FlightplanDB-py

Release 0.8.1

PH-KDX

May 16, 2023

DOCS

1 Welcome **1**

1.1 User docs . . . 1

1.1.1 Introduction . . . 1

1.1.1.1 About . . . 1

1.1.1.2 Prerequisites . . . 1

1.1.1.3 Installation . . . 1

1.1.1.4 Testing . . . 2

1.1.1.5 Request Limits . . . 2

1.1.1.6 Authentication . . . 2

1.1.2 Quickstart . . . 3

1.1.2.1 Example . . . 3

1.1.3 Changelog . . . 4

1.1.3.1 0.8.1 . . . 4

1.1.3.2 0.8.0 . . . 4

1.1.3.3 0.7.2 . . . 4

1.1.3.4 0.7.1 . . . 4

1.1.3.5 0.7.0 . . . 4

1.1.3.6 0.6.0 . . . 4

1.1.3.7 0.5.0 . . . 5

1.1.3.8 0.4.1 . . . 5

1.1.3.9 0.4.0 . . . 5

1.1.3.10 0.3.2 . . . 5

1.1.3.11 0.3.1 . . . 5

1.1.3.12 0.3.0 . . . 5

1.1.3.13 0.2 . . . 5

1.1.3.14 0.1-alpha . . . 5

1.2 API reference . . . 6

1.2.1 API . . . 6

1.2.2 Nav . . . 7

1.2.3 Plan . . . 9

1.2.4 Tags . . . 13

1.2.5 User . . . 13

1.2.6 Weather . . . 15

1.2.7 Datatypes . . . 15

1.2.8 Exceptions . . . 32

1.2.9 Internal functions . . . 33

2 Indices and tables **39**

Python Module Index **41**

Index 43

CHAPTER

ONE

WELCOME

If you're new, check out the *Introduction* or *Quickstart* sections.

1.1 User docs

1.1.1 Introduction

1.1.1.1 About

This is a Python 3 wrapper for the Flight Plan Database API. Flight Plan Database is a website for creating and sharing flight plans for use in flight simulation. For more information on Flight Plan Database, see their excellent About page.

1.1.1.2 Prerequisites

FlightplanDB-py is supported for Python 3.8 or higher. Python 3.7 would probably have worked with the library, but is not officially supported; the absence of AsyncMock means that the unittests will not execute. Python 3.6 or lower will not work due to dataclasses, which were introduced with PEP 557, being used in the library.

1.1.1.3 Installation

The easiest way to install the library is from PyPi, by running

```
$ pip install flightplandb
```

Or, if you prefer, install the directly from the GitHub repo:

```
$ pip install https://github.com/PH-KDX/flightplandb-py/archive/main.zip
```

after which the package and its dependencies are installed.

If you've never used `pip` before, check out this useful overview.

Virtual Environments

It is, of course, possible to install the library in a virtual environment. Start by going to your project's working directory. Create a virtual environment called, for example, `foo` as follows:

```
$ python3 -m venv foo
```

then when you want to use it, activate it on Linux or macOS with

```
$ source foo/activate/bin
```

or on Windows with

```
$ foo\Scripts\activate.bat
```

after which you can install the library as described in *Installation*.

1.1.1.4 Testing

To test if the package has correctly installed, open a Python shell (note: if you're using a virtual environment, make sure you activate it first) and run

```
import flightplandb
import asyncio
asyncio.run(flightplandb.api.ping())
```

which should return `StatusResponse(message='OK', errors=None)` if all has gone well.

1.1.1.5 Request Limits

API requests are rate limited on a 24 hour rolling basis to ensure fair access to all users. If you reach your daily limit, a *TooManyRequestsException()* will be raised on your requests. To check your limit and used requests, look at the output of *flightplandb.api.limit_cap()* and *flightplandb.api.limit_used()* respectively. These calls, together with *flightplandb.api.ping()*, will not increment your request counter.

The limit for unauthenticated users is IP-based, and is currently set to 100. The limit for authenticated users is key-based, and is currently set to 2500.

Please note that some functions which return an iterable, such as the user search or plan search, can make multiple HTTP requests to fetch all the paginated information, thus increasing your request count by more than 1.

1.1.1.6 Authentication

Whilst many parts of the API are publicly accessible, some endpoints require authentication with an API access key, which is an alphanumeric string such as `VtF93tXp5IUZE307kPjijoGCUtBq4INmNTS4wlRG`. If provided, this key must be passed into every authenticated request, using the `key` argument.

To get an API key, visit your Flight Plan Database account settings page. Your account will need a verified email address to add an API key.

Endpoints that require authentication are marked as such in the API docs. Failing to provide valid authentication credentials on these endpoints will result in an *UnauthorizedException()* being raised. You are responsible for maintaining the security of your private API key, which gives near full access to your Flight Plan Database account. If your key is exposed, please use *flightplandb.api.revoke()* to revoke your key manually.

1.1.2 Quickstart

This document assumes you have the library installed; if not, check out the Installation section of the introduction.

1.1.2.1 Example

This is a small example program which demonstrates basic usage of the library. In this example, the only authenticated requests are those for which it is required. In most cases, all requests would be authenticated, since authentication raises the request limit from 100 to 2500.

```
import flightplandb as fpdb
import asyncio

# obviously, substitute your own token
API_KEY = "VtF93tXp5IUZE307kPjijoGCUtBq4INmNTS4wlRG"

async def main():
    # list all users named lemon
    async for user in fpdb.user.search(username="lemon"):
        print(user)

    # fetch most relevant user named lemon
    print(await fpdb.user.fetch(username="lemon"))

    # fetch first 20 of lemon's plans
    lemon_plans = fpdb.user.plans(username="lemon", limit=20)
    async for plan in lemon_plans:
        print(plan)

    # define a query to search for all plans
    query = fpdb.datatypes.PlanQuery(fromICAO="EHAM",
                                     toICAO="EGLL")
    # then search for the first three results of that query, sorted by distance
    # the route is included, which requires authentication
    resp = fpdb.plan.search(
        plan_query=query,
        include_route=True,
        sort="distance",
        limit=3,
        key=API_KEY
    )
    # and print each result in the response
    async for i in resp:
        print(i)

    # fetch the weather for Schiphol Airport
    print(await fpdb.weather.fetch("EHAM"))

    # then check remaining requests by subtracting the requests made from the total limit
    print((await fpdb.api.limit_cap())-(await fpdb.api.limit_used()))

asyncio.run(main())
```

Try saving this program in a file in your project directory and running it. Experiment around with different commands to get a feel for the library.

For specific commands, check out the *API reference*.

1.1.3 Changelog

1.1.3.1 0.8.1

This makes the library compatible with mypy strict checking, and slightly cleans up the release workflow.

1.1.3.2 0.8.0

This makes the entire library compatible with PEP-561, so that it can now be used with a static type checker like mypy. The codebase has been reformatted with black and isort, and the tags field of a PlanQuery now takes a list of strings, rather than a single string containing the tags separated by commas and spaces. `pdf` is now the only return format which returns bytes; `native` returns a dataclass and all other formats return a UTF-8 string.

The changelog has also been updated to include the changes of version 0.5.0 and earlier. A pre-commit file has been added to ensure all checks will pass before committing.

1.1.3.3 0.7.2

This fixes a bug in the core API interface where HTTP headers were being passed into requests as parameters.

1.1.3.4 0.7.1

This is a minor update, which adds support for Python 3.11 and moves the package configuration from setup.py to pyproject.toml. No breaking changes have been introduced. A bug has been fixed which was causing aiohttp to crash on null parameters, and a bug in the quickstart example has been fixed.

1.1.3.5 0.7.0

This is another complete rewrite of the library, in which it is entirely converted to async. This should mean faster execution of parallel requests, and no blocking when called from another async library. Support for Python 3.7 has been dropped in this release. Python 3.11 is not yet supported as aiohttp does not yet support Python 3.11 at the time of release.

1.1.3.6 0.6.0

This is a complete rewrite of the library, which moves functions out of classes. This does have the side effect of requiring a key to be passed into every authenticated request, instead of being passed into a class once on initialisation. The rewrite also incorporates several small bugfixes, and changes the test environment from unittest to pytest. Python 3.10 is now supported.

