pyflight Documentation

Release 0.1.2

Volcyy

Aug 03, 2017

Contents

Contents:

CHAPTER 1

API Reference

This page shows the functions and classes exposed by pyflight. A lot of attributes wrap the required parameters for the QPX API and thus result in documentation similiar to the one found on the official QPX Express API reference, licensed under the Creative Commons Attribution 3.0 License.

Basic Configuration

pyflight.**set_api_key**(*key: str*)
Set the API key to use with the API.

Parameters **key** (*str*) – The API key to execute requests with.

Making Requests

class pyflight.**Request**
Represents a Request that can be sent to the API instead of using a dictionary manually.

Please note that each Request requires at least 1 adult or senior passenger. Optional attributes default to `None`.

raw_data
dict – The raw JSON / dictionary data which will be sent to the API.

adult_count
int – The amount of passengers that are adults.

children_count
int – The amount of passengers that are children.

infant_in_lap_count
int – The amount of passengers that are infants travelling in the lap of an adult.

infant_in_seat_count
int – The amount of passengers that are infants assigned a seat.

senior_count
int – The amount of passengers that are senior citizens.

max_price
Optional[str] – The maximum price below which results should be returned. The currency is specified in ISO-4217, and setting this attribute is validated using the regex `[A-Z]{3}\d+(\.\d+)?`. If it does not match, a `ValueError` is raised.

sale_country
Optional[str] – The IATA country code representing the point of sale. Determines the currency.

ticketing_country
Optional[str] – The IATA country code representing the point of ticketing, for example `DE`.

refundable
Optional[bool] – Whether to return only results with refundable fares or not.

solution_count
int – The amount of solutions to return. Defaults to 1, maximum is 500. Raises a `ValueError` when trying to assign a value outside of 1 to 500.

add_slice(*slice_: pyflight.requester.Slice*)
Adds a slice to this Request.

Parameters **slice** (*Slice*) – The Slice to be added to the request.

Returns To ease chaining of this function, `self` is returned.

Return type self

adult_count
The amount of passengers that are adults.

as_dict() → dict
Returns the raw data associated with this request, which is sent to the API when calling send_sync or send_async.

children_count
The amount of passengers that are children.

infant_in_lap_count
The amount of passengers that are infants travelling in the lap of an adult.

infant_in_seat_count
The amount of passengers that are infants assigned a seat.

max_price
The maximum price below which results should be returned, specified in ISO-421 format.

refundable
Whether to return only results with refundable fares or not.

sale_country
The IATA country code representing the point of sale. Determines the currency.

send_async(*use_containers: bool = True*) → typing.Union[pyflight.results.Result, dict]
Asynchronously execute a request.

Internally, this calls *pyflight.send_async()*. You can also call the function directly. For further information, please view documentation for *pyflight.send_async()*.

send_sync(*use_containers: bool = True*) → typing.Union[pyflight.results.Result, dict]
Synchronously execute a request.

Internally, this calls *pyflight.send_sync()*. You can also call the function directly. For further information, please view documentation for *pyflight.send_sync()*.

senior_count
The amount of passengers that are senior citizens.

solution_count
The amount of solutions to return. Defaults to 1.

ticketing_country
The IATA country code representing the point of ticketing, for example DE.

class pyflight.**Slice**(*origin: str*, *destination: str*, *date: str*)
Represents a slice that makes up a single itinerary of this trip.

For example, for one-way trips, usually one slice is used. A round trip would use two slices. (e.g. SFO - FRA - SFO)

Optional attributes default to None or an empty list if applicable, but can be set if wanted.

raw_data
dict – The raw JSON / dictionary data which will be sent to the API.

origin
str – The airport or city IATA designator of the origin.

destination
str – The airport or city IATA designator of the destination.

date
str – The date on which this flight should take place, in the format YYYY-MM-DD.

max_stops
Optional[int] – The maximum amount of stops that the passenger(s) are willing to accept on this slice.

max_connection_duration
Optional[int] – The longest duration (in minutes) between two legs that passengers are willing to accept

preferred_cabin
Optional[str] – The preferred cabin for this slice. Allowed values are COACH, PREMIUM_COACH, BUSINESS, and FIRST. A ValueError is raised if a value is assigned that is not listed above.

