# pyHaFAS

***Release 0.3.1***

**n0emis, em0lar**

**Jul 01, 2023**

# INSTALLATION AND USAGE

pyHaFAS is a client for the API of HaFAS public transport management system.

HaFAS is a software sold by the company "HaCon" and is used by a lot of public transport providers for routing and providing departure information to their customers.

Every public transport providers using HaFAS has their own deployment. In general all of them have the same API but there are some small differences between them. To cover this we have a *profile*[G] for each HaFAS deployment.

**Warning:** pyHaFAS is still in beta. The interface might change, so please read the changelog carefully before you update.

CHAPTER

ONE

# CONTRIBUTING

If you have a question, found a bug or want to propose a feature, have a look at the issues page. Even better than creating an issue is creating a pull request. If you want to do that please read the *development introduction*.

## 1.1 Get started

### 1.1.1 Terminology

In pyHaFAS, we often use the following terms. Please read them, so you can understand the documentation and project better.

Most other pyHaFAS-specific words are defined in the *Glossary*. If one of these words is used in this documentation it's marked as a link with a superscript G as follows: *profile*[G]

| Term | Meaning |
|---|---|
| profile | Customization for each HaFAS deployment - Contains the endpoint, tokens and possible changes for the deployment |
| FPTF | Abbreviation for Friendly Public Transport Format - Used as the basis for returned data |
| Station Board | Generalization for `arrivals` and `departures` requests |

### 1.1.2 Installation

You only need to install the pyhafas package, for example using pip:

```
$ pip install pyhafas
```

### 1.1.3 Sample Starter Code

Below is a sample code for easy usage. It has multiple parts:

1. It imports the `HafasClient` and the `DBProfile` of pyHaFAS and creates the `HafasClient` with the *profile*[G]. The `DBProfile` is the *profile*[G] belonging to the HaFAS deployment of Deutsche Bahn.
2. It searches for locations (stations) with the term "Berlin" and prints the result
3. It searches for departing trains at Berlin Central Station. Every station is identified by an ID, which (in this case *8011160*) can be obtained by a *location*-request with pyhafas.

```
import datetime
from pyhafas import HafasClient
from pyhafas.profile import DBProfile

client = HafasClient(DBProfile())

print(client.locations("Berlin"))

print(client.departures(
    station='8011160',
    date=datetime.datetime.now(),
    max_trips=5
))
```

### 1.1.4 What's next?

For a good start with pyHaFAS you should go on reading the documentation. Especially the pages *Usage Examples* and *Profiles* are a good start.

## 1.2 Profiles

Here's a list of all HaFAS deployments pyHaFAS supports. If the *profile*[G] has any differences, they will be mentioned here. Also, available and default *products*[G] are defined.

### 1.2.1 Deutsche Bahn (DB)

#### Usage

```
from pyhafas.profile import DBProfile
client = HafasClient(DBProfile())
```

#### Available Products

| pyHaFAS Internal Name | Example Train Type |
|---|---|
| long_distance_express | ICE/ECE |
| long_distance | IC/EC |
| regional_express | RE/IRE |
| regional | RB |
| suburban | S |
| bus | BUS |
| ferry | F |
| subway | U |
| tram | STR/T |
| taxi | Group Taxi |

#### Default Products

All available products specified above are enabled by default.

#### Other interesting Stuff

- Mapping list with station IDs exists: https://data.deutschebahn.com/dataset/data-haltestellen

### 1.2.2 Verkehrsverbund Süd-Niedersachsen (VSN)

#### Usage

```
from pyhafas.profile import VSNProfile
client = HafasClient(VSNProfile())
```

#### Available Products

| pyHaFAS Internal Name | Example Train Type |
|---|---|
| long_distance_express | ICE/ECE |
| long_distance | IC/EC/CNL |
| regional_express | RE/IRE |
| regional | NV (e.g. RB) |
| suburban | S |
| bus | BUS |
| ferry | F |
| subway | U |
| tram | STR/T |
| anruf_sammel_taxi | Group Taxi |

#### Default Products

All available products specified above are enabled by default.

#### Specialities

- The *max_trips* filter in station board (departures/arrival) requests seems not to work

### 1.2.3 Kölner Verkehrsbetriebe (KVB)

#### Usage

```
from pyhafas.profile import KVBProfile
client = HafasClient(KVBProfile())
```

#### Available Products

| pyHaFAS Internal Name | Example Train Type |
|---|---|
| s-bahn | S |
| stadtbahn | U |
| bus | BUS |
| fernverkehr | ICE/ECE/IC/EC |
| regionalverkehr | RE/IRE |
| taxibus | Group Taxi |

#### Default Products

All available products specified above are enabled by default.

## 1.3 HafasClient

The *HafasClient* is the interface between your program and pyhafas internal code. You need it when you're using pyHaFAS. Below you can find the API of the client.

**class** pyhafas.client.**HafasClient**(*profile*, *ua='pyhafas'*, *debug=False*)

The interface between the user's program and pyHaFAS internal code.

**Parameters**

- **profile** (*ProfileInterface*) – *Profile* to be used
- **ua** (str) – (optional, not recommended to change) The user-agent which will be sent to HaFAS. By default "pyhafas", but is often overwritten by profile to emulate the app.
- **debug** (bool) – (optional) Whether debug mode should be enabled. Defaults to False.

**arrivals**(*station*, *date*, *max_trips=- 1*, *duration=- 1*, *products={}*, *direction=None*)

Returns arriving trips at the specified station

To get detailed information on the trip use the *trip* method with the id

**Parameters**

- **station** (Union[*Station*, str]) – FPTF *Station* object or ID of station
- **date** (datetime) – Date and Time when to search
- **max_trips** (int) – (optional) Maximum number of trips to be returned. Default is "whatever HaFAS wants"
- **duration** (int) – (optional) Minutes after *date* in which is search is made. Default is "whatever HaFAS wants"
- **products** (Dict[str, bool]) – (optional) Dict of product name(s) and whether it should be enabled or not. Modifies the default products specified in the profile.
- **direction** (Union[*Station*, str, None]) – (optional) Direction (end) station of the vehicle. Default is any direction station is allowed

**Return type** List[*StationBoardLeg*]

**Returns** List of FPTF *StationBoardLeg* objects with arriving trips

**departures**(*station*, *date*, *max_trips=- 1*, *duration=- 1*, *products={}*, *direction=None*)
Returns departing trips at the specified station

To get detailed information on the trip use the *trip* method with the id

**Parameters**

- **station** (Union[*Station*, str]) – FPTF *Station* object or ID of station
- **date** (datetime) – Date and Time when to search
- **max_trips** (int) – (optional) Maximum number of trips to be returned. Default is "whatever HaFAS wants"
- **duration** (int) – (optional) Minutes after *date* in which is search is made. Default is "whatever HaFAS wants"
- **products** (Dict[str, bool]) – (optional) Dict of product name(s) and whether it should be enabled or not. Modifies the default products specified in the profile.
- **direction** (Union[*Station*, str, None]) – (optional) Direction (end) station of the vehicle. Default is any direction station is allowed

**Return type** List[*StationBoardLeg*]

**Returns** List of FPTF *StationBoardLeg* objects with departing trips

**journey**(*journey*)
Returns information about a specific journey by its ID

Useful if you want to refresh the data of the trip, e.g. the real-time data.

**Parameters** **journey** (Union[*Journey*, str]) – FPTF *Journey* object or journey ID

**Return type** *Journey*

**Returns** FPTF *Journey* object with current/updated information

**journeys**(*origin*, *destination*, *date*, *via=[]*, *min_change_time=0*, *max_changes=- 1*, *products={}*, *max_journeys=- 1*)
Returns possible journeys between two destinations

Possible journeys between two destinations are calculated by HaFAS and returned. It's also possible to add multiple via stations.