1.1.3.7 0.5.0

This adds support for the OM, MM, and IM navaid types, fixing issue #14. `include_route` is made into a function argument rather than a dataclass field, fixing issue #13. Parts of the code are also refactored to use keyword arguments instead of positional arguments to help reduce bugs.

1.1.3.8 0.4.1

This documents the return format options for plan fetching, and differentiates between a default `native` option where the returned json is unpacked into an appropriate dataclass and a `json` option where it is returned as json.

1.1.3.9 0.4.0

This adds a dark theme to the sphinx documentation. Exceeding of the API limit now raises a dedicated TooManyRequestsException. Additionally, all `Union[<type>, None]` type hints have been replaced by `Optional[<type>]`

1.1.3.10 0.3.2

This updates the documentation, and fixes some incorrect type hints.

1.1.3.11 0.3.1

This splits the codebase up into separate submodules. Custom exception classes have been written to handle different HTTP errors. Additionally, unit tests have been written, and Github workflows have been added to run tests and lint the codebase on a push and upload to pip on a version release.

1.1.3.12 0.3.0

This changes the wrapper file into an installable Python package, moves the documentation from the readme to a Sphinx project on readthedocs, and adds docstrings for all functions. All data is now handled via dataclasses. The project now uses semantic versioning.

1.1.3.13 0.2

This adds functions for all remaining API endpoints which have not yet been wrapped. Error handling has also been added, and the readme has been expanded.

1.1.3.14 0.1-alpha

This is the initial, incomplete release of this wrapper. Many functions are not yet implemented, and the wrapper is highly unstable.

1.2 API reference

This is a Python 3 wrapper for the Flight Plan Database API. Flight Plan Database is a website for creating and sharing flight plans for use in flight simulation. For more information on Flight Plan Database, read their excellent About page at https://flightplandatabase.com/about. For more information about this library, read the documentation at https://flightplandb-py.readthedocs.io/.

1.2.1 API

These functions return information about the API.

async flightplandb.api.**header_value**(*header_key: str*, *key: str | None = None*) → str

Gets header value for key. Do not call directly.

Parameters

- **header_key** (*str*) -- One of the HTTP header keys
- **key** (*str*, optional) -- API authentication key.

Returns
The value corresponding to the passed key

Return type
str

async flightplandb.api.**version**(*key: str | None = None*) → int

API version that returned the response.

Parameters
key (*str*, optional) -- API authentication key.

Returns
API version

Return type
int

async flightplandb.api.**units**(*key: str | None = None*) → str

The units system used for numeric values. https://flightplandatabase.com/dev/api#units

Parameters
key (*str*, optional) -- API authentication key.

Returns
AVIATION, METRIC or SI

Return type
str

async flightplandb.api.**limit_cap**(*key: str | None = None*) → int

The number of requests allowed per day, operated on an hourly rolling basis. i.e requests used between 19:00 and 20:00 will become available again at 19:00 the following day. API key authenticated requests get a higher daily rate limit and can be raised if a compelling use case is presented. See *Request Limits* for more details.

Parameters
key (*str*, optional) -- API authentication key.

Returns
number of allowed requests per day

Return type
int

`async` flightplandb.api.**limit_used**(*key: str | None = None*) → int

The number of requests used in the current period by the presented API key or IP address. See *Request Limits* for more details.

Parameters
key (*str*, optional) -- API authentication key.

Returns
number of requests used in period

Return type
int

`async` flightplandb.api.**ping**(*key: str | None = None*) → *StatusResponse*

Checks API status to see if it is up

Parameters
key (*str*, optional) -- API authentication key.

Returns
OK 200 means the service is up and running.

Return type
StatusResponse

`async` flightplandb.api.**revoke**(*key: str*) → *StatusResponse*

Revoke the API key in use in the event it is compromised. See *Authentication* for more details.

Requires authentication.

Parameters
key (*str*) -- API authentication key.

Returns
If the HTTP response code is 200 and the status message is "OK", then the key has been revoked and any further requests will be rejected. Any other status code or message indicates an error has occurred and the errors array will give further details.

Return type
StatusResponse

1.2.2 Nav

Commands related to navigation aids and airports.

`async` flightplandb.nav.**airport**(*icao: str*, *key: str | None = None*) → *Airport*

Fetches information about an airport.

Parameters

- **icao** (*str*) -- The airport ICAO to fetch information for
- **key** (*str*, optional) -- API authentication key.

Returns
Airport if the airport was found.

Return type
Union[*Airport*, None]

Raises
BadRequestException -- No airport with the specified ICAO code was found.

async flightplandb.nav.**nats**(*key: str | None = None*) → List[*Track*]

Fetches current North Atlantic Tracks.

Parameters
key (*str*, optional) -- API authentication key.

Returns
List of NATs

Return type
List[*Track*]

async flightplandb.nav.**pacots**(*key: str | None = None*) → List[*Track*]

Fetches current Pacific Organized Track System tracks.

Parameters
key (*str*, optional) -- API authentication key.

Returns
List of PACOTs

Return type
List[*Track*]

async flightplandb.nav.**search**(*query: str*, *type_: str | None = None*, *key: str | None = None*) → AsyncIterable[*SearchNavaid*]

Searches navaids using a query.

Parameters

- **query** (*str*) -- The search query. Searches the navaid identifier and navaid name
- **type_** (*str*, optional) -- Navaid type. Must be either `None` (default value, returns all types) or one of *validtypes*
- **key** (*str*, optional) -- API authentication key.

Yields
AsyncIterable[SearchNavaid] -- A iterable of navaids with either a name or ident matching the `query`

1.2.3 Plan

Table 1: Table of permitted plan return types

Plan format	Key
Native dataclass format	native
JSON formatted plan	json
XML formatted plan	xml
CSV formatted plan	csv
PDF formatted plan	pdf
Google Earth KML formatted plan	kml
X-Plane FMS (8, 9 & 10) formatted plan	xplane
X-Plane 11 formatted plan	xplane11
FS2004/FS9 formatted plan	fs9
FSX XML formatted plan	fsx
Squawkbox formatted plan	squawkbox
X-FMC formatted plan	xfmc
PMDG rte formatted plan	pmdg
Airbus X formatted plan	airbusx
QualityWings formatted plan	qualitywings
iFly 747 (.route) formatted plan	ifly747
FlightGear formatted plan (version 2 XML)	flightgear
TFDi Design 717 (version 1 XML)	tfdi717
Infinite Flight	infiniteflight

Flightplan-related commands.

async flightplandb.plan.**fetch**(*id_: int*, *return_format: Literal['native'] = 'native'*, *key: str | None = None*) → *Plan*

async flightplandb.plan.**fetch**(*id_: int*, *return_format: Literal['pdf']*, *key: str | None = None*) → bytes

async flightplandb.plan.**fetch**(*id_: int*, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *key: str | None = None*) → str

Fetches a flight plan and its associated attributes by ID. Returns it in specified format.

Parameters

- **id_** (*int*) -- The ID of the flight plan to fetch
- **return_format** (*str*) -- The API response format, defaults to `"native"`. Must be one of the keys in the *Table of permitted plan return types*.
- **key** (*str*, optional) -- API authentication key.

Returns

Plan of the specified plan if `"native"` is specified as the `return_format` (default).

`bytes` if PDF was specified as the `return_format`.

`str` if a different `return_format` was specified.

Return type
Union[*Plan*, Dict, bytes, str]

Raises
NotFoundException -- No plan with the specified id was found.

async flightplandb.plan.**create**(*plan:* Plan, *return_format: Literal['native'] = 'native'*, *key: str | None = None*) → *Plan*

async flightplandb.plan.**create**(*plan:* Plan, *return_format: Literal['pdf']*, *key: str | None = None*) → bytes

async flightplandb.plan.**create**(*plan:* Plan, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *key: str | None = None*) → str

Creates a new flight plan.

Requires authentication.

Parameters

- **plan** (Plan) -- The Plan object to register on the website
- **return_format** (*str*) -- The API response format, defaults to "native". Must be one of the keys in the *Table of permitted plan return types*.
- **key** (*str*, optional) -- API authentication key.

Returns

Plan of the registered plan created on Flight Plan Database if "native" is specified as the return_format (default).

bytes if PDF was specified as the return_format.

str if a different return_format was specified.

