Package: eurocontrol (via r-universe)

October 12, 2026

Type Package

Title Helper functions for EUROCONTROL useRs

Version 0.1.25

Maintainer Enrico Spinielli <enrico.spinielli@eurocontrol.int>

Description The helper functions in this package are designed to make it easy, homogeneus and transparent to perform common tasks usually needed by data analysts and useRs in EUROCONTROL.

License MIT + file LICENSE

Imports cli, DBI, dbplyr, dplyr, forcats, geosphere, lubridate, magrittr, readr, rlang (>= 0.4.11), stringr, tibble, withr

Encoding UTF-8

LazyData true

Suggests arrow, ROracle, roxygen2, sf

Roxygen list(markdown = TRUE)

URL https://eurocontrol.github.io/eurocontrol/, https://github.com/eurocontrol/eurocontrol

Depends R (>= 4.1.0)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev libx11-dev

Repository https://euctrl-pru.r-universe.dev

Date/Publication 2026-06-26 19:03:07 UTC

RemoteUrl https://github.com/eurocontrol/eurocontrol

RemoteRef HEAD

RemoteSha 09186a4ad9e1ee70484fa5a723572667b44bdda4

Contents

acc_sf . . . 2
adrr_flights_tidy . . . 4
aircraft_model . . . 5
aircraft_type . . . 6
airlines_tbl . . . 6
airlines_tidy . . . 7
airports_oa . . . 8
airspace_profile_tbl . . . 9
airspace_profiles_tidy . . . 10
ansp_sf . . . 12
apdf_tbl . . . 13
apdf_tidy . . . 14
db_connection . . . 16
es_sf . . . 17
fir_sf . . . 18
flights_airspace_profiles_tidy . . . 19
flights_tbl . . . 21
flights_tidy . . . 22
generate_so6 . . . 26
iata_season_for_date . . . 27
member_state . . . 27
point_profile_tbl . . . 28
point_profiles_tidy . . . 29
season_iata . . . 31

Index **32**

`acc_sf` *Retrieve ACC/OAC airspace geometries for a given AIRAC cycle*

Description

Returns ACC (Area Control Centre) and OAC (Oceanic Area Control Centre) airspace geometries as an `sf` object for the specified CFMU AIRAC cycle.

Usage

```
acc_sf(conn = NULL, cfmu_airac)
```

Arguments

`conn`	Database connection or instantiate the default one.
`cfmu_airac`	the AIRAC cycle number (CFMU format), e.g. `"517"` or `517`. Both character and numeric inputs are accepted. See `airac::cfmu_airac()` to look up the cycle number for a given date.

Value

An sf object with the following columns:

- AC_ID: the AIRAC cycle id
- AV_AIRSPACE_ID: the airspace identifier
- MIN_FLIGHT_LEVEL: the minimum flight level
- MAX_FLIGHT_LEVEL: the maximum flight level
- NAME: the airspace name
- CODE: the airspace code
- AIRSPACE_TYPE: the airspace type (ACC or OAC)
- geometry: the airspace polygon geometry

Note

You need to either provide a connection conn that has access to ENV_SP.AIRSPACE, PRU_STAT_AUA and SWH_MAP or go with the default which uses PRU_DEV to establish a db_connection().

The sf package is required but only suggested; install it if not available.

See Also

airac::cfmu_airac() from the airac package.

Examples

```
## Not run:
acc <- acc_sf(cfmu_airac = "517")

# write to GeoJSON
sf::st_write(acc, "acc-517.geojson", driver = "GeoJSON")

# if you re-use DB connections
conn <- eurocontrol::db_connection("PRU_DEV")
acc <- acc_sf(conn = conn, cfmu_airac = "517")

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)

## End(Not run)
```

`adrr_flights_tidy` *Extract ADRR flight list in an interval of dates*

Description

Extract ADRR flight list in an interval of dates

Usage

```
adrr_flights_tidy(conn = NULL, wef, til)
```

Arguments

`conn`	Database connection or instantiate the default one.
`wef`	**W**ith **EF**fect date (included) at Zulu time in a format recognized by `lubridate::as_datetime()`
`til`	un**TIL**l date (excluded) at Zulu time in a format recognized by `lubridate::as_datetime()`

Value

A `dbplyr::tbl_dbi()` with the following columns (as defined in the Aviation Data Repository for Research manual):

Flight details:

- `ECTRL ID`: flight unique id.
- `ADEP`: ICAO code of the **A**erodrome of **DEP**arture
- `ADEP Latitude`: `ADEP` latitude (decimal degrees North)
- `ADEP Longitude`: `ADEP` longitude (decimal degrees East)
- `ADES`: ICAO code of the **A**erodrome of **DES**tination
- `ADES Latitude`: `ADES` latitude (decimal degrees North)
- `ADES Longitude`: `ADES` longitude (decimal degrees East)
- `FILED OFF BLOCK TIME`: (Last) filed off-block time
- `FILED ARRIVAL TIME`: (Last) filed arrival time
- `ACTUAL OFF BLOCK TIME`: actual off-block time for flown (M3) trajectory
- `ACTUAL ARRIVAL TIME`: arrival time for flown (M3) trajectory
- `AC Type`: the ICAO code for the aircraft type, for example A30B for an Airbus A-300B2-200.
- `AC Operator`: the ICAO Airline Designator, i.e. `OAL` for `Olympic`
- `AC Registration`: the aircraft registration (with spaces, dashes, ... stripped), e.g. `GEUUU`
- `ICAO Flight Type`: flight type (see FPL Item 8). Possible values:
  - `S` for scheduled air service
  - `N` for non-scheduled air service
- `STATFOR Market Segment`: market segment type as defined on the Market Segment Rules, it can be:
  - "Mainline"
  - "Regional"

  - "Low-Cost"
  - "Business Aviation"
  - "All-Cargo"
  - "Charter" (Non-Scheduled)
  - "Military"
  - "Other"
  - "Not classified"
- `Requested FL`: requested cruising flight level (FL)
- `Actual Distance Flown (nm)`: route length (in Nautical Miles) for for flown (M3) trajectory

Note

You need to either provide a connection `conn` that has access to `SWH_FCT.DIM_FLIGHT_TYPE_RULE` (for `FLT_RULES`), `PRUDEV.V_COVID_DIM_AO` (for aircraft and aircraft group info) and `SWH_FCT.V_FAC_FLIGHT_MS` (for market segment info) or go with the default which uses PRU_READ to establish a `db_connection()`.