**Parameters**

- **origin** (Union[*Station*, str]) – FPTF *Station* object or ID of origin/starting station
- **destination** (Union[*Station*, str]) – FPTF *Station* object or ID of destination/ending station
- **date** (datetime) – Date and Time when to search
- **via** (List[Union[*Station*, str]]) – (optional) List of via stations. The route is calculated via all of these stations in the order of the list. The stations have to be a FPTF *Station* object or the ID of the station. The default is no via stations.
- **min_change_time** (int) – (optional) Minimum transfer/change time at each station. Default is the default that HaFAS specifies internal.
- **max_changes** (int) – (optional) Maximum number of changes. Default is unlimited.
- **products** (Dict[str, bool]) – (optional) Dict of product name(s) and whether it should be enabled or not. Modifies the default products specified in the profile.

- **max_journeys** (int) – (optional) Maximum number of returned journeys. Default is the default that HaFAS specifies internal.

**Return type** List[*Journey*]

**Returns** List of FPTF *Journey* objects

**journeys_from_leg**(*origin*, *destination*, *via=[]*, *min_change_time=0*, *max_changes=- 1*, *products={}*)

Returns possible journeys from a leg to a destination

Possible journeys between two destinations are calculated by HaFAS and returned. It's also possible to add multiple via stations.

**Parameters**

- **origin** (*Leg*) – FPTF *Leg* object from where to search
- **destination** (Union[*Station*, str]) – FPTF *Station* object or ID of destination/ending station
- **via** (List[Union[*Station*, str]]) – (optional) List of via stations. The route is calculated via all of these stations in the order of the list. The stations have to be a FPTF *Station* object or the ID of the station. The default is no via stations.
- **min_change_time** (int) – (optional) Minimum transfer/change time at each station. Default is the default that HaFAS specifies internal.
- **max_changes** (int) – (optional) Maximum number of changes. Default is unlimited.
- **products** (Dict[str, bool]) – (optional) Dict of product name(s) and whether it should be enabled or not. Modifies the default products specified in the profile.

**Return type** List[*Journey*]

**Returns** List of FPTF *Journey* objects

**locations**(*term*, *rtype='S'*)

Returns stations (and addresses) that are searched with the provided term

The further forward the station is in the list, the higher the similarity to the search term.

**Parameters**

- **term** (str) – Search term
- **rtype** (str) – Result types. One of ['S' for stations only, 'ALL' for addresses and stations]

**Return type** List[*Station*]

**Returns** List of FPTF *Station* objects

**nearby**(*location*)

Not implemented yet.

**radar**(*north*, *west*, *south*, *east*)

Not implemented yet.

**stop**(*stop*)

Not implemented yet.

**trip**(*id*)

Returns detailed information about a trip based on its ID

**Parameters** **id** (str) – ID of the trip

**Return type** *Leg*

**Returns** Detailed trip information as FPTF *Leg* object

## 1.4 Friendly Public Transport Format (FPTF)

Most types used in pyHaFAS are specified in the Friendly Public Transport Format. With this specification, we build python classes in the module *pyhafas.types.fptf*. You can find the reference for those classes below.

**class** pyhafas.types.fptf.**FPTFObject**
Bases: object

**class** pyhafas.types.fptf.**Journey**(*id*, *date=None*, *duration=None*, *legs=None*)
Bases: *pyhafas.types.fptf.FPTFObject*

FPTF *Journey* object

A journey is a computed set of directions to get from A to B at a specific time. It would typically be the result of a route planning algorithm.

**Variables**

- **id** (*str*) – ID of the Journey
- **date** (*Optional[datetime.date]*) – Starting date of the journey (maybe *None*)
- **duration** (*Optional[datetime.timedelta]*) – Duration of the complete journey (maybe *None*)
- **legs** (*Optional[List[Leg]]*) – Longitude coordinate of the Station (maybe *None*)

**class** pyhafas.types.fptf.**Leg**(*id*, *origin*, *destination*, *departure*, *arrival*, *mode=<Mode.TRAIN>*, *name=None*, *cancelled=False*, *distance=None*, *departure_delay=None*, *departure_platform=None*, *arrival_delay=None*, *arrival_platform=None*, *stopovers=None*, *remarks=None*)
Bases: *pyhafas.types.fptf.FPTFObject*

FPTF *Leg* object

A leg or also named trip is most times part of a journey and defines a journey with only one specific vehicle from A to B.

**Variables**

- **id** (*str*) – ID of the Leg
- **origin** (Station) – FPTF *Station* object of the origin station
- **destination** (Station) – FPTF *Station* object of the destination station
- **departure** (*datetime.datetime*) – Planned Date and Time of the departure
- **arrival** (*datetime.datetime*) – Planned Date and Time of the arrival
- **mode** (Mode) – Type of transport vehicle - Must be a part of the FPTF *Mode* enum. Defaults to *Mode.TRAIN*
- **name** (*Optional[str]*) – Name of the trip (e.g. ICE 123) (maybe *None*)
- **cancelled** (*bool*) – Whether the trip is completely cancelled (not only some stops)
- **distance** (*Optional[int]*) – Distance of the walk trip in metres. Only set if *mode* is *Mode.WALKING* otherwise None

- **departureDelay** (*Optional[datetime.timedelta]*) – Delay at the departure station (maybe *None*)
- **departurePlatform** (*Optional[str]*) – Real-time platform at the departure station (maybe *None*)
- **arrivalDelay** (*Optional[datetime.timedelta]*) – Delay at the arrival station (maybe *None*)
- **arrivalPlatform** (*Optional[str]*) – Real-time platform at the arrival station (maybe *None*)
- **stopovers** (*Optional[List[Stopover]]*) – List of FPTF *Stopover* objects (maybe *None*)
- **remarks** (*List[Remark]*) – (optional) List of remarks

**class** pyhafas.types.fptf.**Mode**(*value*)

Bases: enum.Enum

FPTF *Mode* object

The mode of a *Leg* specifies the general type of transport vehicle (it can also be a walking leg)

**AIRCRAFT = 'aircraft'**

**BICYCLE = 'bicycle'**

**BUS = 'bus'**

**CAR = 'car'**

**GONDOLA = 'gondola'**

**TAXI = 'taxi'**

**TRAIN = 'train'**

**WALKING = 'walking'**

**WATERCRAFT = 'watercraft'**

**class** pyhafas.types.fptf.**Remark**(*remark_type=None*, *code=None*, *subject=None*, *text=None*, *priority=None*, *trip_id=None*)

Bases: *pyhafas.types.fptf.FPTFObject*

A remark is a textual comment/information, usually added to a Stopover or Leg

**Variables**

- **remark_type** (*Optional[str]*) – Type/Category of the remark. May have a different meaning depending on the network
- **code** (*Optional[str]*) – Code of the remark. May have a different meaning depending on the network
- **subject** (*Optional[str]*) – Subject of the remark
- **text** (*Optional[str]*) – Actual content of the remark
- **priority** (*Optional[int]*) – Priority of the remark, higher is better
- **trip_id** (*Optional[str]*) – ID to a Trip added to this remark (e.g. a replacement train)

**class** pyhafas.types.fptf.**Station**(*id*, *lid=None*, *name=None*, *latitude=None*, *longitude=None*)
Bases: *pyhafas.types.fptf.FPTFObject*

FPTF *Station* object

A station is a point where vehicles stop. It may be a larger building or just a small stop without special infrastructure.