Return type
Union[*Plan*, Dict, bytes, str]

Raises
BadRequestException -- The plan submitted had incorrect arguments or was otherwise unusable.

async flightplandb.plan.**edit**(*plan:* Plan, *return_format: Literal['native'] = 'native'*, *key: str | None = None*) → *Plan*

async flightplandb.plan.**edit**(*plan:* Plan, *return_format: Literal['pdf']*, *key: str | None = None*) → bytes

async flightplandb.plan.**edit**(*plan:* Plan, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *key: str | None = None*) → str

Edits a flight plan linked to your account.

Requires authentication.

Parameters

- **plan** (Plan) -- The new Plan object to replace the old one associated with that ID
- **return_format** (*str*) -- The API response format, defaults to "native". Must be one of the keys in the *Table of permitted plan return types*.
- **key** (*str*, optional) -- API authentication key.

Returns

Plan of the registered flight plan created on flight plan database, corresponding to the route after being edited if "native" is specified as the return_format (default).

bytes if PDF was specified as the return_format.

str if a different return_format was specified.

Return type
Union[*Plan*, Dict, bytes, str]

Raises

- ***BadRequestException*** -- The plan submitted had incorrect arguments or was otherwise unusable.
- ***NotFoundException*** -- No plan with the specified id was found.

async flightplandb.plan.**delete**(*id_: int*, *key: str | None = None*) → *StatusResponse*

Deletes a flight plan that is linked to your account.

Requires authentication.

Parameters

- **id_** (*int*) -- The ID of the flight plan to delete
- **key** (*str*, optional) -- API authentication key.

Returns
OK 200 means a successful delete

Return type
StatusResponse

Raises
NotFoundException -- No plan with the specified id was found.

async flightplandb.plan.**search**(*plan_query:* PlanQuery, *sort: str = 'created'*, *include_route: bool | None = False*, *limit: int = 100*, *key: str | None = None*) → AsyncIterable[*Plan*]

Searches for flight plans. A number of search parameters are available. They will be combined to form a search request.

Requires authentication if route is included in results

Parameters

- **plan_query** (PlanQuery) -- A dataclass containing multiple options for plan searches
- **sort** (*str*) -- Sort order to return results in. Valid sort orders are created, updated, popularity, and distance
- **limit** (*int*) -- Maximum number of plans to return, defaults to 100
- **include_route** (*bool, optional*) -- Include route in response, defaults to False
- **key** (*str*, optional) -- API authentication key.

Yields
AsyncIterable[Plan] -- An iterable containing *Plan* objects.

async flightplandb.plan.**has_liked**(*id_: int*, *key: str | None = None*) → bool

Fetches your like status for a flight plan.

Requires authentication.

Parameters

- **id_** (*int*) -- ID of the flightplan to be checked
- **key** (*str*, optional) -- API authentication key.

Returns
`True`/`False` to indicate that the plan was liked / not liked

Return type
bool

async flightplandb.plan.**like**(*id_: int*, *key: str | None = None*) → *StatusResponse*

Likes a flight plan.

Requires authentication.

Parameters

- **id_** (*int*) -- ID of the flightplan to be liked
- **key** (*str*, optional) -- API authentication key.

Returns
`message=Created` means the plan was successfully liked. `message=OK` means the plan was already liked.

Return type
StatusResponse

Raises
NotFoundException -- No plan with the specified id was found.

async flightplandb.plan.**unlike**(*id_: int*, *key: str | None = None*) → bool

Removes a flight plan like.

Requires authentication.

Parameters

- **id_** (*int*) -- ID of the flightplan to be unliked
- **key** (*str*, optional) -- API authentication key.

Returns
`True` for a successful unlike

Return type
bool

Raises
NotFoundException -- No plan with the specified id was found, or the plan was found but wasn't liked.

async flightplandb.plan.**generate**(*gen_query:* GenerateQuery, *include_route: bool | None = False*, *key: str | None = None*) → *Plan*

Creates a new flight plan using the route generator.

Requires authentication.

Parameters

- **gen_query** (`GenerateQuery`) -- A dataclass with options for flight plan generation
- **include_route** (*bool, optional*) -- Include route in response, defaults to False
- **key** (*str*, optional) -- API authentication key.

Returns
The registered flight plan created on flight plan database, corresponding to the generated route

Return type
Plan

async flightplandb.plan.**decode**(*route: str*, *key: str | None = None*) → *Plan*

Creates a new flight plan using the route decoder.

Requires authentication.

Parameters

- **route** (*str*) -- The route to decode. Use a comma or space separated string of waypoints, beginning and ending with valid airport ICAOs (e.g. KSAN BROWS TRM LRAIN KDEN). Airways are supported if they are preceded and followed by valid waypoints on the airway (e.g. 06TRA UL851 BEGAR). SID and STAR procedures are not currently supported and will be skipped, along with any other unmatched waypoints.
- **key** (*str*, optional) -- API authentication key.

Returns
The registered flight plan created on flight plan database, corresponding to the decoded route

Return type
Plan

Raises
BadRequestException -- The encoded plan submitted had incorrect arguments or was otherwise unusable.

1.2.4 Tags

Contains the command for fetching flight plan tags.

async flightplandb.tags.**fetch**(*key: str | None = None*) → List[*Tag*]

Fetches current popular tags from all flight plans. Only tags with sufficient popularity are included.

Parameters
key (*str*, optional) -- API authentication key.

Returns
A list of the current popular tags.

Return type
List[*Tag*]

1.2.5 User

Commands related to registered users.

async flightplandb.user.**me**(*key: str | None = None*) → *User*

Fetches profile information for the currently authenticated user.

Requires authentication.

Parameters
key (*str*, optional) -- API authentication key.

Returns
The User object of the currently authenticated user

Return type
User

Raises
UnauthorizedException -- Authentication failed.

async flightplandb.user.**fetch**(*username: str*, *key: str | None = None*) → *User*

Fetches profile information for any registered user

Parameters

- **username** (*str*) -- Username of the registered User
- **key** (*str*, optional) -- API authentication key.

Returns
The User object of the user associated with the username

Return type
User

Raises
NotFoundException -- No user was found with this username.

async flightplandb.user.**plans**(*username: str*, *sort: str = 'created'*, *limit: int = 100*, *key: str | None = None*) → AsyncIterable[*Plan*]

Fetches flight plans created by a user.

Parameters

- **username** (*str*) -- Username of the user who created the flight plans
- **sort** (*str*) -- Sort order to return results in. Valid sort orders are created, updated, popularity, and distance
- **limit** (*int*) -- Maximum number of plans to fetch, defaults to `100`
- **key** (*str*, optional) -- API authentication key.

Yields
AsyncIterable[Plan] -- An iterator with all the flight plans a user created, limited by `limit`

async flightplandb.user.**likes**(*username: str*, *sort: str = 'created'*, *limit: int = 100*, *key: str | None = None*) → AsyncIterable[*Plan*]

Fetches flight plans liked by a user.

Parameters

- **username** (*str*) -- Username of the user who liked the flight plans
- **sort** (*str*) -- Sort order to return results in. Valid sort orders are created, updated, popularity, and distance
- **limit** (*int*) -- Maximum number of plans to fetch, defaults to `100`
- **key** (*str*, optional) -- API authentication key.

Yields
AsyncIterable[Plan] -- An iterable with all the flight plans a user liked, limited by `limit`

async flightplandb.user.**search**(*username: str*, *limit: int = 100*, *key: str | None = None*) → AsyncIterable[*UserSmall*]

Searches for users by username. For more detailed info on a specific user, use *fetch()*

Parameters

- **username** (*str*) -- Username to search user database for

- **limit** (*type*) -- Maximum number of users to fetch, defaults to `100`
- **key** (*str*, optional) -- API authentication key.

Yields
AsyncIterable[UserSmall] -- An iterable with a list of users approximately matching `username`, limited by `limit`. UserSmall is used instead of User, because less info is returned.

1.2.6 Weather

Weather. I mean, how much is there to say?

async flightplandb.weather.**fetch**(*icao: str*, *key: str | None = None*) → *Weather*

Fetches current weather conditions at an airport

Parameters

- **icao** (*str*) -- ICAO code of the airport for which the weather will be fetched
- **key** (*str*, optional) -- API authentication key.