Examples

```
## Not run:
withr::local_envvar(c(TZ = "UTC", ORA_SDTZ = "UTC", NLS_LANG = ".AL32UTF8"))
conn <- withr::local_db_connection(db_connection(schema = "PRU_READ"))
wef <- "2025-01-01"
til <- "2025-01-04"
adrr_flights_tidy(conn, wef, til)

## End(Not run)
```

`aircraft_model` *ICAO's Manufacturer codes*

Description

A data frame with the following fields

model_full_name the model full name, e.g. "A-320neo".

manufacturer_code the manufacturer's code, e.g. "AIRBUS".

designator the model's designator, e.g. "A20N"

last_updated the date when the data have been last updated, e.g. "2023-05-19".

Usage

```
aircraft_model
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 7260 rows and 4 columns.

aircraft_type *ICAO's Aircraft types*

Description

A data frame with the following fields

designator the aircraft type designator, e.g. "A310".

aircraft_description the aircraft description, e.g. "LandPlane".

description the aircraft , e.g. "IT".

wtc the aircraft wake turbulence category, e.g. "M".

engine_count the number pf engines, e.g. "2". **Note**: this is not a number unfortunately, there is one model encoded `C`

engine_type the engine type, e.g. "Jet".

last_updated the date when the data have been last updated, e.g. "2023-05-19".

Usage

```
aircraft_type
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 2612 rows and 7 columns.

airlines_tbl *Return a reference to the Airlines table*

Description

The returned `dbplyr::tbl_dbi()` is referencing the airlines table in PRISME. You can use `dplyr`/`dbplyr` verbs to filter, join, ... with other datasets.

Usage

```
airlines_tbl(conn = NULL)
```

Arguments

`conn`	Database connection or instantiate the default one.

Value

a `dbplyr::tbl_dbi()` referencing the Oracle table for airlines.

Note

You need to either provide a connection conn that has access to PRUDEV.V_COVID_DIM_AO or go with the default which uses PRU_READ to establish a db_connection().

Examples

```
## Not run:
arl <- airlines_tbl()
# other operations on arl, i.e. filtering,
# followed by a collect() to retrieve the concrete data frame
arl_filtered <- arl |>
   dplyr::filter(AO_ISO_CTRY_CODE == "IT") |>
   collect()

# NOTE: you can reuse the connection for other API calls
conn <- arl$src$con

# other ops requiring conn
# ...

# IMPORTANT: close the DB connection
DBI::dbDisconnect(conn)

## End(Not run)
```

---

airlines_tidy *Airline info including group affiliation*

Description

Airline info including group affiliation

Usage

```
airlines_tidy(conn = NULL)
```

Arguments

conn	Optional connection to the PRU_READ schema.

Value

A dbplyr::tbl_dbi() with the following columns:

- AO_CODE: the the ICAO Airline Designator, i.e. 'OAL'
- AO_NAME: the airline's name, i.e. 'Olympic'
- AO_GRP_CODE: the airline's affiliation group code, i.e. 'AEE_GRP'
- AO_GRP_NAME: the airline's affiliation group, i.e. 'AEGEAN Group'

- AO_ISO_CTRY_CODE: the ISO2C code of the airline's country, i.e. 'GR'
- EU: (a character) whether the airlines is in a EUROCONTROL's Member State (full, comprehensive or transition plus Kosovo), i.e. 'TRUE'

Note

You need to either provide a connection conn that has access to PRUDEV.V_COVID_DIM_AO or go with the default which uses PRU_READ to establish a db_connection().

Examples

```
## Not run:
arls <- airlines_tidy()
# other operations on arls, i.e. filtering,
# followed by a collect() to retrieve the concrete data frame
arls_filtered <- arls |>
   filter(stringr::str_starts("A")) |>
   collect()

# NOTE: you can reuse the connection for other API calls
conn <- arls$src$con

# other ops requiring conn
# ...

# IMPORTANT: close the DB connection
DBI::dbDisconnect(conn)

## End(Not run)
```

---

airports_oa *retrieve latest airport list from OurAirports*

Description

retrieve latest airport list from OurAirports

Usage

```
airports_oa()
```

Value

a data frame

Examples

```
## Not run:
apts <- airports_oa()

## End(Not run)
```

---

airspace_profile_tbl *Return a reference to the Airspace Profile table*

Description

The returned `dbplyr::tbl_dbi()` is referencing the airspace profiles table in PRISME. You can use `dplyr/dbplyr` verbs to filter, join, ... with other datasets.

Usage

```
airspace_profile_tbl(conn = NULL)
```

Arguments

`conn`	Database connection or instantiate the default one.

Value

a `dbplyr::tbl_dbi()` referencing the Oracle table for airspace profiles.

Note

You need to either provide a connection `conn` that has access to `FSD.ALL_FT_ASP_PROFILE` or go with the default which uses PRU_READ to establish a `db_connection()`.