**Variables**

- **id** (*str*) – ID of the Station. Typically a number but as a string
- **lid** (*Optional[str]*) – Location ID of the Station (maybe *None*). Long-form, containing multiple fields
- **name** (*Optional[str]*) – Name of the Station (maybe *None*)
- **latitude** (*Optional[float]*) – Latitude coordinate of the Station (maybe *None*)
- **longitude** (*Optional[float]*) – Longitude coordinate of the Station (maybe *None*)

**class** pyhafas.types.fptf.**StationBoardLeg**(*id*, *name*, *direction*, *station*, *date_time*, *cancelled*, *delay=None*, *platform=None*)
Bases: *pyhafas.types.fptf.FPTFObject*

*StationBoardLeg* object

Returned at Station Board-Requests. This requests do not have enough information for a FPTF *Leg* object. With the ID a *trip* request can be made to get detailed information about the trip

**Variables**

- **id** (*str*) – ID of the Leg
- **name** (*str*) – Name of the trip (e.g. ICE 123)
- **direction** (*str*) – Direction text of the trip (e.g. Berlin Central Station)
- **station** (Station) – FPTF *Station* object of the departing/arriving station
- ***date_time*** (*datetime.datetime*) – Planned Date and Time of the departure/arrival
- **cancelled** (*bool*) – Whether the stop or trip cancelled
- **delay** (*Optional[datetime.timedelta]*) – Delay at the departure station (maybe *None*)
- **platform** (*Optional[str]*) – Real-time platform at the station (maybe *None*)

**class** pyhafas.types.fptf.**Stopover**(*stop*, *cancelled=False*, *arrival=None*, *arrival_delay=None*, *arrival_platform=None*, *departure=None*, *departure_delay=None*, *departure_platform=None*, *remarks=None*)
Bases: *pyhafas.types.fptf.FPTFObject*

FPTF *Stopover* object

A stopover represents a vehicle stopping at a stop/station at a specific time.

**Variables**

- ***stop*** (Station) – Station where the vehicle is stopping
- **cancelled** (*bool*) – Whether the stop is cancelled
- **arrival** (*Optional[datetime.datetime]*) – Planned arrival date and time at the station (maybe *None*)

- **arrivalDelay** (*Optional[datetime.timedelta]*) – Arrival delay at the station (maybe *None*)
- **arrivalPlatform** (*Optional[str]*) – Real-time arrival platform at the station (maybe *None*)
- **departure** (*Optional[datetime.datetime]*) – Planned departure date and time at the station (maybe *None*)
- **departureDelay** (*Optional[datetime.timedelta]*) – Departure delay at the station (maybe *None*)
- **departurePlatform** (*Optional[str]*) – Real-time departure platform at the station (maybe *None*)
- **remarks** (*List[Remark]*) – (optional) List of remarks

## 1.5 Exceptions

pyHaFAS can raise multiple exceptions. In the message of the exception you can always find the complete text error message HaFAS returned.

**exception** pyhafas.types.exceptions.**AccessDeniedError**
Access is denied

**exception** pyhafas.types.exceptions.**AuthenticationError**
Authentiction data missing or wrong

**exception** pyhafas.types.exceptions.**GeneralHafasError**
HaFAS returned an general error

**exception** pyhafas.types.exceptions.**JourneysArrivalDepartureTooNearError**
Journeys search: arrival and departure date are too near

**exception** pyhafas.types.exceptions.**JourneysTooManyTrainsError**
Journeys search: Too many trains, connection is not complete

**exception** pyhafas.types.exceptions.**LocationNotFoundError**
Location/stop not found

**exception** pyhafas.types.exceptions.**NoDepartureArrivalDataError**
No departure/arrival data available

**exception** pyhafas.types.exceptions.**ProductNotAvailableError**
Requested Product is not available in profile

**exception** pyhafas.types.exceptions.**TripDataNotFoundError**
No trips found or trip info not available

## 1.6 Usage Examples

Below you can find usage examples for each method available in *HafasClient*.

### 1.6.1 General Information

In the following code blocks, we only use `departures` but not `arrivals`. Those methods are pretty same, so every time we use `departures` you can exchange this with `arrivals`.

We also only use some of the supported clients. The client can be exchanged, if not specified otherwise.

### 1.6.2 1. locations + departures

The below code gets the departing long-distance trains at the station with the best similarity when searching for "Siegburg/Bonn". Let's get to the parts of the code:

1. The required classes are imported, a `HafasClient` is created with the `DBProfile`
2. **Location-Search**

1. The `HafasClient` searches for locations with the term "Siegburg/Bonn".
2. The best location is chosen from the list (the first object in the list is that with the highest similarity)
3. The `HafasClient` searches for maximum 2 *trips*[G] with the following criteria:

- departing now
- at the best location (from step 2)
- with the products in the categories *long_distance_express* or *long_distance*. *long_distance* is enabled per default and so per default in the list of enabled products and all others (except *long_distance_express*) are disabled.

*long_distance_express is also enabled per default but it can be in the list with True as value to guarantee it's enabled, if it wouldn't be enabled by default, it would be enabled now*

```
import datetime
from typing import List

# Part 1
from pyhafas import HafasClient
from pyhafas.profile import DBProfile
from pyhafas.types.fptf import Leg

client = HafasClient(DBProfile())

# Part 2
locations = client.locations("Siegburg/Bonn")
best_found_location = locations[0]
print(best_found_location)  # <class 'pyhafas.types.fptf.Station'>({'id': '008005556',
↪ 'name': 'Siegburg/Bonn', 'latitude': 50.794051, 'longitude': 7.202616})

# Part 3
departures: List[Leg] = client.departures(
    station=best_found_location.id,
    date=datetime.datetime.now(),
    max_trips=2,
    products={
```

(continues on next page)

(continued from previous page)

```
        'long_distance_express': True,
        'regional_express': False,
        'regional': False,
        'suburban': False,
        'bus': False,
        'ferry': False,
        'subway': False,
        'tram': False,
        'taxi': False
    }
)
print(departures)  # [<class 'pyhafas.types.fptf.Leg'>({...}), <class 'pyhafas.types.
↪fptf.Leg'>({...})]
```

### 1.6.3 2. departures + trip

The below code get the next departing *trip*[G] at the station "Siegburg/Bonn" (with the id *008005556*) and gets after that detailed information with the `trip` method.

Currently, the `trip` method gives the same data as `departures`, but in future versions, there will be more data available in `trip`.

Using the `trip` method is also useful to refresh the data about a specific *trip*[G] by its ID.

```
import datetime

# Part 1
from pyhafas import HafasClient
from pyhafas.profile import DBProfile
from pyhafas.types.fptf import Leg

client = HafasClient(DBProfile())

# Part 2
departure: Leg = client.departures(
    station="008005556",
    date=datetime.datetime.now(),
    max_trips=1
)[0]
print(departure)  # <class 'pyhafas.types.fptf.Leg'>({'id': '1|236759|0|80|26072020',␣
↪...})

# Part 3
trip: Leg = client.trip(departure.id)
print(trip)  # <class 'pyhafas.types.fptf.Leg'>({'id': '1|236759|0|80|26072020', ...})
```

### 1.6.4 3. locations + journeys + journey

In the code block below we create search for possible *journeys*[G] between the stations “Göttingen Bahnhof/ZOB” and “Góttingen Campus” via “Göttingen Angerstraße”.

For an explanation of the first and second part please look at *example 1*. After the code, there is also a visualization of a journey HaFAS returns for this request.

In part 3 the HafasClient searches for *journeys*[G] with the following criteria:

- origin station is “Göttingen Bahnhof/ZOB”
- destination station is “Göttingen Campus”
- the *journey*[G] must be via “Göttingen Angerstraße”
- the *journey*[G] may have a maximum of 1 transfer
- each transfer must have at least a time of 15 minutes

In part 4 the *journey*[G] data of the first *journey*[G] found in part 3 is refreshed.

```
import datetime

# Part 1
from pyhafas import HafasClient
from pyhafas.profile import VSNProfile
from pyhafas.types.fptf import Leg

client = HafasClient(VSNProfile())

# Part 2
location_goe_bf = client.locations("Göttingen Bahnhof/ZOB")[0]
location_goe_ang = client.locations("Göttingen Angerstraße")[0]
location_goe_campus = client.locations("Göttingen Campus")[0]

# Part 3
journeys = client.journeys(
    origin=location_goe_bf,
    via=[location_goe_ang],
    destination=location_goe_campus,
    date=datetime.datetime.now(),
    max_changes=1,
    min_change_time=15
)
print(journeys)  # [<class 'pyhafas.types.fptf.Journey'>({...}), <class 'pyhafas.
↪types.fptf.Journey'>({...}), <class 'pyhafas.types.fptf.Journey'>({...}), ...])]

# Part 4
journey = client.journey(journeys[0].id)

print(journey)  # <class 'pyhafas.types.fptf.Journey'>({...})
```