Returns
METAR and TAF for an airport

Return type
Weather

Raises
NotFoundException -- No airport with the specified ICAO code was found.

1.2.7 Datatypes

class flightplandb.datatypes.**StatusResponse**(*message: str*, *errors: List[str] | None*)

Returned for some functions to indicate execution status

message

The message associated with the status returned

Type
str

errors

A list of any errors raised

Type
Optional[List[str]]

class flightplandb.datatypes.**User**(*id: int*, *username: str*, *location: str | None = None*, *gravatarHash: str | None = None*, *joined: datetime | None = None*, *lastSeen: datetime | None = None*, *plansCount: int | None = 0*, *plansDistance: float | None = 0.0*, *plansDownloads: int | None = 0*, *plansLikes: int | None = 0*)

Describes users registered on the website

id

Unique user identifier number

Type
int

`username`

Username

Type
str

`location`

User provided location information. `None` if not available

Type
Optional[str]

`gravatarHash`

Gravatar hash based on user's account email address.

Type
Optional[str]

`joined`

UTC Date and time of user registration

Type
Optional[datetime] = None

`lastSeen`

UTC Date and time the user was last connected

Type
Optional[datetime] = None

`plansCount`

Number of flight plans created by the user

Type
Optional[int]

`plansDistance`

Total distance of all user's flight plans

Type
Optional[float]

`plansDownloads`

Total download count of all user's plans

Type
Optional[int]

`plansLikes`

Total like count of all user's plans

Type
Optional[int]

`class flightplandb.datatypes.UserSmall(id: int, username: str, location: str | None = None, gravatarHash: str | None = None)`

Describes users registered on the website, with far less info

`id`

Unique user identifier number

Type
int

`username`

Username

Type
str

`location`

User provided location information. `None` if not available

Type
Optional[str]

`gravatarHash`

Gravatar hash based on user's account email address.

Type
Optional[str]

`class flightplandb.datatypes.Application(id: int, name: str | None = None, url: str | None = None)`

Describes application associated with a flight plan

`id`

Unique application identifier number

Type
int

`name`

Application name

Type
Optional[str]

`url`

Application URL

Type
Optional[str]

`class flightplandb.datatypes.Via(ident: str, type: str)`

Describes routes to *RouteNode* s

`ident`

desc

Type
str

`type`

Type of Via; must be one of *Via.validtypes*

Type
str

validtypes

Do not change. Valid Via types.

Type
List[str]

class flightplandb.datatypes.**RouteNode**(*ident: str*, *type: str*, *lat: float*, *lon: float*, *id: int | None = None*, *alt: float | None = None*, *name: str | None = None*, *via:* Via | *Dict[str, Any] | None = None*)

Describes nodes in *Route* s

id

For some obscure reason an apparently useless id is included with each node when the node is inside a *Track* route. Goodness knows why.

Type
Optional[int]

ident

Node navaid identifier

Type
str

type

Type of RouteNode; must be one of *RouteNode.validtypes*

Type
str

lat

Node latitude in decimal degrees

Type
float

lon

Node longitude in decimal degrees

Type
float

alt

Suggested altitude at node

Type
Optional[float]

name

Node name.

Type
Optional[str]

via

Route to node.

Type
Optional[*Via*]

validtypes

Do not change. Valid RouteNode types.

Type
List[str]

class flightplandb.datatypes.**Route**(*nodes: List[RouteNode]*, *eastLevels: List[str] | None = None*, *westLevels: List[str] | None = None*)

Describes the route of a *Plan*

nodes

A list of *RouteNode* s. A route must have at least 2 nodes.

Type
List[*RouteNode*]

eastLevels

Valid eastbound flightlevels. Only used inside a NATS *Track*.

Type
Optional[List[str]]

westLevels

Valid westbound flightlevels. Only used inside a NATS *Track*.

Type
Optional[List[str]]

class flightplandb.datatypes.**Cycle**(*id: int*, *ident: str*, *year: int*, *release: int*)

Navdata cycle

id

FlightPlanDB cycle id

Type
int

ident

AIP-style cycle id

Type
str

year

Last two digits of cycle year

Type
int

release

Cycle release

Type
int

class flightplandb.datatypes.**Plan**(*fromICAO: str | None*, *toICAO: str | None*, *fromName: str | None*, *toName: str | None*, *id: int | None = None*, *flightNumber: str | None = None*, *distance: float | None = None*, *maxAltitude: float | None = None*, *waypoints: int | None = None*, *likes: int | None = None*, *downloads: int | None = None*, *popularity: int | None = None*, *notes: str | None = None*, *encodedPolyline: str | None = None*, *createdAt: datetime | str | None = None*, *updatedAt: datetime | str | None = None*, *tags: List[str] | None = None*, *user:* User *| None = None*, *application:* Application *| None = None*, *route:* Route *| None = None*, *cycle:* Cycle *| None = None*)

A flight plan; the thing this whole API revolves around

id

Unique plan identifier number

Type
int

fromICAO

ICAO code of the departure airport

Type
Optional[str]

toICAO

ICAO code of the destination airport

Type
Optional[str]

fromName

Name of the departure airport

Type
Optional[str]

toName

Name of the destination airport

Type
Optional[str]

flightNumber

Flight number of the flight plan

Type
Optional[str]

distance

Total distance of the flight plan route

Type
Optional[float]

maxAltitude

Maximum altitude of the flight plan route

Type
Optional[float]

`waypoints`

Number of nodes in the flight plan route

Type
Optional[int]

`likes`

Number of times the flight plan has been liked

Type
Optional[int]

`downloads`

Number of times the flight plan has been downloaded

Type
Optional[int]

`popularity`

Relative popularity of the plan based on downloads and likes

Type
Optional[int]

`notes`

Extra information about the flight plan

Type
Optional[str]

`encodedPolyline`

Encoded polyline of route, which can be used for quickly drawing maps

Type
Optional[str]

`createdAt`

UTC Date and time of flight plan creation

Type
Optional[datetime]

`updatedAt`

UTC Date and time of the last flight plan edit

Type
Optional[datetime]

`tags`

List of flight plan tags

Type
Optional[List[str]]

`user`

User associated with the item. `None` if no user linked

Type
Optional[*User*]

`application`

Application associated with the item. `None` if no application linked

Type
Optional[*Application*]

`route`

The flight plan route

Type
Optional[*Route*]

`cycle`

Navigation data cycle associated with the item. `None` if no cycle linked

Type
Optional[*Cycle*]

class flightplandb.datatypes.**PlanQuery**(*q: str | None = None*, *From: str | None = None*, *to: str | None = None*, *fromICAO: str | None = None*, *toICAO: str | None = None*, *fromName: str | None = None*, *toName: str | None = None*, *flightNumber: str | None = None*, *distanceMin: str | None = None*, *distanceMax: str | None = None*, *tags: List[str] | None = None*, *includeRoute: bool | None = None*)

Simple search query.

`q`

Username, tags and the flight number

Type
Optional[str]

`From`

From search query. Search departure ICAO & name

Type
Optional[str]

`to`

To search query. Search departure ICAO & name

Type
Optional[str]

`fromICAO`

Matches departure airport ICAO

Type
Optional[str]

`toICAO`

Matches destination airport ICAO

Type
Optional[str]

`fromName`

Matches departure airport name

Type
Optional[str]

toName

Matches destination airport name

Type
Optional[str]

flightNumber

Matches flight number

Type
Optional[str]

distanceMin

Minimum route distance

Type
Optional[str]

distanceMax

Maximum route distance, with units determined by the X-Units header

Type
Optional[str]

tags

List of tag names to search

Type
Optional[List[str]]

class flightplandb.datatypes.**GenerateQuery**(*fromICAO: str*, *toICAO: str*, *useNAT: bool | None = True*, *usePACOT: bool | None = True*, *useAWYLO: bool | None = True*, *useAWYHI: bool | None = True*, *cruiseAlt: float | None = 35000*, *cruiseSpeed: float | None = 420*, *ascentRate: float | None = 2500*, *ascentSpeed: float | None = 250*, *descentRate: float | None = 1500*, *descentSpeed: float | None = 250*)

Generate plan query.