Examples

```
## Not run:
pp <- airspace_profile_tbl()
# other operations on pp, i.e. filtering,
# followed by a collect() to retrieve the concrete data frame
# IMPORTANT: close the DB connection when done
DBI::dbDisconnect(pp$src$con)

# if you use a DB connection for many different APIs
conn <- eurocontrol::db_connection("PRU_READ")
pp <- airspace_profile_tbl(conn = conn)

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)

## End(Not run)
```

`airspace_profiles_tidy` *Provide all airspace profile segments intersecting an interval of interest*

Description

The returned `dbplyr::tbl_dbi()` includes segments for scheduled and non-scheduled flights temporally intersecting the right-opened interval `[wef, til)`.

General aviation, State, military and sensitive flight are excluded.

Usage

```
airspace_profiles_tidy(
  conn = NULL,
  wef,
  til,
  airspace = "FIR",
  profile = "CTFM"
)
```

Arguments

`conn`	Database connection or instantiate the default one.
`wef`	**W**ith **EF**fect date (included) at Zulu time in a format recognized by `lubridate::as_datetime()`
`til`	un**TIL**l date (excluded) at Zulu time in a format recognized by `lubridate::as_datetime()`
`airspace`	the type of airspace (default: 'FIR'), one of: • 'FIR' (Flight Information Region) • 'NAS' (National Airspace) • 'AUA' (ATC Unit Airspace) • 'ES' (Elementary Sector)
`profile`	the model of the trajectory profile (default: 'CTFM'), one of: • 'FTFM', Filed Tactical Flight Model • 'RTFM', Regulated Tactical Flight Model • 'CTFM', Current Tactical Flight Model • 'CPF', Correlated Position reports for a Flight • 'DCT', Direct route • 'SCR', Shortest Constrained Route • 'SRR', Shortest RAD restrictions applied Route • 'SUR', Shortest Unconstrained Route

Value

a dbplyr::tbl_dbi() with the following columns

- ID: the so called SAM ID, used internally by PRISME
- FLT_UID: flight unique id.
- SEQ_ID: the sequence number of the segment for the relevant airspace profile
- ENTRY_TIME: the time of entry into the relevant airspace
- ENTRY_LON: the longitude of entry into the relevant airspace
- ENTRY_LAT: the latitude of entry into the relevant airspace
- ENTRY_FL: the flight level of entry into the relevant airspace
- EXIT_TIME: the time of exit out of the relevant airspace
- EXIT_LON: the longitude of exit out of the relevant airspace
- EXIT_LAT: the latitude of exit out of the relevant airspace
- EXIT_FL: the flight level of exit out of the relevant airspace
- AIRSPACE_ID: the airspace ID
- AIRSPACE_TYPE: the airspace type as per airspace input parameter
- MODEL_TYPE: the trajectory model as per profile input parameter

Note

You need to either provide a connection conn that has access to as noted in airspace_profile_tbl() and flights_tidy() or go with the default which uses PRU_READ to establish a db_connection().

Examples

```
## Not run:
ps <- airspace_profiles_tidy(wef = "2023-01-01", til = "2023-04-01")
# IMPORTANT: always close the DB connection when done
DBI::dbDisconnect(ps$src$con)

# if you re-use DB connections
conn <- eurocontrol::db_connection("PRU_READ")
ps <- airspace_profiles_tidy(conn = conn)

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)

## End(Not run)
```

ansp_sf *Retrieve ACE ANSP airspace geometries for a given AIRAC cycle*

Description

Returns ANSP (Air Navigation Service Provider) airspace geometries from the ACE (ATM Cost-Effectiveness) benchmarking dataset as an `sf` object for the specified CFMU AIRAC cycle.

Airspaces with code `AIRPORT`, `UNKNOWN` or `MILITARY` are excluded.

Usage

```
ansp_sf(conn = NULL, cfmu_airac)
```

Arguments

`conn`	Database connection or instantiate the default one.
`cfmu_airac`	the AIRAC cycle number (CFMU format), e.g. `"517"` or `517`. Both character and numeric inputs are accepted. See `airac::cfmu_airac()` to look up the cycle number for a given date.

Value

An `sf` object with the following columns:

- `airac_cfmu`: the AIRAC cycle id
- `id`: the airspace identifier
- `code`: the ANSP code
- `name`: the ANSP name
- `ace_code`: the ACE benchmarking code
- `min_fl`: the minimum flight level
- `max_fl`: the maximum flight level
- `airspace_type`: the airspace type
- `geometry`: the airspace polygon geometry

Note

You need to either provide a connection `conn` that has access to `ENV_SP.AIRSPACE`, `PRU_CFMU_ANSP` and `SWH_MAP` or go with the default which uses PRU_DEV to establish a `db_connection()`.

The `sf` package is required but only suggested; install it if not available.

See Also

`airac::cfmu_airac()` from the airac package.

Examples

```
## Not run:
ansps <- ansp_sf(cfmu_airac = "481")

# write to GeoJSON
sf::st_write(ansps, "ansp_ace_481.geojson", driver = "GeoJSON")

# if you re-use DB connections
conn <- eurocontrol::db_connection("PRU_DEV")
ansps <- ansp_sf(conn = conn, cfmu_airac = "481")

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)

## End(Not run)
```

---

apdf_tbl *Return a reference to the Airport Operator Data Flow table*

Description

The returned `dbplyr::tbl_dbi()` is referencing the airport operator data flow table in PRISME. You can use `dplyr`/`dbplyr` verbs to filter, join, ... with other datasets.

Usage

```
apdf_tbl(conn = NULL)
```

Arguments

conn	Database connection or instantiate the default one.

Value

a `dbplyr::tbl_dbi()` referencing the Oracle table for airport operator data flow.

Note

You need to either provide a connection `conn` that has access to `SWH_FCT.FAC_APDS_FLIGHT_IR691` or go with the default which uses PRU_READ to establish a `db_connection()`.

