vvspy

Release 1.2.0

Sep 18, 2022

Contents

vvspy is a easy to use, complete API wrapper for the EFA / VVS API.

Features:

- Easy to use with an object oriented design
- Implements the entire VVS API
- full customizable requests and parameters
- Well tested and maintained
- Departures, Arrivals, Trips, Station info, Upcoming events, Maintenance work

CHAPTER 1

Documentation Contents

1.1 Introduction

I always wanted to get some insights on public transport and as I am very programming affine I started investigating into the VVS API. However, I noticed really fast that the EFA system is not really programmer-friendly, at least if you are still getting into it.

I saw some projects from CodeOfGermany and various others, but as they are way out to date, I figured that I want to publish my own solution.

For example I am using this library to track my daily used connections and sending push notifications on my mobile device if one is delayed.

But there are various other things you can done. For example a simple dashboard that displays upcoming departures at a nearby station.

1.1.1 Installation

```
pip install vvspy
```

1.1.2 Basic Concept

For examples see our readme or /example/ folder.

1.2 VVSPY Requests

1.2.1 Departures

vvspy.**get_departures**(*station_id: Union[str, int], check_time: datetime.datetime = None, limit: int = 100, debug: bool = False, request_params: dict = None, return_resp: bool = False, session: requests.sessions.Session = None, **kwargs*) → Union[List[vvspy.obj.departure.Departure], requests.models.Response, None]

Returns: List[*vvspy.obj.Departure*] Returns none on webrequest errors.

Basic usage:

```
results = vvspy.get_departures("5006115", limit=3)  # Stuttgart main station
```

Set proxy for request:

```
proxies = {}  # see https://stackoverflow.com/a/8287752/9850709
results = vvspy.get_departures("5006115", request_params={"proxies": proxies})
```

station_id Union[int, str] Station you want to get departures from. See csv on root of repository to get your id.

check_time Optional[datetime.datetime] Time you want to check. default datetime.now()

limit Optional[int] Limit request/result on this integer. default 100

debug Optional[bool] Get advanced debug prints on failed web requests default False

request_params Optional[dict] params parsed to the api request (e.g. proxies) default {}

return_resp Optional[bool] if set, the function returns the response object of the API request.

session Optional[requests.Session] if set, uses a given requests.session object for requests

kwargs Optional[dict] Check departures.py to see all available kwargs.

vvspy.**get_departure**(*station_id: Union[str, int], check_time: datetime.datetime = None, debug: bool = False, request_params: dict = None, return_resp: bool = False, session: requests.sessions.Session = None, **kwargs*) → Union[vvspy.obj.departure.Departure, requests.models.Response, None]

Same as *get_departures* But limited to one obj as result.

Returns: *vvspy.obj.Departure* Returns none on webrequest errors or no results found.

vvspy.**departures_now**(*station_id: Union[str, int], limit: int = 100, return_resp: bool = False, session: requests.sessions.Session = None, **kwargs*) → Union[List[vvspy.obj.departure.Departure], requests.models.Response, None]

Same as *get_departures* But *datetime.datetime.now()* is already used as parameter.

Returns: List[*vvspy.obj.Departure*] Returns none on webrequest errors or no results found.

1.2.2 Arrivals

vvspy.**get_arrivals**(*station_id: Union[str, int], check_time: datetime.datetime = None, limit: int = 100, debug: bool = False, request_params: dict = None, return_resp: bool = False, session: requests.sessions.Session = None, **kwargs*) → Union[List[vvspy.obj.arrival.Arrival], requests.models.Response, None]

Returns: List[*vvspy.obj.Arrival*] Returns none on webrequest errors.

Basic usage:

```
results = vvspy.get_arrivals("5006115", limit=3)  # Stuttgart main station
```

Set proxy for request:

```
proxies = {}  # see https://stackoverflow.com/a/8287752/9850709
results = vvspy.get_arrivals("5006115", request_params={"proxies": proxies})
```

station_id Union[int, str] Station you want to get arrivals from. See csv on root of repository to get your id.

check_time Optional[datetime.datetime] Time you want to check. default datetime.now()

limit Optional[int] Limit request/result on this integer. default 100

debug Optional[bool] Get advanced debug prints on failed web requests default False

request_params Optional[dict] params parsed to the api request (e.g. proxies) default {}

return_resp Optional[bool] if set, the function returns the response object of the API request.

session Optional[requests.Session] if set, uses a given requests.session object for requests

kwargs Optional[dict] Check arrivals.py to see all available kwargs.

vvspy.**get_arrival**(*station_id: Union[str, int], check_time: datetime.datetime = None, debug: bool = False, request_params: dict = None, return_resp: bool = False, session: requests.sessions.Session = None, **kwargs*) → Union[vvspy.obj.arrival.Arrival, requests.models.Response, None]

Same as *get_arrivals* But limited to one obj as result.