fromICAO

The departure airport ICAO code

Type
str

toICAO

The destination airport ICAO code

Type
str

useNAT

Use Pacific Organized Track System tracks in the route generation

Type
Optional[bool]

usePACOT

Use Pacific Organized Track System tracks in the route generation

Type
Optional[bool]

`useAWYLO`
Use low-level airways in the route generation

Type
Optional[bool]

`useAWYHI`
Use high-level airways in the route generation

Type
Optional[bool]

`cruiseAlt`
Basic flight profile cruise altitude (altitude)

Type
Optional[float]

`cruiseSpeed`
Basic flight profile cruise speed (speed)

Type
Optional[float]

`ascentRate`
Basic flight profile ascent rate (climb rate)

Type
Optional[float]

`ascentSpeed`
Basic flight profile ascent speed (speed)

Type
Optional[float]

`descentRate`
Basic flight profile descent rate (climb rate)

Type
Optional[float]

`descentSpeed`
Basic flight profile descent speed (speed)

Type
Optional[float]

`class flightplandb.datatypes.Tag(`*name: str*, *description: str | None*, *planCount: int*, *popularity: int*`)`
Flight plan tag

`name`
Tag name

Type
str

`description`
Description of the tag. `None` if no description is available

Type
Optional[str]

planCount

Number of plans with this tag

Type
int

popularity

Popularity index of the tag

Type
int

class flightplandb.datatypes.**Timezone**(*name: str | None*, *offset: float | None*)

Contains timezone information

name

The IANA timezone the airport is located in. None if not available

Type
Optional[str]

offset

The number of seconds the airport timezone is currently offset from UTC. Positive is ahead of UTC. None if not available

Type
Optional[float]

class flightplandb.datatypes.**Times**(*sunrise: datetime | str*, *sunset: datetime | str*, *dawn: datetime | str*, *dusk: datetime | str*)

Contains relevant times information

sunrise

Time of sunrise

Type
datetime

sunset

Time of sunset

Type
datetime

dawn

Time of dawn

Type
datetime

dusk

Time of dusk

Type
datetime

class flightplandb.datatypes.**RunwayEnds**(*ident: str*, *lat: float*, *lon: float*)

Ends of *Runway* . No duh.

`ident`

The identifier of the runway end

Type
str

`lat`

The latitude of the runway end

Type
float

`lon`

The longitude of the runway end

Type
float

`class flightplandb.datatypes.Navaid(ident: str, type: str, lat: float, lon: float, airport: str, runway: str, frequency: float | None, slope: float | None, bearing: float | None, name: str | None, elevation: float, range: float)`

Describes a navigational aid

`ident`

The navaid identifier

Type
str

`type`

The navaid type. Must be one of *Navaid.validtypes*

Type
str

`lat`

The navaid latitude

Type
float

`lon`

The navaid longitude

Type
float

`airport`

The airport associated with the navaid

Type
str

`runway`

The runway associated with the navaid

Type
str

`frequency`

The navaid frequency in Hz. `None` if not available

Type
Optional[float]

`slope`

The navaid slope in degrees from horizontal used for type GS

Type
Optional[float]

`bearing`

The navaid bearing in true degrees. `None` if not available

Type
Optional[float]

`name`

The navaid name. `None` if not available

Type
Optional[str]

`elevation`

The navaid elevation above mean sea level (elevation)

Type
float

`range`

The navaid range; units determined by the X-Units header (distance)

Type
float

`validtypes`

Do not change. Valid Navaid types.

Type
List[str]

`class flightplandb.datatypes.Runway`(*ident: str*, *width: float*, *length: float*, *bearing: float*, *surface: str*, *markings: List[str]*, *lighting: List[str]*, *thresholdOffset: float*, *overrunLength: float*, *ends: List[*RunwayEnds*]*, *navaids: List[*Navaid*]*)

Describes a runway at an *Airport*

`ident`

The runway identifier

Type
str

`width`

The runway width, with units determined by the X-Units header (length)

Type
float

`length`
The runway length, with units determined by the X-Units header (length)

Type
float

`bearing`
The runway bearing in true degrees

Type
float

`surface`
The runway surface material

Type
str

`markings`
List of strings of runway markings

Type
List[str]

`lighting`
List of strings of runway lighting types

Type
List[str]

`thresholdOffset`
The distance of the displaced threshold from the runway end (length)

Type
float

`overrunLength`
The runway overrun length, with units determined by the X-Units header

Type
float

`ends`
Two element List containing the location of the two ends of the runway

Type
List[*RunwayEnds*]

`navaids`
List of navaids associated with the current runway

Type
List[*Navaid*]

`class flightplandb.datatypes.Frequency(type: str, frequency: float, name: str | None)`
Holds frequency information

`type`
The frequency type

Type
str

frequency

The frequency in Hz

Type
float

name

The frequency name. `None` if not available

Type
Optional[str]

class flightplandb.datatypes.**Weather**(*METAR: str | None*, *TAF: str | None*)

Contains weather reports and predictions

METAR

Current METAR report for the airport

Type
Optional[str]

TAF

Current TAF report for the airport

Type
Optional[str]

class flightplandb.datatypes.**Airport**(*ICAO: str*, *IATA: str | None*, *name: str*, *regionName: str | None*, *elevation: float*, *lat: float*, *lon: float*, *magneticVariation: float*, *timezone:* Timezone, *times:* Times, *runwayCount: int*, *runways: List[*Runway*]*, *frequencies: List[*Frequency*]*, *weather:* Weather)

Describes an airport. An oversized dataclass with more information than you'd need in 500 years.

ICAO

The airport ICAO code

Type
str

IATA

The airport IATA code. `None` if not available

Type
Optional[str]

name

The airport name

Type
str

regionName

The geographical region the airport is located in. `None` if not available

Type
Optional[str]

elevation

The airport elevation above mean sea level (elevation)

Type
float

`lat`

The airport latitude in degrees

Type
float

`lon`

The airport longitude in degrees

Type
float

`magneticVariation`

The current magnetic variation/declination at the airport, based on the World Magnetic Model

Type
float

`timezone`

The airport timezone information

Type
Timezone

`times`

Relevant times at the airport

Type
Times

`runwayCount`

The number of runways at the airport

Type
int

`runways`

List of runways. Note: each physical runway will appear twice, once from each end

Type
List[*Runway*]

`frequencies`

List of frequencies associated with the airport

Type
List[*Frequency*]

`weather`

Airport weather information

Type
Weather

`class flightplandb.datatypes.Track(ident: str | int, route: Route, validFrom: datetime, validTo: datetime)`

Used for NATS and PACOTS tracks

ident

Track identifier; str in NATS, int in PACOTS

Type
Union[str, int]

route

Route of the track

Type
Route

validFrom

UTC datetime the track is valid from

Type
datetime

validTo

UTC datetime the track is valid to

Type
datetime

class flightplandb.datatypes.**SearchNavaid**(*ident: str*, *type: str*, *lat: float*, *lon: float*, *elevation: float*, *runwayIdent: str | None = None*, *airportICAO: str | None = None*, *name: float | None = None*)

Describes a navigational aid, as returned by the search function

ident

The navaid identifier

Type
str

type

The navaid type. Must be one of *SearchNavaid.validtypes*

Type
str

lat

The navaid latitude

Type
float

lon

The navaid longitude

Type
float

elevation

The navaid elevation above mean sea level (elevation)

Type
float

runwayIdent

The runway associated with the navaid. None if not available

Type
Optional[str]

airportICAO

The ICAO of the airport associated with the navaid. None if not available

Type
Optional[str]

name

The navaid name. None if not available

Type
Optional[float]

validtypes

Do not change. Valid SearchNavaid types

Type
List[str]

1.2.8 Exceptions

Contains all the internally defined exceptions used by the library.

exception flightplandb.exceptions.**BaseErrorHandler**(*status_code: int*, *message: str*)

Base exception. The other exceptions all inherit from this one, but this exception will be raised directly if no others match the returned HTTP status code.

status_code

Status code of the error

message

Description of the error

exception flightplandb.exceptions.**BadRequestException**(*status_code: int*, *message: str*)

An incorrect request was made to the server. Raised for an HTTP status code 400.

status_code

Status code of the error

message

A verbose description of this error.