Examples

```
## Not run:
apdf <- apdf_tbl()
# ...
# IMPORTANT: close the DB connection when done with `aodf`
DBI::dbDisconnect(apdf$src$con)

## End(Not run)
```

---

apdf_tidy *Extract a clean airport operator data flow list in an interval*

Description

The returned `dbplyr::tbl_dbi()` includes movements information in the interval `[wef, til)`. **NOTE**: it can only cover ONE month at a time

Usage

```
apdf_tidy(conn = NULL, wef, til)
```

Arguments

`conn`	Database connection or instantiate the default one.
`wef`	**W**ith **EF**fect date (included) at Zulu time in a format recognized by `lubridate::as_datetime()`
`til`	un**TIL**l date (excluded) at Zulu time in a format recognized by `lubridate::as_datetime()`

Value

A `dbplyr::tbl_dbi()` with the following columns:

- APDS_ID: the airport operator dataflow unique record id.
- ID: the so called `SAM ID`, used internally by PRISME
- AP_C_FLTID: flight identifier (aource Airport)
- AP_C_FLTRUL: which sets of regulations the flight is operated under. Possible values are:
  - `IFR` for IFR
  - `VFR` for VFR
  - `NA` if unknown
- AP_C_REG: the aircraft registration (with spaces, dashes, ... stripped), e.g. GEUUU.
- ADEP_ICAO: (ICAO code of the) **A**erodrome of **DEP**arture (source airport).
- ADES_ICAO: (ICAO code of the) **A**erodrome of **DES**tination (source airport).
- SRC_PHASE: flight phase. `DEP`=departure, `ARR`=arrival.

- MVT_TIME_UTC: (best available) movement time (takeoff if SRC_PHASE = DEP, landing if SRC_PHASE = ARR).
- BLOCK_TIME_UTC: Block time (off-block if SRC_PHASE = DEP, in-block if SRC_PHASE = ARR).
- SCHED_TIME_UTC: scheduled time (of departure if SRC_PHASE = DEP, of arrival if SRC_PHASE = ARR; source airport).
- ARCTYP: (best available) the ICAO code for the aircraft type, for example A21N for Airbus A321neo.
- AP_C_RWY: Runway ID (of departure if SRC_PHASE = DEP, of arrival if SRC_PHASE = ARR; source airport).
- AP_C_STND: Stand ID (of departure if SRC_PHASE = DEP, of arrival if SRC_PHASE = ARR; source airport).
- C40_CROSS_TIME: time of first (last) crossing at 40 NM from ARP for departure (arrival).
- C40_CROSS_LAT: latitude of first (last) crossing at 40 NM from ARP for departure (arrival).
- C40_CROSS_LON: longitude of first (last) crossing at 40 NM from ARP for departure (arrival).
- C40_CROSS_FL: flight level of first (last) crossing at 40 NM from ARP for departure (arrival).
- C40_BEARING: bearing of first (last) crossing at 40 NM from ARP for departure (arrival).
- C100_CROSS_TIME: time of first (last) crossing at 100 NM from ARP for departure (arrival).
- C100_CROSS_LAT: latitude of first (last) crossing at 100 NM from ARP for departure (arrival).
- C100_CROSS_LON: longitude of first (last) crossing at 100 NM from ARP for departure (arrival).
- C100_CROSS_FL: flight level of first (last) crossing at 100 NM from ARP for departure (arrival).
- C100_BEARING: bearing of first (last) crossing at 100 NM from ARP for departure (arrival).

Note

You need to either provide a connection conn that has access to SWH_FCT.FAC_APDS_FLIGHT_IR691, or go with the default which uses PRU_READ to establish a db_connection().

Examples

```
## Not run:
my_apdf <- apdf_tidy(wef = "2023-01-01", til = "2023-01-02")
# ...
DBI::dbDisconnect(my_apdf$src$con)

## End(Not run)
```

db_connection *Provide a connection to the relevant Oracle database*

Description

Provide a connection to the relevant Oracle database

Usage

```
db_connection(schema = "PRU_READ")
```

Arguments

schema	the Oracle DB schema to connect to.

Value

A connection to a database (specifically an implementation of DBI::DBIConnection for an Oracle database.)

Note

The schema is in fact the prefix of the environment variables where the credentials are stored, like <schema>_USR, <schema>_PWD and <schema>_DBNAME. Possible values for schema are PRU_READ, PRU_PROD, PRU_DEV, PRU_TEST, ...

Examples

```
## Not run:
# Within a function call (or interactive session, a restart will close the
# `withr` calls) you should use the following snippet of code
withr::local_envvar(c(TZ = "UTC", ORA_SDTZ = "UTC", NLS_LANG = ".AL32UTF8"))
# connect with PRU_READ credentials
conn <- db_connection("PRU_READ") |> withr::local_db_connection()
# or alternativelly more basically (you need to explicitly close the
# the DB connection):
conn <- db_connection()
# ... perform other API operations re-using the same connection
# ...
DBI::dbDisconnect(conn)

## End(Not run)
```

es_sf *Retrieve Elementary Sector airspace geometries for a given AIRAC cycle*

Description

Returns ES (Elementary Sector) airspace geometries as an `sf` object for the specified CFMU AIRAC cycle.

Usage

```
es_sf(conn = NULL, cfmu_airac)
```

Arguments

`conn`	Database connection or instantiate the default one.
`cfmu_airac`	the AIRAC cycle number (CFMU format), e.g. `"517"` or `517`. Both character and numeric inputs are accepted. See `airac::cfmu_airac()` to look up the cycle number for a given date.