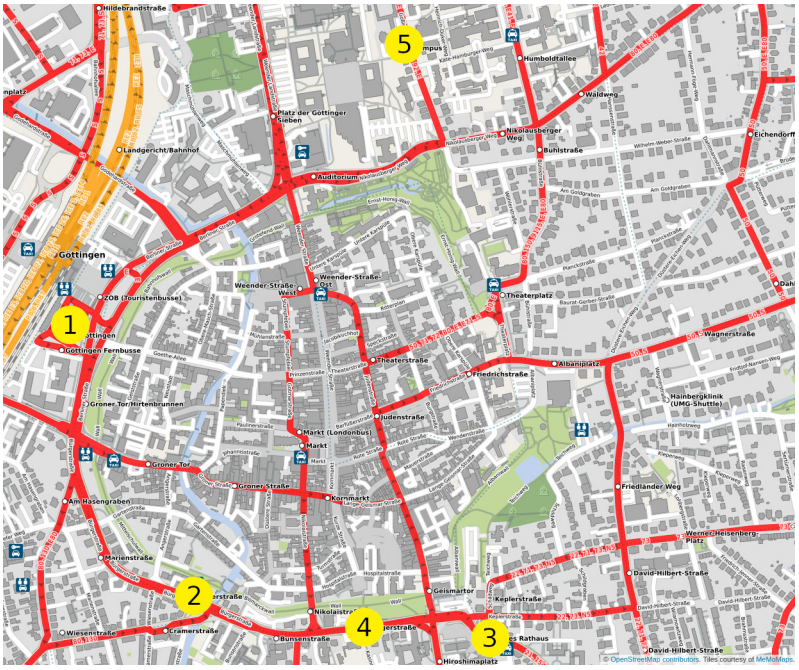
*As short-form for Göttingen, we use GOE*

Here is a table with the *journey*[G] in the variable *journey* of the code example above. Here some explanation on the routing algorithm of HaFAS:

- You might see that the walk leg is exactly 15 minutes. This is because we set a minimum change time of 15 minutes. A normal walking time would be about 5 minutes.

- A walk leg does not count in the number of changes between legs. The maximum number of changes only specifies how many vehicles you change.
- You might think that there's a bug because the via station (GOE Angerstraße, 2) is not in the table below. That's correct. For HaFAS it's enough when a vehicle stops at the via station. In this example, the first and second bus both stops at "GOE Angerstraße".

| origin station | destination station | departure time | arrival time | mode of transport |
|---|---|---|---|---|
| GOE Bahnhof (1) | GOE Neues Rathaus (3) | 11:40 | 11:44 | BUS |
| GOE Neues Rathaus (3) | GOE Bürgerstraße (4) | 11:44 | 11:59 | WALKING |
| GOE Bürgerstraße (4) | GOE Campus (5) | 12:00 | 12:13 | BUS |




map showing the stations, © OpenStreetMap contributors. Tiles courtesy of MeMoMaps.

## 1.7 Introduction

Thank you for to interest in contributing to pyHaFAS. In general, we manage the project via the GitHub issues page. When you are interested in one of these issues, please write a comment, and then we eagerly await your pull request. In special we are happy if you help us with the issues having the help wanted label.

We want to give you an easy start with coding on pyHaFAS, so please read the pages in the category "for developers".

Most of pyHaFAS code is documented with docstrings in the code directly. Most of these docstrings are also rendered in this HTML docs on the page *API*.

If you need help you can always write to us. Good ways for that are on the GitHub Issue or if the question is general, join our matrix-room #pyhafas:matrix.org

### 1.7.1 Testing

Before you submit your pull request please run the tests. They are also automatically run when you create the pull request. You can execute the tests with the following command in the root directory of the project:

```
$ pytest
```

## 1.8 Code Structure

### 1.8.1 Classes and methods

pyHaFAS is built object-orientated with a lot of classes. You may know already the *HafasClient* and the *profile*[G] classes but there a lot more classes for internal use only. For example the *BaseProfile* class consists among others of `BaseRequestHelper`, `BaseJourneyRequest` and very important the *ProfileInterface*

Every class in a *profile*[G] has an *interface* that defines abstract methods that the profile class must implement.

A more detailed view of the construction of a *profile*[G] is given on the page *Profile (Developer)*.

### 1.8.2 File Structure

pyHaFAS's code is split in multiple files. These files are sorted as shown in the structure below.

- **/pyhafas** - base directory of source code
  - **profile** - contains the profiles (every subdirectory should have the same structure as *base*)
    - **interfaces** - contains all abstract classes
    - **base** - Base profile - contains the default handling classes and methods
      - **__init__.py** - contains the Profile
      - **helper** - contains helper functions that are used by multiple requests
      - **mappings** - contains mapping Enum classes
      - **requests** - contains code responsible for requests (there is a file for each request-type containing all methods belonging to it)
    - **PROFILENAME** - contains the files for the profile (only files and directories with changes)
  - **types** - Base directory of all types valid for all profiles
    - **exceptions.py** - exceptions pyHaFAS can raise
    - **fptf.py** - most of the types important for pyHaFAS
    - **hafas_response.py** - contains the `HafasRepsonse` class
  - **client.py** - contains the *HafasClient*

## 1.9 Profile (Developer)

Since every HaFAS deployment is different we have the concept of *profiles*[G] for managing the differences between them.

For a "standard" HaFAS we have the `BaseProfile` which contains code working for most HaFAS deployments. This profile inherits the methods of all classes living in the folder of the profile. That classes all have an interface that defines the abstract methods of the class.

Most methods are instance methods, so they have a *self* parameter. Since the class using the method is the profile we use as type hint for *self* `ProfileInterface`. This interface defines the variables and methods a profile has.

### 1.9.1 How to build a profile?

#### Requirements

- You need to know how to authenticate to the HaFAS deployment. In most cases, you need to know if the authentication is via mic-mac or checksum+salt and you need the salt. The salt can get out of the source code of official apps of the transport provider or you can look if other HaFAS libraries already support the transport provider and get the salt from them
- You need the general requests body. You can get this best with mitmproxy on the official app of the transport provider or of other libraries as in the other requirement

One good source for both requirements is hafas-client in the specific profile folder.

#### Steps

1. Read the API of a profile. See *below*.
2. Have a look at the already existing profiles and the `ProfileInterface` to get the structure of a profile.
3. Create a new product folder with an *__init__.py* file containing the profile class
4. Fill the required variables
5. Add your profile to */pyhafas/profile/__init__.py*
6. Test if everything works
7. Yes? Perfect, add your profile in the documentation and create a pull request! No? Go on with step 8.
8. Log the request pyHaFAS sends and compare this request with the one the official app sends. Is the difference only in one specific request type or in all? If it is only in one go on stop step 8a otherwise step 8b.

a. You could create a new class overwriting the *format*-method of the request. An example where this is done is in the VSNProfile the `format_journey_request()` method in the `pyhafas.profile.vsn.requests.journey.VSNJourneyRequest` class.

b. Please make sure the *requestBody* in your profile is correct. If it is, please *contact* us or try to find the method causing the error yourself.

If you need help with any of that you can *contact* us always!

### API

Here are the minimum required variables of a profile (generated from `ProfileInterface`):

**class** pyhafas.profile.interfaces.**ProfileInterface**

The profile interface is the abstract class of a profile.

It inherits all interfaces, so it has all methods a normal profile has available as abstract methods. Therefore it can be used as type hint for *self* in methods which are inherited by profiles.

**addChecksum: bool**
Whether the checksum authentication method should be activated. Exclusive with *addMicMac*

**addMicMac: bool**
Whether the mic-mac authentication method should be activated. Exclusive with *addChecksum*

**availableProducts: Dict[str, List[int]]**
Should contain all products available in HaFAS. The key is the name the end-user will use, the value is a list of bitmasks (numbers) needed for the product. In most cases, this is only one number. This bitmasks will be add up to generate the final bitmask.