earliest_departure_time
Optional[str] – The earliest time for departure, local to the point of departure. Formatted as HH:MM.

latest_departure_time
Optional[str] – The latest time for departure, local to the point of departure. Formatted as HH:MM.

permitted_carriers
List[str] – A list of 2-letter IATA airline designators for which results should be returned.

prohibited_carriers
List[str] – A list of 2-letter IATA airline designators, for which no results will be returned.

date
The date on which this flight should take place, in the format YYYY-MM-DD.

destination
The airport or city IATA designator of the destination.

earliest_departure_time
The earliest time for departure, local to the point of departure. Formatted as HH:MM.

latest_departure_time
The latest time for departure, local to the point of departure. Formatted as HH:MM.

max_connection_duration
The longest duration (in minutes) between two legs that passengers are willing to accept

max_stops
The maximum amount of stops that the passenger(s) are willing to accept on this slice.

origin
The airport or city IATA designator of the origin.

permitted_carriers
A list of 2-letter IATA airline designators for which results should be returned.

preferred_cabin
The preferred cabin for this slice. Allowed values are COACH, PREMIUM_COACH, BUSINESS, and FIRST. A `ValueError` is raised if a value is assigned that is not listed above.

prohibited_carriers
A list of 2-letter IATA airline designators,

for which no results will be returned.

pyflight.**send_async**(*request_body: typing.Union[dict, pyflight.requester.Request], use_containers: bool = True*)

Asynchronously execute and send a JSON Request or a *Request*. This is a coroutine - calling this function must be awaited.

Parameters

- **request_body** (*Union[dict, Request]*) – The body of the request to be sent to the API. This must follow the structure described here: https://developers.google.com/qpx-express/v1/trips/search It is heavily recommended to use *Request* instead of constructing request bodies manually.
- **use_containers** (*Optional[bool]*) – Whether the containers given should be used or not. If False is given, any API call will return a dictionary of the "raw" API data without any modification. Otherwise, an API call will return a `Result` object.

Raises *APIException* – If the API call did not return the normal *200* status code and thus, an error occurred.

Returns

- `Result` – If `use_containers` is `True` and no Error occurred.
- *dict* – If `use_containers` is `False`, as a raw dictionary without any adjustments.

pyflight.**send_sync**(*request_body: typing.Union[dict, pyflight.requester.Request], use_containers: bool = True*)

Synchronously execute and send a JSON-Request or a :class:'Request. Note that this function is blocking.

Parameters

- **request_body** (*Union[dict, Request]*) – The body of the request to be sent to the API. This must follow the structure described here: https://developers.google.com/qpx-express/v1/trips/search It is heavily recommended to use *Request* instead of constructing request bodies manually.

- **use_containers** (*Optional[bool]*) – Whether the containers given should be used or not. If False is given, any API call will return a dictionary of the "raw" API data without any modification. Otherwise, the API call will return a `Result` object.

Raises *APIException* – If the API call did not return the normal *200* status code and thus, an error occurred.

Returns

- `Result` – If `use_containers` is `True` and no Error occurred.
- *dict* – If `use_containers` is ‘‘False‘, as a raw dictionary without any adjustments.

class `pyflight.`**APIException**(*code: int*, *message: str*, *reason: str*, **args*, ***kwargs*)

Custom Exception that is raised from the Requests when an API call goes wrong, meaning the API did not return a status code of 200.

code
int – The code of the Error that was returned

message
str – The error message as returned by the API

reason
str – The reason as specified by the API

Examples

```
try:
    flight_info = send_sync(my_request_body, use_containers=False)
except pyflight.APIException as err:
    print('Error trying to execute a request:')
    print(err)
else:
    ...
```

The Exception will be formatted as: *‘<status-code>: <error-message> (reason)’*, for example `400: Bad Request (keyInvalid)`

Working with the Response

These provide several Classes that contain the Results of a Request to simplify accessing them, as well as offering several Methods to work with the Data from the Result.

Some of the Documentation is extracted from the resource reference from the API itself, for which a full documentation can be found here: https://developers.google.com/qpx-express/v1/trips/search

class `pyflight.results.`**Result**(*data: dict*)

Contains Results of an API Call.