Value

An `sf` object with the following columns:

- `AC_ID`: the AIRAC cycle id
- `AV_AIRSPACE_ID`: the airspace identifier
- `MIN_FLIGHT_LEVEL`: the minimum flight level
- `MAX_FLIGHT_LEVEL`: the maximum flight level
- `NAME`: the airspace name
- `CODE`: the airspace code
- `AIRSPACE_TYPE`: the airspace type (`ES`)
- `geometry`: the airspace polygon geometry

Note

You need to either provide a connection `conn` that has access to `ENV_SP.AIRSPACE`, `PRU_CFMU_ES` and `SWH_MAP` or go with the default which uses PRU_DEV to establish a `db_connection()`.

The `sf` package is required but only suggested; install it if not available.

See Also

`airac::cfmu_airac()` from the airac package.

Examples

```
## Not run:
es <- es_sf(cfmu_airac = "517")

# write to GeoJSON
sf::st_write(es, "es-517.geojson", driver = "GeoJSON")

# if you re-use DB connections
conn <- eurocontrol::db_connection("PRU_DEV")
es <- es_sf(conn = conn, cfmu_airac = "517")

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)

## End(Not run)
```

`fir_sf` *Retrieve FIR (Flight Information Region) airspace geometries for a given AIRAC cycle*

Description

Returns FIR (Flight Information Region) airspace geometries as an `sf` object for the specified CFMU AIRAC cycle.

An `icao` column is derived from the first two characters of the airspace `code`.

Usage

```
fir_sf(conn = NULL, cfmu_airac)
```

Arguments

`conn`	Database connection or instantiate the default one.
`cfmu_airac`	the AIRAC cycle number (CFMU format), e.g. `"517"` or `517`. Both character and numeric inputs are accepted. See `airac::cfmu_airac()` to look up the cycle number for a given date.

Value

An `sf` object with the following columns:

- `airac_cfmu`: the AIRAC cycle id
- `icao`: the 2-letter ICAO prefix (derived from `code`)
- `id`: the airspace volume id
- `code`: the FIR code

- name: the FIR name
- min_fl: the minimum flight level
- max_fl: the maximum flight level
- airspace_type: the airspace type (FIR)
- geometry: the airspace polygon geometry

Note

You need to either provide a connection conn that has access to ENV_SP.AIRSPACE_VOLUME, PRU_CFMU_FIR and SWH_MAP or go with the default which uses PRU_DEV to establish a db_connection().

The sf package is required but only suggested; install it if not available.

See Also

airac::cfmu_airac() from the airac package.

Examples

```
## Not run:
firs <- fir_sf(cfmu_airac = "517")

# write to GeoJSON
sf::st_write(firs, "ir-517.geojson", driver = "GeoJSON")

# if you re-use DB connections
conn <- eurocontrol::db_connection("PRU_DEV")
firs <- fir_sf(conn = conn, cfmu_airac = "517")

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)

## End(Not run)
```

---

flights_airspace_profiles_tidy

Extract the segments intersecting a set of airspace IDs in an interval of time and complement them with flight information

---

Description

The returned dbplyr::tbl_dbi() includes scheduled and non-scheduled flights whose airspace segments temporally intersecting the right-opened interval [wef, til). General aviation, State, military and sensitive flight are excluded.

Usage

```
flights_airspace_profiles_tidy(
  conn = NULL,
  wef,
  til,
  airspaces,
  airspace = "FIR",
  profile = "CTFM"
)
```

Arguments

conn	Database connection or instantiate the default one.
wef	**W**ith **EF**fect date (included) at Zulu time in a format recognized by `lubridate::as_datetime()`
til	un**TIL**l date (excluded) at Zulu time in a format recognized by `lubridate::as_datetime()`
airspaces	list of airspace ids
airspace	the type of airspace (default: 'FIR'), one of: • 'FIR' (Flight Information Region) • 'NAS' (National Airspace) • 'AUA' (ATC Unit Airspace) • 'ES' (Elementary Sector)
profile	the model of the trajectory profile (default: 'CTFM'), one of: • 'FTFM', Filed Tactical Flight Model • 'RTFM', Regulated Tactical Flight Model • 'CTFM', Current Tactical Flight Model • 'CPF', Correlated Position reports for a Flight • 'DCT', Direct route • 'SCR', Shortest Constrained Route • 'SRR', Shortest RAD restrictions applied Route • 'SUR', Shortest Unconstrained Route

Value

a `dbplyr::tbl_dbi()` with the same columns as `flights_tidy()` plus the airspace id, `AIRSPACE_ID`, being crossed, one row per crossed airspace.

Note

You need to either provide a connection `conn` that has access to as noted in `airspace_profile_tbl()` and `flights_tidy()` or go with the default which uses PRU_READ to establish a `db_connection()`.

Examples

```
## Not run:
withr::local_envvar(c(TZ = "UTC", ORA_SDTZ = "UTC", NLS_LANG = ".AL32UTF8"))
conn <- withr::local_db_connection(db_connection("PRU_READ"))
flights_airspace_profiles_tidy(
  conn = conn,
  wef = "2023-01-01",
  til = "2023-04-01",
  airspace = "ES",
  profile = "CTFM",
  airspaces = c("LYBALW2", "LYBATW2")
)

## End(Not run)
```

---

flights_tbl *Return a reference to the Flights table*

Description

The returned `dbplyr::tbl_dbi()` is referencing the flights table in PRISME. You can use `dplyr/dbplyr` verbs to filter, join, ... with other datasets.

Usage

```
flights_tbl(conn = NULL)
```

Arguments

conn	Database connection or instantiate the default one.

Value

a `dbplyr::tbl_dbi()` object referencing the Oracle table for flights.

Note

You need to either provide a connection conn that has access to `SWH_FCT.V_FAC_FLIGHT_MS` or go with the default which uses PRU_READ to establish a `db_connection()`. Market Segment is not available before 2004.