**baseUrl: str**
Complete http(s) URL to mgate.exe of the HaFAS deployment. Other endpoints are (currently) incompatible with pyHaFAS

**defaultProducts: List[str]**
List of products (item must be a key in *availableProducts*) which should be activated by default

**defaultUserAgent: str**
(optional) User-Agent in header when connecting to HaFAS. Defaults to pyhafas. A good option would be the app ones. Can be overwritten by the user.

**locale: str**
(used in future) Locale used for i18n. Should be an IETF Language-Region Tag

Examples: https://tools.ietf.org/html/bcp47#appendix-A

**requestBody: dict**
Static part of the request body sent to HaFAS. Normally contains informations about the client and another authentication

**salt: str**
(required if *addMicMac* or *addChecksum* is true). The salt for calculating the checksum or mic-mac

**timezone: datetime.tzinfo**
Timezone HaFAS lives in. Should be a *pytz timezone* object

**userAgent: str**
(optional) Do not change, unless you know what you're doing. Disallows the user to change the user agent. For usage in internal code to get the user-agent which should be used.

## 1.10 API

### 1.10.1 BaseProfile

**Contents**



For a documentation of the variables, please look at the documentation of *ProfileInterface*

**class** pyhafas.profile.base.**BaseProfile**(*ua=None*)

Bases: *pyhafas.profile.base.helper.request.BaseRequestHelper*, *pyhafas.profile.base.helper.format_products_filter.BaseFormatProductsFilterHelper*, *pyhafas.profile.base.helper.parse_lid.BaseParseLidHelper*, *pyhafas.profile.base.helper.date_time.BaseDateTimeHelper*, *pyhafas.profile.base.helper.parse_leg.BaseParseLegHelper*, pyhafas.profile.base.helper.parse_remark.BaseParseRemarkHelper, *pyhafas.profile.base.requests.location.BaseLocationRequest*, *pyhafas.profile.base.requests.journey.BaseJourneyRequest*, *pyhafas.profile.base.requests.journeys.BaseJourneysRequest*, *pyhafas.profile.base.requests.station_board.BaseStationBoardRequest*, *pyhafas.profile.base.requests.trip.BaseTripRequest*, *pyhafas.profile.interfaces.ProfileInterface*

Profile for a “normal” HaFAS. Only for other profiles usage as basis.

**addChecksum**: **bool** = **False**

**addMicMac**: **bool** = **False**

**availableProducts**: **Dict[str, List[int]]** = **{}**

**baseUrl**: **str** = **''**

**defaultProducts**: **List[str]** = **[]**

**defaultUserAgent**: *str* = 'pyhafas'

**requestBody**: *dict* = {}

**salt**: *str* = ''

### Helper

#### format_products_filter

**class** pyhafas.profile.base.helper.format_products_filter.**BaseFormatProductsFilterHelper**
Bases: *pyhafas.profile.interfaces.helper.format_products_filter.FormatProductsFilterHelperInterface*

Helper for creating the products filter

**format_products_filter**(*requested_products*)
Create the products filter given to HaFAS

**Parameters** **requested_products** (dict) – Mapping of Products to whether it's enabled or disabled

**Return type** dict

**Returns** value for HaFAS "jnyFltrL" attribute

#### date_time

**class** pyhafas.profile.base.helper.date_time.**BaseDateTimeHelper**
Bases: *pyhafas.profile.interfaces.helper.date_time.DateTimeHelperInterface*

**parse_date**(*date_string*)
Parses the date HaFAS returns

**Parameters** **date_string** (str) – Date sent by HaFAS

**Return type** date

**Returns** Parsed date object

**parse_datetime**(*time_string*, *date*)
Parses the time format HaFAS returns and combines it with a date

**Parameters**

- **time_string** (str) – Time string sent by HaFAS (multiple formats are supported. One example: 234000)
- **date** (date) – Start day of the leg/journey

**Return type** datetime

**Returns** Parsed date and time as datetime object

**parse_timedelta**(*time_string*)
Parses the time HaFAS returns as timedelta object

Example use case is when HaFAS returns a duration of a leg :type time_string: str :param time_string: Time string sent by HaFAS (example for format is: 033200) :rtype: timedelta :return: Parsed time as timedelta object

**transform_datetime_parameter_timezone**(*date_time*)
Transfers datetime parameters incoming by the user to the profile timezone

**Parameters** **date_time** (datetime) – datetime parameter incoming by user. Can be timezone aware or unaware

**Return type** datetime

**Returns** Timezone aware datetime object in profile timezone

### parse_leg

**class** pyhafas.profile.base.helper.parse_leg.**BaseParseLegHelper**
Bases: *pyhafas.profile.interfaces.helper.parse_leg.ParseLegHelperInterface*

**parse_leg**(*journey*, *common*, *departure*, *arrival*, *date*, *jny_type='JNY'*, *gis=None*)
Parses Leg HaFAS returns into Leg object

Different Types of HaFAS responses can be parsed into a leg object with the multiple variables

**Parameters**

- **journey** (dict) – Journey object given back by HaFAS (Data of the Leg to parse)
- **common** (dict) – Common object given back by HaFAS
- **departure** (dict) – Departure object given back by HaFAS
- **arrival** (dict) – Arrival object given back by HaFAS
- **date** (date) – Parsed date of Journey (Departing date)
- **jny_type** (str) – HaFAS Journey type
- **gis** – GIS object given back by HaFAS. Currently only used by "WALK" journey type.

**Return type** *Leg*

**Returns** Parsed Leg object

**parse_legs**(*jny*, *common*, *date*)
Parses Legs (when multiple available)

**Parameters**

- **jny** (dict) – Journeys object returned by HaFAS (contains secL list)
- **common** (dict) – Common object returned by HaFAS
- **date** (date) – Parsed date of Journey (Departing date)

**Return type** List[*Leg*]

**Returns** Parsed List of Leg objects

### parse_lid

**class** pyhafas.profile.base.helper.parse_lid.**BaseParseLidHelper**
Bases: *pyhafas.profile.interfaces.helper.parse_lid.ParseLidHelperInterface*

**parse_lid**(*lid*)
Converts the LID given by HaFAS

Splits the LID (e.g. A=1@O=Siegburg/Bonn) in multiple elements (e.g. A=1 and O=Siegburg/Bonn). These are converted into a dict where the part before the equal sign is the key and the part after the value.

**Parameters** **lid** (str) – Location identifier (given by HaFAS)

**Return type** dict

**Returns** Dict of the elements of the dict

**parse_lid_to_station**(*lid*, *name=''*, *latitude=0*, *longitude=0*)
Parses the LID given by HaFAS to a station object

**Parameters**

- **lid** (str) – Location identifier (given by HaFAS)
- **name** (str) – Station name (optional, if not given, LID is used)
- **latitude** (float) – Latitude of the station (optional, if not given, LID is used)
- **longitude** (float) – Longitude of the station (optional, if not given, LID is used)

**Return type** *Station*

**Returns** Parsed LID as station object

### request

**class** pyhafas.profile.base.helper.request.**BaseRequestHelper**
Bases: *pyhafas.profile.interfaces.helper.request.RequestHelperInterface*

**activate_retry**(*retries=4*, *backoff_factor=1*)

**Return type** None

**calculate_checksum**(*data*)
Calculates the checksum of the request (required for most profiles)

**Parameters** **data** (str) – Complete body as string

**Return type** str

**Returns** Checksum for the request

**calculate_mic_mac**(*data*)
Calculates the mic-mac for the request (required for some profiles)

**Parameters** **data** (str) – Complete body as string

**Return type** Tuple[str, str]

**Returns** Mic and mac to be sent to HaFAS

**request**(*body*)
Sends the request and does a basic parsing of the response and error handling

**Parameters** **body** – Reqeust body as dict (without the *requestBody* of the profile)

**Return type** *HafasResponse*

**Returns** HafasRespone object or Exception when HaFAS response returns an error

**request_session = <requests.sessions.Session object>**

**url_formatter**(*data*)
Formats the URL for HaFAS (adds the checksum or mic-mac)

**Parameters** **data** (str) – Complete body as string

**Return type** str

**Returns** Request-URL (maybe with checksum or mic-mac)

### Mappings

#### error_codes

**class** pyhafas.profile.base.mappings.error_codes.**BaseErrorCodesMapping**(*value*)
Bases: *pyhafas.profile.interfaces.mappings.error_codes.ErrorCodesMappingInterface*