This Class supports various *magic methods*:

x == y Checks if two *Result*s are identical. This is equivalent to `x.request_id == y.request_id`.

x != y Checks if two *Result*s are not identical to each other. This is equivalent to `x.request_id != y.request_id`.

str(x) Returns the `request_id` for the *Result* this is invoked on.

for trip in x This will call __iter__ of *Result* and return an iterator over the *Trip*s saved in this *Result*.

request_id
str – Specifies the Request ID, unique for each Request.

airports
List[*Airport*] – Contains Data for the Flights found in the Response.

aircraft
List[Aircraft] – Contains the Code and the Name of the Aircraft found in the Response.

carriers
List[*Carrier*] – Contains the Code and the Name of the Carriers found in the Response.

cities
List[*City*] – Contains the Code and the Name of the Cities found in the Response.

taxes
List[*Tax*] – Contains the Code and the Name of the Taxes found in the Response.

trips
List[*Trip*] – Contains information about trips (itinerary solutions) returned by the API. The Amount of Trips is determined by the amount of Solutions set in the Request.

as_dict() → dict
Returns a dictionary representation of this *Result*.

Useful for serializing data to JSON. Internally, this calls as_dict() on all of its members.

Returns The data stored in this *Result* as key / value pairs.

Return type dict

class pyflight.results.**Carrier**(*code: str*, *name: str*)
This Class inherits from FlightData and thus, supports all operations that FlightData supports. This represents a Tax with a code (unique identifier) and a Name. This will also be reflected in the Pricing section of a Trip, but with more information such as the charge type, the country, and the price of the Tax. For Examples, view the "Examples" section for FlightData.

class pyflight.results.**City**(*code: str*, *name: str*)
This Class inherits from FlightData and thus, supports all operations that FlightData supports. This represents a Tax with a code (unique identifier) and a Name. This will also be reflected in the Pricing section of a Trip, but with more information such as the charge type, the country, and the price of the Tax. For Examples, view the "Examples" section for FlightData.

class pyflight.results.**Tax**(*code: str*, *name: str*)
This Class inherits from FlightData and thus, supports all operations that FlightData supports. This represents a Tax with a code (unique identifier) and a Name. This will also be reflected in the Pricing section of a Trip, but with more information such as the charge type, the country, and the price of the Tax. For Examples, view the "Examples" section for FlightData.

class pyflight.results.**Airport**(*airport: dict*)
Contains Data of an Airport and its City Code.

This Class supports various *magic methods*:

x == y Compare two Airports with each other for equality by their Airport and City Codes.

x != y Compare two Airports with each other for inequality.

str(x) Get the Airport's Name

```
>>> str(my_airport)
'ABC International'
```

code
str – The Code of this Airport

name
str – The Name of this Airport

city
str – The Code of the City associated with the Airport

as_dict()
Get a dictionary representation of the Airport.

Example

```
>>> airport = {
    'code': '3E7', 'city': 'XYZ', 'name': 'Example Airport'
}
>>> example_airport = Airport(airport)
>>> example_airport.as_dict()
{
    'code': '3E7',
    'city': 'XYZ',
    'name': 'Example Airport',
}
```

Returns A dictionary representing this Airport.

Return type dict

class pyflight.results.**Trip**(*trip_data: dict*)
Contains Information about one Trip - an itinerary solution - from the API.

This class supports various *magic methods*:

x == y Compares two *Trip*s with each other for equality. Returns `True` when `x.id == y.id`.

x != y Compares two *Trip*s with each other for inequality. Returns `True` when `x.id != y.id`.

str(x) Returns the `id` of the *Trip* this is invoked on.

total_price
str – The total price as Currency followed by the Amount for all Passengers on the Trip, e.g. `'USD59.00'`

id
str – The unique ID given to each Trip

routes
List[`Route`] – A list of Routes from this Trip

pricing
List[`Pricing`] – A list of pricing data from this Trip

as_dict() → dict
Get a dictionary representation of this *Trip*.

Returns A dictionary containing the attributes of this *Trip* as key / value pairs.

Return type dict

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

T