exception flightplandb.exceptions.**UnauthorizedException**(*status_code: int*, *message: str*)

You are incorrectly authorised and may not make this request. Raised for an HTTP status code 401.

status_code

Status code of the error

message

A verbose description of this error.

exception flightplandb.exceptions.**ForbiddenException**(*status_code: int*, *message: str*)

The server refuses to fulfill this request, for instance due to insufficient authentication. Raised for an HTTP status code 403.

status_code

Status code of the error

message

A verbose description of this error.

exception flightplandb.exceptions.**NotFoundException**(*status_code: int*, *message: str*)

The server couldn't find a resource matching the request. Raised for an HTTP status code 404.

status_code

Status code of the error

message

A verbose description of this error.

exception flightplandb.exceptions.**TooManyRequestsException**(*status_code: int*, *message: str*)

Your requests limit for the server has been exceeded. Raised for an HTTP status code 429.

status_code

Status code of the error

message

A verbose description of this error.

exception flightplandb.exceptions.**InternalServerException**(*status_code: int*, *message: str*)

Something unspecified went wrong with the server. Raised for an HTTP status code 500.

status_code

Status code of the error

message

A verbose description of this error.

flightplandb.exceptions.**status_handler**(*status_code: int*, *ignore_statuses: Tuple[int] | Tuple = ()*) → None

Raises correct custom exception for appropriate HTTP status code.

1.2.9 Internal functions

This file mostly contains internal functions called by the API, so you're unlikely to ever use them.

async flightplandb.internal.**request**(*method: str*, *path: str*, *return_format: Literal['native'] = 'native'*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → Tuple[CIMultiDictProxy[str], Dict[str, Any]]

async flightplandb.internal.**request**(*method: str*, *path: str*, *return_format: Literal['pdf']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → Tuple[CIMultiDictProxy[str], bytes]

async flightplandb.internal.**request**(*method: str*, *path: str*, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → Tuple[CIMultiDictProxy[str], str]

General HTTP requests function for non-paginated results.

Parameters

- **method** (*str*) -- An HTTP request type. One of GET, POST, PATCH, or DELETE
- **path** (*str*) -- The endpoint's path to which the request is being made
- **return_format** (*str*, optional) -- The API response format, defaults to "native"
- **ignore_statuses** (*Tuple*, optional) -- Statuses (together with 200 OK) which don't raise an HTTPError, defaults to None
- **params** (*Dict*, optional) -- Any other HTTP request parameters, defaults to None
- **json_data** (*Dict*, optional) -- Custom JSON data to be formatted into the request body
- **key** (*str*) -- API token, defaults to None (which makes it unauthenticated)

Returns

A tuple of:

1. A dict of the response headers, but the keys are case-insensitive

2. A Dict if return_format is "native", otherwise str or bytes depending on if the return format is UTF-8 or something else.

Return type
Tuple[CIMultiDict, Union[Dict, str, bytes]]

Raises

- **ValueError** -- Invalid return_format option
- **HTTPError** -- Invalid HTTP status in response

async flightplandb.internal.**get_headers**(*key: str | None = None*) → CIMultiDictProxy[str]

Calls *request()* for request headers.

Parameters
key (*str*, optional) -- API token, defaults to None (which makes it unauthenticated)

Returns
A dict of the response headers, but the keys are case-insensitive.

Return type
CIMultiDictProxy

async flightplandb.internal.**get**(*path: str*, *return_format: Literal['native'] = 'native'*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *key: str | None = None*) → Dict[str, Any] | List[Any]

async flightplandb.internal.**get**(*path: str*, *return_format: Literal['pdf']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *key: str | None = None*) → bytes

async flightplandb.internal.**get**(*path: str*, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *key: str | None = None*) → str

Calls *request()* for get requests.

Parameters

- **path** (*str*) -- The endpoint's path to which the request is being made
- **return_format** (*str*, optional) -- The API response format, defaults to "native"
- **ignore_statuses** (*Tuple*, optional) -- Statuses (together with 200 OK) which don't raise an HTTPError, defaults to None
- **params** (*Dict*, optional) -- Any other HTTP request parameters, defaults to None
- **key** (*str*, optional) -- API token, defaults to None (which makes it unauthenticated)

Returns
A Dict if return_format is "native", otherwise str or bytes depending on if the return format is UTF-8 or something else.

Return type
Union[Dict, bytes]

async flightplandb.internal.**post**(*path: str*, *return_format: Literal['native'] = 'native'*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → Dict[str, Any]

async flightplandb.internal.**post**(*path: str*, *return_format: Literal['pdf']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → bytes

async flightplandb.internal.**post**(*path: str*, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → str

Calls *request()* for post requests.

Parameters

- **path** (*str*) -- The endpoint's path to which the request is being made
- **return_format** (*str*, optional) -- The API response format, defaults to "native"
- **ignore_statuses** (*Tuple*, optional) -- Statuses (together with 200 OK) which don't raise an HTTPError, defaults to None
- **params** (*Dict*, optional) -- Any other HTTP request parameters, defaults to None
- **json_data** (*Dict*, optional) -- Custom JSON data to be formatted into the request body
- **key** (*str*, optional) -- API token, defaults to None (which makes it unauthenticated)

Returns
A Dict if return_format is "native", otherwise str or bytes depending on if the return format is UTF-8 or something else.

Return type
Union[Dict, bytes]

async flightplandb.internal.**patch**(*path: str*, *return_format: Literal['native'] = 'native'*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → Dict[str, Any]

async flightplandb.internal.**patch**(*path: str*, *return_format: Literal['pdf']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → bytes

async flightplandb.internal.**patch**(*path: str*, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *json_data: Dict[str, Any] | None = None*, *key: str | None = None*) → str

Calls *request()* for patch requests.

Parameters

- **path** (*str*) -- The endpoint's path to which the request is being made
- **return_format** (*str*, optional) -- The API response format, defaults to "native"
- **ignore_statuses** (*Tuple*, optional) -- Statuses (together with 200 OK) which don't raise an HTTPError, defaults to None
- **params** (*Dict*, optional) -- Any other HTTP request parameters, defaults to None
- **json_data** (*Dict*, optional) -- Custom JSON data to be formatted into the request body
- **key** (*str*, optional) -- API token, defaults to None (which makes it unauthenticated)

Returns

A Dict if return_format is "native", otherwise str or bytes depending on if the return format is UTF-8 or something else.

Return type

Union[Dict, bytes]

async flightplandb.internal.**delete**(*path: str*, *return_format: Literal['native'] = 'native'*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *key: str | None = None*) → Dict[str, Any]

async flightplandb.internal.**delete**(*path: str*, *return_format: Literal['pdf']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *key: str | None = None*) → bytes

async flightplandb.internal.**delete**(*path: str*, *return_format: Literal['json', 'xml', 'csv', 'kml', 'xplane', 'xplane11', 'fs9', 'fsx', 'squawkbox', 'xfmc', 'pmdg', 'airbusx', 'qualitywings', 'ifly747', 'flightgear', 'tfdi717', 'infiniteflight']*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *key: str | None = None*) → str

Calls *request()* for delete requests.

Parameters

- **path** (*str*) -- The endpoint's path to which the request is being made
- **return_format** (*str*, optional) -- The API response format, defaults to "native"
- **ignore_statuses** (*Tuple*, optional) -- Statuses (together with 200 OK) which don't raise an HTTPError, defaults to None
- **params** (*Dict*, optional) -- Any other HTTP request parameters, defaults to None

- **key** (*str*, optional) -- API token, defaults to None (which makes it unauthenticated)

Returns
A `Dict` if `return_format` is `"native"`, otherwise `str` or `bytes` depending on if the return format is UTF-8 or something else.

Return type
Union[Dict, bytes]

async flightplandb.internal.**getiter**(*path: str*, *limit: int = 100*, *sort: str = 'created'*, *ignore_statuses: Tuple[int] | Tuple = ()*, *params: Dict[str, Any] | None = None*, *key: str | None = None*) → AsyncIterable[Dict[str, Any]]

Get *request()* for paginated results.