Mapping of the HaFAS error code to the exception class

*default* defines the error when the error code cannot be found in the mapping

**AUTH = <class 'pyhafas.types.exceptions.AuthenticationError'>**

**H500 = <class 'pyhafas.types.exceptions.JourneysTooManyTrainsError'>**

**H890 = <class 'pyhafas.types.exceptions.JourneysArrivalDepartureTooNearError'>**

**LOCATION = <class 'pyhafas.types.exceptions.LocationNotFoundError'>**

**R5000 = <class 'pyhafas.types.exceptions.AccessDeniedError'>**

**SQ005 = <class 'pyhafas.types.exceptions.TripDataNotFoundError'>**

**TI001 = <class 'pyhafas.types.exceptions.TripDataNotFoundError'>**

**default: Exception = <class 'pyhafas.types.exceptions.GeneralHafasError'>**

### Requests

#### journey

**class** pyhafas.profile.base.requests.journey.**BaseJourneyRequest**
Bases: *pyhafas.profile.interfaces.requests.journey.JourneyRequestInterface*

**format_journey_request**(*journey*)
Creates the HaFAS request body for a journey request

**Parameters** **journey** (*Journey*) – Id of the journey (ctxRecon)

**Return type** dict

**Returns** Request body for HaFAS

**parse_journey_request**(*data*)
Parses the HaFAS response for a journey request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** *Journey*

**Returns** List of Journey objects

### journeys

**class** pyhafas.profile.base.requests.journeys.**BaseJourneysRequest**
Bases: *pyhafas.profile.interfaces.requests.journeys.JourneysRequestInterface*

**format_journeys_request**(*origin*, *destination*, *via*, *date*, *min_change_time*, *max_changes*, *products*, *max_journeys*)
Creates the HaFAS request body for a journeys request

**Parameters**

- **origin** (*Station*) – Origin station
- **destination** (*Station*) – Destionation station
- **via** (List[*Station*]) – Via stations, maybe empty list)
- **date** (datetime) – Date and time to search journeys for
- **min_change_time** (int) – Minimum transfer/change time at each station
- **max_changes** (int) – Maximum number of changes
- **products** (Dict[str, bool]) – Allowed products (a product is a mean of transport like ICE,IC)
- **max_journeys** (int) – Maximum number of returned journeys

**Return type** dict

**Returns** Request body for HaFAS

**format_search_from_leg_request**(*origin*, *destination*, *via*, *min_change_time*, *max_changes*, *products*)
Creates the HaFAS request body for a journeys request

**Parameters**

- **origin** (*Leg*) – Origin leg
- **destination** (*Station*) – Destionation station
- **via** (List[*Station*]) – Via stations, maybe empty list)
- **min_change_time** (int) – Minimum transfer/change time at each station
- **max_changes** (int) – Maximum number of changes
- **products** (Dict[str, bool]) – Allowed products (a product is a mean of transport like ICE,IC)

**Return type** dict

**Returns** Request body for HaFAS

**parse_journeys_request**(*data*)
Parses the HaFAS response for a journeys request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** List[*Journey*]

**Returns** List of Journey objects

### location

**class** pyhafas.profile.base.requests.location.**BaseLocationRequest**
Bases: *pyhafas.profile.interfaces.requests.location.LocationRequestInterface*

**format_location_request**(*term*, *rtype='S'*)
Creates the HaFAS request body for a location search request.

**Parameters**

- **term** (str) – Search term
- **type** – Result types. One of ['S' for stations, 'ALL' for addresses and stations]

**Returns** Request body for HaFAS

**parse_location_request**(*data*)
Parses the HaFAS response for a location request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** List[*Station*]

**Returns** List of Station objects

### station_board

**class** pyhafas.profile.base.requests.station_board.**BaseStationBoardRequest**
Bases: *pyhafas.profile.interfaces.requests.station_board.StationBoardRequestInterface*

**format_station_board_request**(*station*, *request_type*, *date*, *max_trips*, *duration*, *products*, *direction*)
Creates the HaFAS request for a station board request (departure/arrival)

**Parameters**

- **station** (*Station*) – Station to get departures/arrivals for
- **request_type** (*StationBoardRequestType*) – ARRIVAL or DEPARTURE
- **date** (datetime) – Date and time to get departures/arrival for
- **max_trips** (int) – Maximum number of trips that can be returned
- **products** (Dict[str, bool]) – Allowed products (e.g. ICE,IC)
- **duration** (int) – Time in which trips are searched
- **direction** (Optional[*Station*]) – Direction (end) station of the train. If none, filter will not be applied

**Return type** dict

**Returns** Request body for HaFAS

**parse_station_board_request**(*data*, *departure_arrival_prefix*)
Parses the HaFAS data for a station board request

**Parameters**

- **data** (*HafasResponse*) – Formatted HaFAS response
- **departure_arrival_prefix** (str) – Prefix for specifying whether its for arrival or departure (either a for arrival or d for departure)

**Return type** List[*StationBoardLeg*]

**Returns** List of StationBoardLeg objects

#### trip

**class** pyhafas.profile.base.requests.trip.**BaseTripRequest**

Bases: *pyhafas.profile.interfaces.requests.trip.TripRequestInterface*

**format_trip_request**(*trip_id*)

Creates the HaFAS request for a trip request

**Parameters** **trip_id** (str) – Id of the trip/leg

**Return type** dict

**Returns** Request body for HaFAS

**parse_trip_request**(*data*)

Parses the HaFAS data for a trip request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** *Leg*

**Returns** Leg objects

### 1.10.2 DBProfile

**Contents**

- *DBProfile*

For a documentation of the variables, please look at the documentation of *ProfileInterface*

**class** pyhafas.profile.db.**DBProfile**(*ua=None*)

Bases: *pyhafas.profile.base.BaseProfile*

Profile of the HaFAS of Deutsche Bahn (DB) - German Railway - Regional and long-distance trains throughout Germany

**addChecksum: bool = True**

**availableProducts: Dict[str, List[int]] = {'bus': [32], 'ferry': [64], 'long_distan**

**baseUrl: str = 'https://reiseauskunft.bahn.de/bin/mgate.exe'**

**defaultProducts: List[str] = ['long_distance_express', 'long_distance', 'regional_exp**

**defaultUserAgent: str = 'DB Navigator/19.10.04 (iPhone; iOS 13.1.2; Scale/2.00)'**

**locale: str = 'de-DE'**

**requestBody: dict = {'auth': {'aid': 'n91dB8Z77MLdoR0K', 'type': 'AID'}, 'client':**

**salt: str = 'bdI8UVj40K5fvxwf'**

**timezone:** *datetime.tzinfo* = <DstTzInfo 'Europe/Berlin' LMT+0:53:00 STD>

### 1.10.3 ProfileInterface

**Contents**



**class** pyhafas.profile.interfaces.**ProfileInterface**

Bases: *pyhafas.profile.interfaces.helper.request.RequestHelperInterface*, *pyhafas.profile.interfaces.helper.format_products_filter.FormatProductsFilterHelperInterface*, *pyhafas.profile.interfaces.helper.parse_lid.ParseLidHelperInterface*, *pyhafas.profile.interfaces.helper.date_time.DateTimeHelperInterface*, *pyhafas.profile.interfaces.helper.parse_leg.ParseLegHelperInterface*, pyhafas.profile.interfaces.helper.parse_remark.ParseRemarkHelperInterface, *pyhafas.profile.interfaces.requests.location.LocationRequestInterface*, *pyhafas.profile.interfaces.requests.journey.JourneyRequestInterface*, *pyhafas.profile.interfaces.requests.journeys.JourneysRequestInterface*, *pyhafas.profile.interfaces.requests.station_board.StationBoardRequestInterface*, *pyhafas.profile.interfaces.requests.trip.TripRequestInterface*, abc.ABC

The profile interface is the abstract class of a profile.