Examples

```
## Not run:
flt <- flights_tbl()
# other operations on flt, i.e. filtering,
# followed by a collect() to retrieve the concrete data frame
flt_filtered <- flt |>
   filter(TO_DATE("2023-06-01 10:00", "YYYY-MM-DD HH24:MI") <= IOBT,
```

```
                    IOBT < TO_DATE("2023-06-02 10:30", "YYYY-MM-DD HH24:MI")) |>
  collect()


# NOTE: you can reuse the connection for other API calls
conn <- flt$src$con

# other ops requiring conn
# ...

# IMPORTANT: close the DB connection
DBI::dbDisconnect(conn)

## End(Not run)
```

`flights_tidy` *Extract a clean flights list in an interval*

Description

The returned `dbplyr::tbl_dbi()` includes scheduled and non-scheduled flight departing in the right-opened interval `[wef, til)`.

Defaults values will assure that General aviation, State, military and sensitive flight will excluded. They can be retrieved via the other function call arguments in case of need.

Usage

```
flights_tidy(
  conn = NULL,
  wef,
  til,
  icao_flt_types = c("S", "N"),
  ids = NULL,
  include_sensitive = FALSE,
  include_military = FALSE,
  include_head = FALSE,
  extra_cols = NULL
)
```

Arguments

`conn`	Database connection or instantiate the default one.
`wef`	**W**ith **EF**fect date (included) at Zulu time in a format recognized by `lubridate::as_datetime()`
`til`	un**TIL**l date (excluded) at Zulu time in a format recognized by `lubridate::as_datetime()`

icao_flt_types — the types of flights as described below in `ICAO_FLT_TYPE`, default `c('S', 'N')`, NULL includes all notwithstanding other argument options. When including military via `include_military` you should either pass NULL or make sure 'M' is included

ids — list of `IDs` (aka `SAM ID`) to return, default NULL for all flights

include_sensitive — include sensitive flights, default FALSE

include_military — include military flights, default FALSE

include_head — include Head of State flights, default FALSE

extra_cols — a character vector with the name of additional columns from `V_FAC_FLIGHT_MS`, default NULL. This can be useful for example to get equipment data, i.e. `ICAO_EQPT` or similar (`COM_EQPT`, `DATA_LINK_EQPT`, `NAV_EQPT`, `SSR_EQPT`, `SURVIVAL_EQPT`).

Value

A `dbplyr::tbl_dbi()` with the following columns (grouped here by flight details, aerodrome details, aircraft info, aircraft operator info and operational details):

Flight details:

- FLT_UID: flight unique id.
- ID: the so called `SAM ID`, used internally by PRISME
- AIRCRAFT_ID: the callsign of the relevant flight, e.g. BAW6VB.
- LOBT: **L**ast received **O**ff-**B**lock **T**ime.
- IOBT: **I**nitial **O**ff-**B**lock **T**ime.
- FLT_RULES (see FPL Item 8): which sets of regulations the flight is operated under. Possible values are:
  - `I` for IFR
  - `V` for VFR
  - `Y` first IFR thereafter VFR
  - `Z` first VFR thereafter IFR
- ICAO_FLT_TYPE (see FPL Item 8): flight type. Possible values:
  - `S` for scheduled air service
  - `N` for non-scheduled air service
  - `G` for general aviation
  - `M` for military (note: filtered out)
  - `X` for other than the preceding categories
- RULE_NAME: market segment type as defined on the Market Segment Rules, it can be:
  - "Mainline"
  - "Regional"
  - "Low-Cost"
  - "Business Aviation"
  - "All-Cargo"
  - "Charter" (Non-Scheduled)

  - "Military"
  - "Other"
  - "Not classified"
- SENSITIVE: 'Y' if sensitive
- SPECIAL_EXEMPT: reasons for special handling by ATS. One of:
  - "AEAP" ATFM exemption approved
  - "EMER" emergency
  - "FIRE" fire fighting
  - "HEAD" flights with Head of State status
  - "MEDE" medical evacuation
  - "NEXE" not exempted
  - "SERE" search & rescue

Aerodrome details:

- ADEP: ICAO code of the **A**erodrome of **DEP**arture
- NAME_ADEP: the (AIU) name of the `ADEP` airport
- COUNTRY_CODE_ADEP: the ISO 2-alpha country code for `ADEP`
- COUNTRY_NAME_ADEP: the country name for `ADEP`
- ADES: ICAO code of the **A**erodrome of **DES**tination (different from `ADES_FILED` in case of diversion)
- NAME_ADES: the (AIU) name of the `ADES` airport
- COUNTRY_CODE_ADES: the ISO 2-alpha country code for `ADES`
- COUNTRY_NAME_ADES: the country name for `ADES`
- ADES_FILED: ICAO code of the **A**erodrome of **DES**tination filed in the Flight Plan. **Note**: it can be different from `ADES` in case of diversion
- NAME_ADES_FILED: the (AIU) name of the `ADES_FILED` airport
- COUNTRY_CODE_ADES_FILED: the ISO 2-alpha country code for `ADES_FILED`
- COUNTRY_NAME_ADES_FILED: the country name for `ADES_FILED`

Aircraft info:

- REGISTRATION: the aircraft registration (with spaces, dashes, ... stripped), e.g. GEUUU.
- AIRCRAFT_ADDRESS: the ICAO 24-bit address of the airframe for ADS-B/Mode S broadcasting.
- AIRCRAFT_TYPE_ICAO_ID: the ICAO code for the aircraft type, for example A30B for an Airbus A-300B2-200.
- WK_TBL_CAT (see FPL Item 9): wake turbulence category, can be
  - `L` LIGHT, i.e. maximum certificated takeoff mass of 7000 kg (15_500 lbs) or less.
  - `M` MEDIUM, i.e maximum certificated takeoff mass less than 136_000 kg (300_000 lbs), but more than 7_000 kg (15_500 lbs)
  - `H` HEAVY, i.e. maximum certificated takeoff mass of 136_000 kg (300_000 lbs) or more (except those specified as `J`)
  - `J` SUPER, presently the only the AIRBUS A-380-800