Parameters

- **path** (*str*) -- The endpoint's path to which the request is being made
- **limit** (*int, optional*) -- Maximum number of results to return, defaults to 100
- **sort** (*str*, optional) -- Sort order to return results in. Valid sort orders are created, updated, popularity, and distance
- **ignore_statuses** (*Tuple*, optional) -- Statuses (together with 200 OK) which don't raise an HTTPError, defaults to None
- **params** (*Dict*, optional) -- Any other HTTP request parameters, defaults to None
- **key** (*str*, optional) -- API token, defaults to None (which makes it unauthenticated)

Returns
An iterable of dicts. Return format cannot be specified.

Return type
AsyncIterable[Dict]

CHAPTER

TWO

INDICES AND TABLES

- genindex

PYTHON MODULE INDEX

f

flightplandb, 6
flightplandb.api, 6
flightplandb.datatypes, 15
flightplandb.exceptions, 32
flightplandb.internal, 33
flightplandb.nav, 7
flightplandb.plan, 9
flightplandb.tags, 13
flightplandb.user, 13
flightplandb.weather, 15

INDEX

A

Airport (*class in flightplandb.datatypes*), 29
airport (*flightplandb.datatypes.Navaid attribute*), 26
airport() (*in module flightplandb.nav*), 7
airportICAO (*flightplandb.datatypes.SearchNavaid attribute*), 32
alt (*flightplandb.datatypes.RouteNode attribute*), 18
Application (*class in flightplandb.datatypes*), 17
application (*flightplandb.datatypes.Plan attribute*), 21
ascentRate (*flightplandb.datatypes.GenerateQuery attribute*), 24
ascentSpeed (*flightplandb.datatypes.GenerateQuery attribute*), 24

B

BadRequestException, 32
BaseErrorHandler, 32
bearing (*flightplandb.datatypes.Navaid attribute*), 27
bearing (*flightplandb.datatypes.Runway attribute*), 28

C

create() (*in module flightplandb.plan*), 9
createdAt (*flightplandb.datatypes.Plan attribute*), 21
cruiseAlt (*flightplandb.datatypes.GenerateQuery attribute*), 24
cruiseSpeed (*flightplandb.datatypes.GenerateQuery attribute*), 24
Cycle (*class in flightplandb.datatypes*), 19
cycle (*flightplandb.datatypes.Plan attribute*), 22

D

dawn (*flightplandb.datatypes.Times attribute*), 25
decode() (*in module flightplandb.plan*), 12
delete() (*in module flightplandb.internal*), 36
delete() (*in module flightplandb.plan*), 11
descentRate (*flightplandb.datatypes.GenerateQuery attribute*), 24
descentSpeed (*flightplandb.datatypes.GenerateQuery attribute*), 24
description (*flightplandb.datatypes.Tag attribute*), 24
distance (*flightplandb.datatypes.Plan attribute*), 20
distanceMax (*flightplandb.datatypes.PlanQuery attribute*), 23
distanceMin (*flightplandb.datatypes.PlanQuery attribute*), 23
downloads (*flightplandb.datatypes.Plan attribute*), 21
dusk (*flightplandb.datatypes.Times attribute*), 25

E

eastLevels (*flightplandb.datatypes.Route attribute*), 19
edit() (*in module flightplandb.plan*), 10
elevation (*flightplandb.datatypes.Airport attribute*), 29
elevation (*flightplandb.datatypes.Navaid attribute*), 27
elevation (*flightplandb.datatypes.SearchNavaid attribute*), 31
encodedPolyline (*flightplandb.datatypes.Plan attribute*), 21
ends (*flightplandb.datatypes.Runway attribute*), 28
errors (*flightplandb.datatypes.StatusResponse attribute*), 15

F

fetch() (*in module flightplandb.plan*), 9
fetch() (*in module flightplandb.tags*), 13
fetch() (*in module flightplandb.user*), 14
fetch() (*in module flightplandb.weather*), 15
flightNumber (*flightplandb.datatypes.Plan attribute*), 20
flightNumber (*flightplandb.datatypes.PlanQuery attribute*), 23
flightplandb
    module, 6
flightplandb.api
    module, 6
flightplandb.datatypes
    module, 15
flightplandb.exceptions
    module, 32
flightplandb.internal
    module, 33
flightplandb.nav
    module, 7
flightplandb.plan

module, 9
flightplandb.tags
module, 13
flightplandb.user
module, 13
flightplandb.weather
module, 15
ForbiddenException, 32
frequencies (*flightplandb.datatypes.Airport attribute*), 30
Frequency (*class in flightplandb.datatypes*), 28
frequency (*flightplandb.datatypes.Frequency attribute*), 28
frequency (*flightplandb.datatypes.Navaid attribute*), 26
From (*flightplandb.datatypes.PlanQuery attribute*), 22
fromICAO (*flightplandb.datatypes.GenerateQuery attribute*), 23
fromICAO (*flightplandb.datatypes.Plan attribute*), 20
fromICAO (*flightplandb.datatypes.PlanQuery attribute*), 22
fromName (*flightplandb.datatypes.Plan attribute*), 20
fromName (*flightplandb.datatypes.PlanQuery attribute*), 22

G

generate() (*in module flightplandb.plan*), 12
GenerateQuery (*class in flightplandb.datatypes*), 23
get() (*in module flightplandb.internal*), 34
get_headers() (*in module flightplandb.internal*), 34
getiter() (*in module flightplandb.internal*), 37
gravatarHash (*flightplandb.datatypes.User attribute*), 16
gravatarHash (*flightplandb.datatypes.UserSmall attribute*), 17

H

has_liked() (*in module flightplandb.plan*), 11
header_value() (*in module flightplandb.api*), 6

I

IATA (*flightplandb.datatypes.Airport attribute*), 29
ICAO (*flightplandb.datatypes.Airport attribute*), 29
id (*flightplandb.datatypes.Application attribute*), 17
id (*flightplandb.datatypes.Cycle attribute*), 19
id (*flightplandb.datatypes.Plan attribute*), 20
id (*flightplandb.datatypes.RouteNode attribute*), 18
id (*flightplandb.datatypes.User attribute*), 15
id (*flightplandb.datatypes.UserSmall attribute*), 16
ident (*flightplandb.datatypes.Cycle attribute*), 19
ident (*flightplandb.datatypes.Navaid attribute*), 26
ident (*flightplandb.datatypes.RouteNode attribute*), 18
ident (*flightplandb.datatypes.Runway attribute*), 27
ident (*flightplandb.datatypes.RunwayEnds attribute*), 25
ident (*flightplandb.datatypes.SearchNavaid attribute*), 31
ident (*flightplandb.datatypes.Track attribute*), 30
ident (*flightplandb.datatypes.Via attribute*), 17
InternalServerException, 33

J

joined (*flightplandb.datatypes.User attribute*), 16

L

lastSeen (*flightplandb.datatypes.User attribute*), 16
lat (*flightplandb.datatypes.Airport attribute*), 30
lat (*flightplandb.datatypes.Navaid attribute*), 26
lat (*flightplandb.datatypes.RouteNode attribute*), 18
lat (*flightplandb.datatypes.RunwayEnds attribute*), 26
lat (*flightplandb.datatypes.SearchNavaid attribute*), 31
length (*flightplandb.datatypes.Runway attribute*), 27
lighting (*flightplandb.datatypes.Runway attribute*), 28
like() (*in module flightplandb.plan*), 12
likes (*flightplandb.datatypes.Plan attribute*), 21
likes() (*in module flightplandb.user*), 14
limit_cap() (*in module flightplandb.api*), 6
limit_used() (*in module flightplandb.api*), 7
location (*flightplandb.datatypes.User attribute*), 16
location (*flightplandb.datatypes.UserSmall attribute*), 17
lon (*flightplandb.datatypes.Airport attribute*), 30
lon (*flightplandb.datatypes.Navaid attribute*), 26
lon (*flightplandb.datatypes.RouteNode attribute*), 18
lon (*flightplandb.datatypes.RunwayEnds attribute*), 26
lon (*flightplandb.datatypes.SearchNavaid attribute*), 31