It inherits all interfaces, so it has all methods a normal profile has available as abstract methods. Therefore it can be used as type hint for *self* in methods which are inherited by profiles.

**addChecksum:** *bool*
Whether the checksum authentication method should be activated. Exclusive with *addMicMac*

**addMicMac:** *bool*
Whether the mic-mac authentication method should be activated. Exclusive with *addChecksum*

**availableProducts: Dict[str, List[int]]**
Should contain all products available in HaFAS. The key is the name the end-user will use, the value is a list of bitmasks (numbers) needed for the product. In most cases, this is only one number. This bitmasks will be add up to generate the final bitmask.

**baseUrl: str**
Complete http(s) URL to mgate.exe of the HaFAS deployment. Other endpoints are (currently) incompatible with pyHaFAS

**defaultProducts: List[str]**
List of products (item must be a key in *availableProducts*) which should be activated by default

**defaultUserAgent: str**
(optional) User-Agent in header when connecting to HaFAS. Defaults to pyhafas. A good option would be the app ones. Can be overwritten by the user.

**locale: str**
(used in future) Locale used for i18n. Should be an IETF Language-Region Tag

Examples: https://tools.ietf.org/html/bcp47#appendix-A

**requestBody: dict**
Static part of the request body sent to HaFAS. Normally contains informations about the client and another authentication

**salt: str**
(required if *addMicMac* or *addChecksum* is true). The salt for calculating the checksum or mic-mac

**timezone: datetime.tzinfo**
Timezone HaFAS lives in. Should be a *pytz timezone* object

**userAgent: str**
(optional) Do not change, unless you know what you're doing. Disallows the user to change the user agent. For usage in internal code to get the user-agent which should be used.

### Helper

#### format_products_filter

**class** pyhafas.profile.interfaces.helper.format_products_filter.**FormatProductsFilterHelperI**
Bases: abc.ABC

**abstract format_products_filter**(*requested_products*)
Create the products filter given to HaFAS

**Parameters** **requested_products** (dict) – Mapping of Products to whether it's enabled or disabled

**Return type** dict

**Returns** value for HaFAS "jnyFltrL" attribute

### date_time

**class** pyhafas.profile.interfaces.helper.date_time.**DateTimeHelperInterface**
Bases: abc.ABC

**abstract parse_date**(*date_string*)
Parses the date HaFAS returns

**Parameters** **date_string** (str) – Date returned from HaFAS

**Return type** date

**Returns** Parsed date object

**abstract parse_datetime**(*time_string*, *date*)
Parses the time format HaFAS returns and combines it with a date

**Parameters**

- **time_string** (str) – Time string sent by HaFAS
- **date** (date) – Start day of the leg/journey

**Return type** datetime

**Returns** Parsed date and time as datetime object

**abstract parse_timedelta**(*time_string*)
Parses the time HaFAS returns as timedelta object

Example use case is when HaFAS returns a duration of a leg :type time_string: str :param time_string: Time string sent by HaFAS :rtype: timedelta :return: Parsed time as timedelta object

**abstract transform_datetime_parameter_timezone**(*date_time*)
Transfers datetime parameters incoming by the user to the profile timezone

**Parameters** **date_time** (datetime) – datetime parameter incoming by user. Can be timezone aware or unaware

**Return type** datetime

**Returns** Timezone aware datetime object in profile timezone

### parse_leg

**class** pyhafas.profile.interfaces.helper.parse_leg.**ParseLegHelperInterface**
Bases: abc.ABC

**abstract parse_leg**(*journey*, *common*, *departure*, *arrival*, *date*, *jny_type='JNY'*, *gis=None*)
Parses Leg HaFAS returns into Leg object

**Parameters**

- **journey** (dict) – Journey object given back by HaFAS (Data of the Leg to parse)
- **common** (dict) – Common object given back by HaFAS
- **departure** (dict) – Departure object given back by HaFAS
- **arrival** (dict) – Arrival object given back by HaFAS
- **date** (date) – Parsed date of Journey (Departing date)
- **jny_type** (str) – HaFAS Journey type

- **gis** – GIS object given back by HaFAS.

**Return type** *Leg*

**Returns** Parsed Leg object

**abstract parse_legs**(*jny*, *common*, *date*)
Parses Legs (when multiple available)

**Parameters**

- **jny** (dict) – Journeys object returned by HaFAS
- **common** (dict) – Common object returned by HaFAS
- **date** (date) – Parsed date of Journey (Departing date)

**Return type** List[*Leg*]

**Returns** Parsed List of Leg objects

### parse_lid

**class** pyhafas.profile.interfaces.helper.parse_lid.**ParseLidHelperInterface**
Bases: abc.ABC

**abstract parse_lid**(*lid*)
Converts the LID given by HaFAS. Splits the LID in multiple elements

**Parameters** **lid** (str) – Location identifier (given by HaFAS)

**Return type** dict

**Returns** Dict of the elements of the dict

**abstract parse_lid_to_station**(*lid*, *name=''*, *latitude=0*, *longitude=0*)
Parses the LID given by HaFAS to a station object

**Parameters**

- **lid** (str) – Location identifier (given by HaFAS)
- **name** (str) – Station name (optional)
- **latitude** (float) – Latitude of the station (optional)
- **longitude** (float) – Longitude of the station (optional)

**Return type** *Station*

**Returns** Parsed LID as station object

### request

**class** pyhafas.profile.interfaces.helper.request.**RequestHelperInterface**
Bases: abc.ABC

**abstract calculate_checksum**(*data*)
Calculates the checksum of the request (required for most profiles)

**Parameters** **data** (str) – Complete body as string

**Return type** str

**Returns** Checksum for the request

**abstract calculate_mic_mac**(*data*)
Calculates the mic-mac for the request (required for some profiles)

**Parameters** **data** (str) – Complete body as string

**Return type** Tuple[str, str]

**Returns** Mic and mac to be sent to HaFAS

**abstract request**(*body*)
Sends the request and does a basic parsing of the response and error handling

**Parameters** **body** – Reqeust body as dict (without the *requestBody* of the profile)

**Return type** *HafasResponse*

**Returns** HafasRespone object or Exception when HaFAS response returns an error

**abstract url_formatter**(*data*)
Formats the URL for HaFAS (adds the checksum or mic-mac)

**Parameters** **data** (str) – Complete body as string

**Return type** str

**Returns** Request-URL (maybe with checksum or mic-mac)

### Mappings

#### error_codes

**class** pyhafas.profile.interfaces.mappings.error_codes.**ErrorCodesMappingInterface**(*value*)
Bases: enum.Enum

Mapping of the HaFAS error code to the exception class

*default* defines the error when the error code cannot be found in the mapping

**default: Exception**

### Requests

#### journey

**class** pyhafas.profile.interfaces.requests.journey.**JourneyRequestInterface**
Bases: abc.ABC

**abstract format_journey_request**(*journey*)
Creates the HaFAS request body for a journey request

**Parameters** **journey** (*Journey*) – Id of the journey (ctxRecon)

**Return type** dict

**Returns** Request body for HaFAS

**abstract parse_journey_request**(*data*)
Parses the HaFAS response for a journey request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** *Journey*

**Returns** List of Journey objects

### journeys

**class** pyhafas.profile.interfaces.requests.journeys.**JourneysRequestInterface**
Bases: abc.ABC

**abstract format_journeys_request**(*origin*, *destination*, *via*, *date*, *min_change_time*, *max_changes*, *products*, *max_journeys*)
Creates the HaFAS request body for a journeys request