Aircraft operator details:

- AIRCRAFT_OPERATOR: the ICAO Airline Designator, i.e. `OAL` for `Olympic`
- AO_GRP_CODE: Aircraft Operator group (code), i.e. AEE_GRP
- AO_GRP_NAME: : Aircraft Operator group (name), i.e. AEGEAN Group
- AO_ISO_CTRY_CODE: ISO country code for AO

Operational details:

- EOBT_1: **E**stimated **O**ff-**B**lock **T**ime for FPL-based (M1) trajectory
- ARVT_1: **AR**i**V**al **T**ime for FPL-based (M1) trajectory
- TAXI_TIME_1: Taxi time for FPL-based (M1) trajectory
- AOBT_3: **A**ctual **O**ff-**B**lock **T**ime for flown (M3) trajectory
- ARVT_3: **AR**i**V**al **T**ime for flown (M3) trajectory
- TAXI_TIME_3: Taxi time for flown (M3) trajectory
- RTE_LEN_1: route length (in Nautical Miles) for FPL-based (M1) trajectory
- RTE_LEN_3: route length (in Nautical Miles) for for flown (M3) trajectory
- FLT_DUR_1: route duration (in minutes) for FPL-based (M1) trajectory
- FLT_DUR_3: route length (in minutes) for flown (M3) trajectory
- FLT_TOW: takeoff weight

Additional details:

- all extra columns as from `extra_cols`

Note

You need to either provide a connection `conn` that has access to `SWH_FCT.DIM_FLIGHT_TYPE_RULE` (for `FLT_RULES`), `PRUDEV.V_COVID_DIM_AO` (for aircraft and aircraft group info) and `SWH_FCT.V_FAC_FLIGHT_MS` (for market segment info) or go with the default which uses PRU_READ to establish a `db_connection()`.

Examples

```
## Not run:
flts <- flights_tidy(wef = "2023-01-01", til = "2023-01-05")
# other operations on flts, i.e. filtering,
# followed by a collect() to retrieve the concrete data frame
flts_filtered <- flts |>
   filter(TO_DATE("2023-06-01 10:00", "YYYY-MM-DD HH24:MI") <= IOBT,
                  IOBT < TO_DATE("2023-01-02 10:30", "YYYY-MM-DD HH24:MI")) |>
   collect()

# NOTE: you can reuse the connection for other API calls
conn <- flts$src$con

# other ops requiring conn
# ...

# IMPORTANT: close the DB connection
DBI::dbDisconnect(conn)

## End(Not run)
```

`generate_so6` *Export trajectory profiles to SO6 format*

Description

The data frame for point trajectories needs to have the following columns:

Name	Description	Type
FLIGHT_ID	Flight ID	int
TIME_OVER	Time over point	datetime
LONGITUDE	Longitude (decimal degrees)	double
LATITUDE	Latitude (decimal degrees)	double
FLIGHT_LEVEL	Flight level	int
POINT_ID	Point ID or NO_POINT	char
AIR_ROUTE	Air route or NO_ROUTE	char
LOBT	Last Off-block Time	datetime
SEQ_ID	Positions' sequence number	int
CALLSIGN	Flight call sign	char
REGISTRATION	Aircraft registration	char
MODEL_TYPE	Aircraft model	char
AIRCRAFT_TYPE	Aircraft ICAO type	char
AIRCRAFT_OPERATOR	Aircraft operator	char
ADEP	Departing aerodrome (ICAO) ID	char
ADES	Destination aerodrome (ICAO) ID	char

Usage

```
generate_so6(trajectory)
```

Arguments

`trajectory` A data frame for point profile trajectories.

Value

A data frame for trajectories in SO6 format.

Examples

```
## Not run:
conn <- eurocontrol::db_connection("PRU_READ")
pf <- point_profiles_tidy(conn = conn,
                          wef = "2020-01-01",
                          til = "2020-01-10") |>
  generate_so6()

# ... do something else with conn
# ...
```

```
# then manually close the connection to the DB
DBI::dbDisconnect(conn)
generate_so6(trj)

## End(Not run)
```

`iata_season_for_date` *Return the corresponding IATA season for a date*

Description

Return the corresponding IATA season for a date

Usage

```
iata_season_for_date(date)
```

Arguments

`date`	a date

Value

the name of the IATA season in the form `summer-yyyy`

Examples

```
## Not run:
season_iata("2024-04-01")

## End(Not run)
```

`member_state` *EUROCONTROL's Member States*

Description

A data frame with the following fields

name the country name, e.g. "Italy"

iso3c the 3-letter ISO code, e.g. "ITA"

iso2c the 2-letter ISO code, e.g. "IT"

icao the 2-letter ICAO code, e.g. "LI"

iso3n the 3-digits ISO code, e.g. "380"

date the date of `status` code, e.g. 1996-04-01

status the status code, e.g. "M" (`M` Member State, `C` Comprehensive Agreement State, `T` Transitional State, NA for Kosovo)

These are useful to grab the right spatial polygons in case of need.