Returns: *vvspy.obj.Arrival* Returns none on webrequest errors or no results found.

1.2.3 Trips

vvspy.**get_trips**(*origin_station_id: Union[str, int], destination_station_id: Union[str, int], check_time: datetime.datetime = None, limit: int = 100, debug: bool = False, request_params: dict = None, return_resp: bool = False, session: requests.sessions.Session = None, **kwargs*) → Union[List[vvspy.obj.trip.Trip], requests.models.Response, None]

Returns: List[*vvspy.obj.Trip*] Returns none on webrequest errors.

Basic usage:

```
results = vvspy.get_trips("5006115", "5006465", limit=3)  # Stuttgart main␣
↪station to Zuffenhausen
```

Set proxy for request:

```
proxies = {}  # see https://stackoverflow.com/a/8287752/9850709
results = vvspy.get_arrivals("5006115", "5006465", request_params={"proxies":␣
↪proxies})
```

station_id Union[int, str] Station you want to get trips from. See csv on root of repository to get your id.

check_time Optional[datetime.datetime] Time you want to check. default datetime.now()

limit Optional[int] Limit request/result on this integer. default 100

debug Optional[bool] Get advanced debug prints on failed web requests default False

request_params Optional[dict] params parsed to the api request (e.g. proxies) default {}

return_resp Optional[bool] if set, the function returns the response object of the API request.

session Optional[requests.Session] if set, uses a given requests.session object for requests

kwargs Optional[dict] Check trips.py to see all available kwargs.

vvspy.**get_trip**(*origin_station_id: Union[str, int], destination_station_id: Union[str, int], check_time: datetime.datetime = None, debug: bool = False, request_params: dict = None, return_resp: bool = False, session: requests.sessions.Session = None, **kwargs*) → Union[vvspy.obj.trip.Trip, requests.models.Response, None]

Same as *get_trips* But limited to one obj as result.

Returns: *vvspy.obj.Trip* Returns none on webrequest errors or no results found.

1.3 VVSPY Models

1.3.1 Departures

class vvspy.obj.**Departure**

Departure object from a departure request of one station.

raw dict Raw dict received by the API.

stop_id str Station_id of the departure.

x str Coordinates of the station.

y str Coordinates of the station.

map_name str Map name the API works on.

area str The area of the station (unsure atm)

platform str Platform / track of the departure.

platform_name str name of the platform.

stop_name str name of the station.

name_wo str name of the station.

countdown int minutes until departure.

datetime datetime.datetime Planned departure datetime.

real_datetime datetime.datetime Estimated departure datetime (equal to `self.datetime` if no real-time data is available).

delay int Delay of departure in minutes.

serving_line *ServingLine* line of the incoming departure.

operator *LineOperator* Operator of the incoming departure.

stop_infos: Optional[dict] All related info to the station (e.g. maintenance work).

line_infos Optional[dict] All related info to the station (e.g. maintenance work).

1.3.2 Arrivals

class vvspy.obj.**Arrival**

Arrival object from a arrival request of one station.

raw **dict** Raw dict received by the API.

stop_id **str** Station_id of the arrival.

x **str** Coordinates of the station.

y **str** Coordinates of the station.

map_name **str** Map name the API works on.

area **str** The area of the station (unsure atm)

platform **str** Platform / track of the arrival.

platform_name **str** name of the platform.

stop_name **str** name of the station.

name_wo **str** name of the station.

countdown **int** minutes until arrival.

datetime **datetime.datetime** Planned arrival datetime.

real_datetime **datetime.datetime** Estimated arrival datetime (equal to `self.datetime` if no realtime data is available).

delay **int** Delay of arrival in minutes.

serving_line ***ServingLine*** line of the incoming arrival.

operator ***LineOperator*** Operator of the incoming arrival.

stop_infos **Optional[dict]** All related info to the station (e.g. maintenance work).

line_infos **Optional[dict]** All related info to the station (e.g. maintenance work).