**Parameters**

- **origin** (*Station*) – Origin station
- **destination** (*Station*) – Destionation station
- **via** (List[*Station*]) – Via stations, maybe empty list)
- **date** (datetime) – Date and time to search journeys for
- **min_change_time** (int) – Minimum transfer/change time at each station
- **max_changes** (int) – Maximum number of changes
- **products** (Dict[str, bool]) – Allowed products (a product is a mean of transport like ICE,IC)
- **max_journeys** (int) – Maximum number of returned journeys

**Return type** dict

**Returns** Request body for HaFAS

**abstract format_search_from_leg_request**(*origin*, *destination*, *via*, *min_change_time*, *max_changes*, *products*)
Creates the HaFAS request body for a search from leg request

**Parameters**

- **origin** (*Leg*) – Origin leg
- **destination** (*Station*) – Destionation station
- **via** (List[*Station*]) – Via stations, maybe empty list)
- **min_change_time** (int) – Minimum transfer/change time at each station
- **max_changes** (int) – Maximum number of changes
- **products** (Dict[str, bool]) – Allowed products (a product is a mean of transport like ICE,IC)

**Return type** dict

**Returns** Request body for HaFAS

**abstract parse_journeys_request**(*data*)
Parses the HaFAS response for a journeys request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** List[*Journey*]

**Returns** List of Journey objects

### location

**class** pyhafas.profile.interfaces.requests.location.**LocationRequestInterface**
Bases: abc.ABC

**abstract format_location_request**(*term*, *rtype='S'*)
Creates the HaFAS request body for a location search request.

**Parameters**

- **term** (str) – Search term
- **rtype** (str) – Result types. One of ['S' for stations, 'ALL' for addresses and stations]

**Returns** Request body for HaFAS

**abstract parse_location_request**(*data*)
Parses the HaFAS response for a location request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** List[*Station*]

**Returns** List of Station objects

### station_board

**class** pyhafas.profile.interfaces.requests.station_board.**StationBoardRequestInterface**
Bases: abc.ABC

**format_station_board_request**(*station*, *request_type*, *date*, *max_trips*, *duration*, *products*, *direction*)
Creates the HaFAS request for Station Board (departure/arrival)

**Parameters**

- **station** (*Station*) – Station to get departures/arrivals for
- **request_type** (*StationBoardRequestType*) – ARRIVAL or DEPARTURE
- **date** (datetime) – Date and time to get departures/arrival for
- **max_trips** (int) – Maximum number of trips that can be returned
- **products** (Dict[str, bool]) – Allowed products (a product is a mean of transport like ICE,IC)
- **duration** (int) – Time in which trips are searched
- **direction** (Optional[*Station*]) – Direction (end) station of the train. If none, filter will not be applied

**Return type** dict

**Returns** Request body for HaFAS

**parse_station_board_request**(*data*, *departure_arrival_prefix*)
Parses the HaFAS data for a station board request

**Parameters**

- **data** (*HafasResponse*) – Formatted HaFAS response
- **departure_arrival_prefix** (str) – Prefix for specifying whether its for arrival or departure

**Return type** List[*StationBoardLeg*]

**Returns** List of StationBoardLeg objects

#### trip

**class** pyhafas.profile.interfaces.requests.trip.**TripRequestInterface**

Bases: abc.ABC

**abstract format_trip_request**(*trip_id*)

Creates the HaFAS request for a trip request

**Parameters** **trip_id** (str) – Id of the trip/leg

**Return type** dict

**Returns** Request body for HaFAS

**abstract parse_trip_request**(*data*)

Parses the HaFAS data for a trip request

**Parameters** **data** (*HafasResponse*) – Formatted HaFAS response

**Return type** *Leg*

**Returns** Leg objects

### 1.10.4 VSNProfile

**Contents**

- *VSNProfile*

For a documentation of the variables, please look at the documentation of *ProfileInterface*

**class** pyhafas.profile.vsn.**VSNProfile**(*ua=None*)

Bases: pyhafas.profile.vsn.requests.journey.VSNJourneyRequest, *pyhafas.profile.base.BaseProfile*

Profile for the HaFAS of "Verkehrsverbund Süd-Niedersachsen" (VSN) - local transportation provider

**addMicMac: bool = True**

**availableProducts: Dict[str, List[int]] = {'anruf_sammel_taxi': [512], 'bus': [32],**

**baseUrl: str = 'https://fahrplaner.vsninfo.de/hafas/mgate.exe'**

**defaultProducts: List[str] = ['long_distance_express', 'long_distance', 'regional_exp**

**defaultUserAgent: str = 'vsn/5.3.1 (iPad; iOS 13.3; Scale/2.00)'**

**locale: str = 'de-DE'**

**requestBody: dict = {'auth': {'aid': 'Mpf5UPC0DmzV8jkg', 'type': 'AID'}, 'client':**

**salt: str = 'SP31mBufSyCLmNxp'**

**timezone: datetime.tzinfo = <DstTzInfo 'Europe/Berlin' LMT+0:53:00 STD>**

### 1.10.5 HafasResponse

**class** pyhafas.types.hafas_response.**HafasResponse**(*raw_hafas_response*, *mapping*)
Bases: object

The class HafasResponse handles the general parsing and error-checking of a raw HaFAS response

**Variables**

- **raw_hafas_response** (*requests.Response*) – The raw response of HaFAS
- **data** (*dict*) – json parsed raw response of HaFAS

**check_for_errors**(*mapping*)
Checks if HaFAS response has error messages and handles them

**Parameters** **mapping** (*ErrorCodesMappingInterface*) – Error Mapping Enum (key is the HaFAS error code, value the error class)

**property common**
Returns the "common" data out of HaFAS data

**Returns** dict with "common" data

**property res**
Returns the "res" data out of HaFAS data

**Returns** dict with "res" data

### 1.10.6 StationBoardRequestType

**class** pyhafas.types.station_board_request.**StationBoardRequestType**(*value*)
Bases: enum.Enum

Mapping of StationBoard request from client to HaFAS

**ARRIVAL = 'ARR'**

**DEPARTURE = 'DEP'**

## 1.11 Glossary

**profile** Customization for each HaFAS deployment - Contains the endpoint, tokens and possible changes for the deployment

**FPTF** Abbreviation for Friendly Public Transport Format - Used as the basis for returned data

**Station Board** Generalization for *arrivals* and *departures* requests

**product** A product is the generalization of all means of transport. When this term is used, all types of transport are meant (e.g. busses, regional trains, ferries).

**journey** A journey is a computed set of directions to get from A to B at a specific time. It would typically be the result of a route planning algorithm.

**leg** A leg or also named trip is most times part of a journey and defines a journey with only one specific vehicle from A to B.

## 1.12 Changelog

**Contents**



### 1.12.1 v0.4.0

- [feature] Parse leg remarks
- [feature] Add KVB Profile
- [feature] Add optional retry parameter for requests. You might activate retries with the *activate_retry()* function on your profile

### 1.12.2 v0.3.1

- [BUG] Fix setting of default user agent

### 1.12.3 v0.3.0

- [FEATURE] Add timezone awareness to all datetime time objects
- [FEATURE] Allow filtering at `pyhafas.client.HafasClient.departures()` and `pyhafas.client.HafasClient.arrivals()` for direction station
- [FEATURE] Add option to `pyhafas.client.HafasClient.journeys()` request to allow setting a maximum number of returned journeys
- [BUG] Fix bug with some HaFAS versions in platform parsing
- [BUG] Changed coordinate type from int to float
- Better dependency requirements (less specific versions)
- Add tests

### 1.12.4 v0.2.0

- [BREAKING] Changed return format of *pyhafas.client.HafasClient.arrivals()* and *pyhafas.client.HafasClient.departures()* methods
- [BREAKING] Removed deprecated parameter *max_journeys* in *pyhafas.client.HafasClient.arrivals()* and *pyhafas.client.HafasClient.departures()* methods
- [BUG] Fixed *pyhafas.client.HafasClient.journey()* request in VSN-Profile

# PYTHON MODULE INDEX

## p

# INDEX

## J



## L



## M



## N



## P

## R



## S



## T



## U



## V



## W