M

magneticVariation (*flightplandb.datatypes.Airport attribute*), 30
markings (*flightplandb.datatypes.Runway attribute*), 28
maxAltitude (*flightplandb.datatypes.Plan attribute*), 20
me() (*in module flightplandb.user*), 13
message (*flightplandb.datatypes.StatusResponse attribute*), 15
message (*flightplandb.exceptions.BadRequestException attribute*), 32
message (*flightplandb.exceptions.BaseErrorHandler attribute*), 32
message (*flightplandb.exceptions.ForbiddenException attribute*), 33
message (*flightplandb.exceptions.InternalServerException attribute*), 33
message (*flightplandb.exceptions.NotFoundException attribute*), 33
message (*flightplandb.exceptions.TooManyRequestsException attribute*), 33
message (*flightplandb.exceptions.UnauthorizedException attribute*), 32

METAR (*flightplandb.datatypes.Weather attribute*), 29
module
    flightplandb, 6
    flightplandb.api, 6
    flightplandb.datatypes, 15
    flightplandb.exceptions, 32
    flightplandb.internal, 33
    flightplandb.nav, 7
    flightplandb.plan, 9
    flightplandb.tags, 13
    flightplandb.user, 13
    flightplandb.weather, 15

N

name (*flightplandb.datatypes.Airport attribute*), 29
name (*flightplandb.datatypes.Application attribute*), 17
name (*flightplandb.datatypes.Frequency attribute*), 29
name (*flightplandb.datatypes.Navaid attribute*), 27
name (*flightplandb.datatypes.RouteNode attribute*), 18
name (*flightplandb.datatypes.SearchNavaid attribute*), 32
name (*flightplandb.datatypes.Tag attribute*), 24
name (*flightplandb.datatypes.Timezone attribute*), 25
nats() (*in module flightplandb.nav*), 8
Navaid (*class in flightplandb.datatypes*), 26
navaids (*flightplandb.datatypes.Runway attribute*), 28
nodes (*flightplandb.datatypes.Route attribute*), 19
notes (*flightplandb.datatypes.Plan attribute*), 21
NotFoundException, 33

O

offset (*flightplandb.datatypes.Timezone attribute*), 25
overrunLength (*flightplandb.datatypes.Runway attribute*), 28

P

pacots() (*in module flightplandb.nav*), 8
patch() (*in module flightplandb.internal*), 35
ping() (*in module flightplandb.api*), 7
Plan (*class in flightplandb.datatypes*), 19
planCount (*flightplandb.datatypes.Tag attribute*), 25
PlanQuery (*class in flightplandb.datatypes*), 22
plans() (*in module flightplandb.user*), 14
plansCount (*flightplandb.datatypes.User attribute*), 16
plansDistance (*flightplandb.datatypes.User attribute*), 16
plansDownloads (*flightplandb.datatypes.User attribute*), 16
plansLikes (*flightplandb.datatypes.User attribute*), 16
popularity (*flightplandb.datatypes.Plan attribute*), 21
popularity (*flightplandb.datatypes.Tag attribute*), 25
post() (*in module flightplandb.internal*), 35

Q

q (*flightplandb.datatypes.PlanQuery attribute*), 22

R

range (*flightplandb.datatypes.Navaid attribute*), 27
regionName (*flightplandb.datatypes.Airport attribute*), 29
release (*flightplandb.datatypes.Cycle attribute*), 19
request() (*in module flightplandb.internal*), 33
revoke() (*in module flightplandb.api*), 7
Route (*class in flightplandb.datatypes*), 19
route (*flightplandb.datatypes.Plan attribute*), 22
route (*flightplandb.datatypes.Track attribute*), 31
RouteNode (*class in flightplandb.datatypes*), 18
Runway (*class in flightplandb.datatypes*), 27
runway (*flightplandb.datatypes.Navaid attribute*), 26
runwayCount (*flightplandb.datatypes.Airport attribute*), 30
RunwayEnds (*class in flightplandb.datatypes*), 25
runwayIdent (*flightplandb.datatypes.SearchNavaid attribute*), 31
runways (*flightplandb.datatypes.Airport attribute*), 30

S

search() (*in module flightplandb.nav*), 8
search() (*in module flightplandb.plan*), 11
search() (*in module flightplandb.user*), 14
SearchNavaid (*class in flightplandb.datatypes*), 31
slope (*flightplandb.datatypes.Navaid attribute*), 27
status_code (*flightplandb.exceptions.BadRequestException attribute*), 32
status_code (*flightplandb.exceptions.BaseErrorHandler attribute*), 32
status_code (*flightplandb.exceptions.ForbiddenException attribute*), 33
status_code (*flightplandb.exceptions.InternalServerException attribute*), 33
status_code (*flightplandb.exceptions.NotFoundException attribute*), 33
status_code (*flightplandb.exceptions.TooManyRequestsException attribute*), 33
status_code (*flightplandb.exceptions.UnauthorizedException attribute*), 32
status_handler() (*in module flightplandb.exceptions*), 33
StatusResponse (*class in flightplandb.datatypes*), 15
sunrise (*flightplandb.datatypes.Times attribute*), 25
sunset (*flightplandb.datatypes.Times attribute*), 25
surface (*flightplandb.datatypes.Runway attribute*), 28

T

TAF (*flightplandb.datatypes.Weather attribute*), 29
Tag (*class in flightplandb.datatypes*), 24
tags (*flightplandb.datatypes.Plan attribute*), 21
tags (*flightplandb.datatypes.PlanQuery attribute*), 23
thresholdOffset (*flightplandb.datatypes.Runway attribute*), 28

Times (*class in flightplandb.datatypes*), 25
times (*flightplandb.datatypes.Airport attribute*), 30
Timezone (*class in flightplandb.datatypes*), 25
timezone (*flightplandb.datatypes.Airport attribute*), 30
to (*flightplandb.datatypes.PlanQuery attribute*), 22
toICAO (*flightplandb.datatypes.GenerateQuery attribute*), 23
toICAO (*flightplandb.datatypes.Plan attribute*), 20
toICAO (*flightplandb.datatypes.PlanQuery attribute*), 22
toName (*flightplandb.datatypes.Plan attribute*), 20
toName (*flightplandb.datatypes.PlanQuery attribute*), 23
TooManyRequestsException, 33
Track (*class in flightplandb.datatypes*), 30
type (*flightplandb.datatypes.Frequency attribute*), 28
type (*flightplandb.datatypes.Navaid attribute*), 26
type (*flightplandb.datatypes.RouteNode attribute*), 18
type (*flightplandb.datatypes.SearchNavaid attribute*), 31
type (*flightplandb.datatypes.Via attribute*), 17

U

UnauthorizedException, 32
units() (*in module flightplandb.api*), 6
unlike() (*in module flightplandb.plan*), 12
updatedAt (*flightplandb.datatypes.Plan attribute*), 21
url (*flightplandb.datatypes.Application attribute*), 17
useAWYHI (*flightplandb.datatypes.GenerateQuery attribute*), 24
useAWYLO (*flightplandb.datatypes.GenerateQuery attribute*), 24
useNAT (*flightplandb.datatypes.GenerateQuery attribute*), 23
usePACOT (*flightplandb.datatypes.GenerateQuery attribute*), 23
User (*class in flightplandb.datatypes*), 15
user (*flightplandb.datatypes.Plan attribute*), 21
username (*flightplandb.datatypes.User attribute*), 15
username (*flightplandb.datatypes.UserSmall attribute*), 17
UserSmall (*class in flightplandb.datatypes*), 16

V

validFrom (*flightplandb.datatypes.Track attribute*), 31
validTo (*flightplandb.datatypes.Track attribute*), 31
validtypes (*flightplandb.datatypes.Navaid attribute*), 27
validtypes (*flightplandb.datatypes.RouteNode attribute*), 18
validtypes (*flightplandb.datatypes.SearchNavaid attribute*), 32
validtypes (*flightplandb.datatypes.Via attribute*), 17
version() (*in module flightplandb.api*), 6
Via (*class in flightplandb.datatypes*), 17
via (*flightplandb.datatypes.RouteNode attribute*), 18

W

waypoints (*flightplandb.datatypes.Plan attribute*), 20
Weather (*class in flightplandb.datatypes*), 29
weather (*flightplandb.datatypes.Airport attribute*), 30
westLevels (*flightplandb.datatypes.Route attribute*), 19
width (*flightplandb.datatypes.Runway attribute*), 27

Y

year (*flightplandb.datatypes.Cycle attribute*), 19