1.3.3 Trips

class vvspy.obj.**Trip**

Result object from a trip request from one station to another including interchanges

raw **dict** Raw dict received by the API.

connections **List[*Connection*]** List of connections the trip consists of.

duration **int** seconds the trip takes overall.

zones **Optional[List[str]]** List of zones this trip goes through.

fare **Optional[dict]** misc info about this trip, ticket prices, etc.

Connection

class vvspy.obj.**Connection**

Several connections describe one *Trip*.

raw **dict** Raw dict received by the API.

duration **int** seconds this connection takes

is_realtime_controlled `bool` whether or not this connection has realtime tracking

origin *Origin* Origin, where this connection starts

destination *Destination* Where this connection is heading to

transportation *Transportation* Transportation info of this connection

stop_sequence Optional[List[`dict`]] stop sequence of this connection

foot_path_info Optional[] Info if you really want to walk ?

infos Optional[List[]] ~

coords Optional[List[List[`int`]]] coords of this connection

path_description Optional[] ~

interchange Optional[] ~

properties Optional[`dict`] misc info about this connection

Origin

class vvspy.obj.**Origin**

Describes the origin of a *Connection*.

raw `dict` Raw dict received by the API.

is_global_id `bool` ~

id `str` station id of the origin station

name `str` name of the origin station

disassembled_name Optional[`str`] detailed name of the origin station.

type `str` type of the origin station. (e.g. bus, track)

point_type Optional[`str`] ~

coord List[`int`] coords of the station

niveau `int` ~

parent `dict` ~

departure_time_planned `datetime.datetime` Time planned of arrival.

departure_time_estimated `datetime.datetime` Time estimated with realtime info (same as *departure_time_planned* if no realtime data is available).

delay `int` Minutes of delay.

properties `dict` misc info about the origin.

Destination

class vvspy.obj.**Destination**

Describes the destination of a *Connection*.

raw `dict` Raw dict received by the API.

is_global_id `bool` ~

id `str` station id of the destination station

name `str` name of the destination station

disassembled_name `Optional[str]` detailed name of the destination station.

type `str` type of the destination station. (e.g. bus, track)

point_type `Optional[str]` ~

coord `List[int]` coords of the station

niveau `int` ~

parent `dict` ~

arrival_time_planned `datetime.datetime` Time planned of arrival.

arrival_time_estimated `datetime.datetime` Time estimated with realtime info (same as *arrival_time_planned* if no realtime data is available).

delay `int` Minutes of delay.

properties `dict` misc info about the destination.

Transportation

class vvspy.obj.**Transportation**

Describes info about transportation of a *Connection*.

raw `dict` Raw dict received by the API.

id `str` id of the transportation.

name `str` name of the transportation.

disassembled_name `str` detailed name of the transportation.

number `str` line number of the transportation.

description `str` description, most of the time the string that is displayed on the bus/train itself.

product `dict` describes the mean of transport (bus, train, etc.)

operator *LineOperator* describes the Operator of the transport.

destination `dict` destination of the transport.

properties `dict` misc info about the transport.

1.3.4 ServingLine

class vvspy.obj.**ServingLine**

Describes the line, departing or arriving in a Departure/Arrival result.

raw `dict` Raw dict received by the API.

key `str` key (most likely an ID) of the line.

code `str` code (most likely type) of the line.

number `str` number of line (e.g. U12).

symbol `str` symbol displayed on the transport itself (e.g. U12).

mot_type `str` ~

mt_sub_code `str` ~

real_time `bool` whether or not the transport supports realtime tracking.

direction `str` Last station the transport is heading to.

direction `str` Last station the transport is heading to.

direction_from `str` Starting station of the transport.

name `str` name of the line type (e.g. Stadtbahn).

train_num `str` Last station the transport is heading to.

delay `Optional[str]` Minutes of delay.

li_erg_ri_proj `dict` Detailed line information (e.g. network)

dest_id `str` station id of the destination

stateless `str` ~

1.3.5 LineOperator

class vvspy.obj.**LineOperator**

Describes the operator of a *Connection*.

raw `dict` Raw dict received by the API.

id `str` id of the operator.

name `str` display name of the operator.

public_code `str` public_code of the operator.

1.4 Need Help?

Feel free to open an issue on Github.

1.5 License

Index