Usage

```
member_state
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 45 rows and 7 columns.

Note

Kosovo is also included in the list.

---

point_profile_tbl *Return a reference to the Point Profile table*

Description

The returned `dbplyr::tbl_dbi()` is referencing the point profiles table in PRISME. You can use `dplyr/dbplyr` verbs to filter, join, ... with other datasets.

Usage

```
point_profile_tbl(conn = NULL)
```

Arguments

`conn`	Database connection or instantiate the default one.

Value

a `dbplyr::tbl_dbi()` referencing the Oracle table for point profiles.

Note

You need to either provide a connection `conn` that has access to `FSD.ALL_FT_POINT_PROFILE` or go with the default which uses PRU_READ to establish a `db_connection()`.

Examples

```
## Not run:
pt <- point_profile_tbl()

# if you re-use DB connections
conn <- eurocontrol::db_connection("PRU_READ")
pt <- point_profile_tbl(conn = conn)

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)
```

```
## End(Not run)
```

`point_profiles_tidy` *Export point profile from NM trajectories*

Description

Extract NM point profile trajectories from PRISME database. When a bbox is defined, we return only the (full) point profiles for the flights flying thru the region.

Usage

```
point_profiles_tidy(
  conn = NULL,
  wef,
  til = lubridate::today(tzone = "UTC"),
  profile = "CTFM",
  bbox = NULL
)
```

Arguments

`conn`	Database connection or instantiate the default one.
`wef`	**W**ith **EF**fect date (included) at Zulu time in a format recognized by `lubridate::as_datetime()`
`til`	un**TIL**l date (excluded) at Zulu time in a format recognized by `lubridate::as_datetime()`
`profile`	the model of the trajectory profile (default: 'CTFM'), one of: • 'FTFM', Filed Tactical Flight Model • 'RTFM', Regulated Tactical Flight Model • 'CTFM', Current Tactical Flight Model • 'CPF', Correlated Position reports for a Flight • 'DCT', Direct route • 'SCR', Shortest Constrained Route • 'SRR', Shortest RAD restrictions applied Route • 'SUR', Shortest Unconstrained Route
`bbox`	(Optional) axis aligned bounding box (xmin, ymin, xmax, ymax)

Value

a dataframe representing a flight trajectory with the following columns:

- FLIGHT_ID: a unique identifier for the flight
- TIME_OVER: the time over llon/lat
- LONGITUDE: the longitude

- LATITUDE: the latitude
- FLIGHT_LEVEL: the flight level
- POINT_ID: the published point ID ('NO_POINT' otherwise)
- AIR_ROUTE: the air rout name ('DCT' otherwise)
- LOBT: the last off-block time
- SEQ_ID: the progressive sequence number in the trajectory points
- CALLSIGN: the callsign of the flight
- REGISTRATION: the aircraft registration
- MODEL_TYPE: the trajectory model as per `profile` input parameter
- AIRCRAFT_TYPE: the ICAO aircraft type
- AIRCRAFT_OPERATOR: the flight operator
- ICAO24: the ICAO 24-bit address of the aircraft
- ADEP: the Aerodrom of Departure
- ADES: the aerodrome of Destination

Note

You need to either provide a connection conn that has access to as noted in `airspace_profile_tbl()` and `flights_tidy()` or go with the default which uses PRU_READ to establish a `db_connection()`.

Examples

```
## Not run:
# export 1 day of NM (planned) trajectories
pf1 <- point_profiles_tidy(wef = "2019-07-14",
                           til = "2019-07-15",
                           profile = "FTFM")

# export 2 hours of NM (flown) trajectories
pf2 <- point_profiles_tidy(wef = "2019-07-14 22:00",
                           til = "2019-07-15")

# export 1 day of NM (flown) trajectories
pf3 <- point_profiles_tidy(wef = "2019-07-14",
                           til = "2019-07-15",
                           profile = "CTFM")

# export all CTFM trajectories within a bounding box 40 NM around EDDF
bb <- c(xmin = 7.536746, xmax = 9.604390, ymin = 49.36732, ymax = 50.69920)
pf4 <- point_profiles_tidy(wef = "2019-01-01 00:00",
                           til = "2019-01-02 00:00",
                           bbox = bb)


# if you re-use DB connections
conn <- eurocontrol::db_connection("PRU_READ")
pf <- point_profiles_tidy(conn = conn,
```

```
                              wef = "2020-01-01",
                              til = "2020-01-10")

# ... do something else with conn
# ...
# then manually close the connection to the DB
DBI::dbDisconnect(conn)

## End(Not run)
```

`season_iata` *return the interval for an IATA season*

Description

IATA summer season begins on the last Sunday of March and ends on the last Saturday of October. IATA winter season begins on the last Sunday of October and ends Saturday of before next year summer season.

Usage

```
season_iata(year, season = "summer")
```

Arguments

`year`	the year for the season definition
`season`	the (northern hemisphere) season, either "summer" (default) or "winter"

Value

an interval for the season definition, end/start dates are inclusive

Examples

```
## Not run:
season_iata(2019)

## End(Not run)
```

Index

∗ **datasets**
    aircraft_model, 5
    aircraft_type, 6
    member_state, 27
∗ **read/export**
    point_profiles_tidy, 29

acc_sf, 2
adrr_flights_tidy, 4
airac::cfmu_airac(), *2, 3*, *12*, *17–19*
aircraft_model, 5
aircraft_type, 6
airlines_tbl, 6
airlines_tidy, 7
airports_oa, 8
airspace_profile_tbl, 9
airspace_profile_tbl(), *11*, *20*, *30*
airspace_profiles_tidy, 10
ansp_sf, 12
apdf_tbl, 13
apdf_tidy, 14

db_connection, 16
db_connection(), *3*, *5*, *7–9*, *11–13*, *15*, *17*, *19–21*, *25*, *28*, *30*
DBI::DBIConnection, *16*
dbplyr::tbl_dbi(), *4*, *6, 7*, *9–11*, *13, 14*, *19–23*, *28*

es_sf, 17

fir_sf, 18
flights_airspace_profiles_tidy, 19
flights_tbl, 21
flights_tidy, 22
flights_tidy(), *11*, *20*, *30*

generate_so6, 26

iata_season_for_date, 27

lubridate::as_datetime(), *4*, *10*, *14*, *20*, *22*, *29*

member_state, 27

point_profile_tbl, 28
point_profiles_tidy, 29

season_iata, 31