO address 4ca7b3
# during the 4th of November, 2020.

if(interactive()){
  vectors <- getAircraftStateVectorsSeries(aircraft="4ca7b3",
    startTime="2020-11-04 10:30:00", endTime="2020-11-04 12:00:00",
    timeZone="Europe/London", timeResolution=300)

  plotRoute(vectors)
}
```

plotRoutes

Plot several aircraft routes

Description

Draws the given routes on a ggmap object. The routes must be given as a list of objects of class [openSkiesStateVectorSet](#), all of them with field `time_series=TRUE`.

Usage

```
plotRoutes(stateVectorSetList, pathColors="blue", gmapObject=NULL,
plotResult=TRUE, paddingFactor=0.2, lineSize=1, lineAlpha=0.5, pointSize=0.3,
pointAlpha=0.8, includeArrows=FALSE, arrowLength=0.3, literalColors=TRUE)
```

Arguments

stateVectorSetList	list of objects of class <code>openSkiesStateVectorSet</code> with field <code>time_series=TRUE</code> , each of them containing positional information of given aircraft along different timepoints.
pathColors	If <code>literalColors=TRUE</code> , vector with the colors of the paths and points that will be used to draw the routes. If the number of routes is greater than the number of colors, these will be rotated. Each color must be a value accepted by <code>gmap</code> 's color attributes. Alternatively, if <code>literalColors=FALSE</code> , a factor defining a certain feature for each route, in which case a color will be assigned for each level of the factor.
gmapObject	optional <code>gmap</code> object on which the routes will be drawn. By default, a new <code>gmap</code> object will be created, covering the necessary space plus an amount of padding determined by the <code>paddingFactor</code> argument.
plotResult	wether or not the resulting <code>gmap</code> object with the added routes should be plotted.
paddingFactor	amount of padding to add to the map if no <code>gmap</code> object is provided. The added padding will be equal to <code>paddingFactor</code> multiplied by the height and width of the map (determined by the difference between the maximum and minimum longitudes/latitudes).
lineSize	width of the line that connects the points of the routes in the plot.
lineAlpha	opacity of the line that connects the points of the routes in the plot.
pointSize	size of the points of the routes in the plot.
pointAlpha	opacity of the points of the routes in the plot.
includeArrows	logical indicating if arrows showing the direction of the flight should be added to the plot. The default value of <code>FALSE</code> can speed up the generation of the plot considerably when a large amount of routes are plotted.
arrowLength	length of the segment arrows in centimeters.
literalColors	logical indicating if the values provided through <code>pathColors</code> should be interpreted as color names/codes. By default, <code>TRUE</code> . If set to <code>FALSE</code> , <code>pathColors</code> will be interpreted as a factor, and a color for each different value will be automatically assigned.

Value

A `gmap` object with added paths and points representing the routes.

References

<https://opensky-network.org/apidoc/rest>

Examples

```
# Plot the routes followed by two aircrafts departing from Sevilla airport the  
# 4th of November, 2020.
```

```
if(interactive()){  
  vectors1=getAircraftStateVectorsSeries(aircraft="345107",  
    startTime="2020-11-04 11:55:00", endTime="2020-11-04 13:10:00",  
    timeZone="Europe/London", timeResolution=300)  
  
  vectors2=getAircraftStateVectorsSeries(aircraft = "4ca7b3",  
    startTime="2020-11-04 10:30:00", endTime="2020-11-04 12:00:00",  
    timeZone="Europe/London", timeResolution=300)  
  
  plotRoutes(list(vectors1, vectors2), pathColors=c("red", "blue"))  
}
```

Index

ADSBDecoder, [2](#)

cluster, [4](#)
clusterRoutes, [3](#)

dbscan, [4](#)
dist, [22](#)

getAircraftFlights, [5](#), [22](#)
getAircraftMetadata, [7](#), [19](#)
getAircraftStateVectorsSeries, [8](#), [25](#)
getAirportArrivals, [9](#), [22](#)
getAirportDepartures, [11](#), [22](#)
getAirportMetadata, [12](#), [21](#)
getIntervalFlights, [13](#), [22](#)
getRouteMetadata, [14](#), [23](#)
getSingleTimeStateVectors, [15](#), [24](#), [25](#)
getVectorSetFeatures, [17](#)
getVectorSetListFeatures, [3](#), [4](#), [18](#)

openSkiesAircraft, [7](#), [19](#)
openSkiesAirport, [7](#), [12](#), [21](#)
openSkiesFlight, [6](#), [10](#), [12](#), [14](#), [22](#), [26](#)
openSkiesRoute, [15](#), [23](#)
openSkiesStateVector, [9](#), [16](#), [20](#), [24](#), [25](#)
openSkiesStateVectorSet, [3](#), [4](#), [9](#), [16–20](#),
[22](#), [25](#), [25](#), [27–29](#)

plotPlanes, [26](#)
plotRoute, [27](#)
plotRoutes, [28](#)

R6Class, [2](#), [19](#), [21–25](#)

scale, [18](#)