YETI - Yet Another Emissions From Traffic Inventory Documentation

Release 1.0.0

Tom Wollnik, Joana Leitao, Tim Butler

Aug 30, 2019

User Documentation

YETI is a tool for street level bottom-up traffic emission calculation. It helps you create high-resolution traffic emission inventories.

YETI supports common emission calculation methodologies like COPERT or HBEFA. The methodologies are implemented as *Strategies*. You can select which Strategy to use for a model run in the *configuration file*.

YETI was originally built for the City of Berlin, but is flexible enough to be adopted to different datasets. Find out how to customize YETI for your needs in the developer section. Start here: *How does it work?*

The code for YETI can be found on GitHub.

CHAPTER 1

Installation and Setup

1.1 1. Make sure your Python version is supported

This project requires Python 3.6 or above. You can find our your Python version by running `python --version` on the command line. If your Python version is below 3.6, please upgrade to a newer version.

Note that YETI is tested for Python 3.6 and 3.7. However it should also work with newer Python versions. When in doubt run the tests on your computer. If they pass you are good to go.

1.2 2. Clone the GitHub repository

Clone the GitHub repositiory by running `git clone https://github.com/twollnik/YETI.git` on the command line. You need to have git installed for this step. If you don't have git, get it here.

These directories will be downloaded: `code`, `diagrams`, `docs`, `example`, and `tests`.

1.3 3. Install the necessary packages

Install dependencies with pip by running `pip install -r requirements.txt` on the command line from the repository root directory. If you want to do development work you should also install dev dependencies: `pip install -r requirements-dev.txt`.

CHAPTER 2

Demo

We have included example configuration files and example data for you to try out. You can find the example files in the folder `example/`. To run the demo, execute the following command on the command line from the repository root directory: `python -m run_yeti -c example/example_configs/copert_hot_config.yaml`. Instead of the `copert_hot_config.yaml` you can use any of the config files in `example/example_configs/`
.

CHAPTER 3

Usage

3.1 Run the model

All interactions with YETI use the script `run_yeti.py`. Run the script on the command line: `python -m run_yeti`. Make sure to run the script from the repository root directory.

`run_yeti.py` uses a configuration file in YAML format where a Strategy for the emission calculation method is defined together with all the necessary input/output file locations and other parameters.

You may specify the location of the config file: `python -m run_yeti -c path/to/config.yaml`. If you don't specify a location for the config file explicitly, the path `./config.yaml` is used. Look here for more detailed information what should be included in the config file.

You can pass the argument `-q` to run YETI in quiet mode: `python -m run_yeti -q`. In quiet mode no DEBUG information will be displayed.

Run `python -m run_yeti --help` for short usage information.

Output of a model run are one or multiple emissions csv files and a file `run_info.txt`. All output files will be in the `output_folder` that you specify in the configuration file.

3.2 Run the tests

We include Python unit tests to test most of the YETI code. If you modified the code and want to see if it still works, you may want to execute the tests. Note that the tests are also run on our test server (Travis CI)automatically every time someone pushes to the GitHub repository.

Execute the tests by running `make test` on the command line from the repository root directory. Note that GNU Make needs to be installed on your computer for this to work. If you don't have GNU Make installed, you can run the tests with `python -m unittest tests/*/test*.py tests/test*.py`.

CHAPTER 4

Configuring YETI

The configuration arguments for a YETI run are stored in a YAML file. The configuration file (sometimes referred to as "config file" or "config.yaml") contains all the configurations for a model run.

What goes in the configuration file depends on the Strategy used for the run. You can find example configuration files for most strategies in the folder `example/example_configs/`. We encourage you to adapt the example configuration files to your needs instead of writing your own configuration files from scratch.

A YETI config file contains the following:

4.1 Mode

You need to specify a mode. The mode expresses which *data format* you would like to use for the run. Add one of the following lines to your configuration file:

```
mode:             berlin_format
```

if you are looking to use `berlin_format` data or

```
mode:           yeti_format
```

if you want to use `yeti_format` data.

4.2 Pollutants

The list of pollutants you want to calculate emissions for. Add this line to your configuration file:

```
pollutants: [PollutantType.{pollutant1}, PollutantType.{pollutant2}, ...]
```

For example:

```
pollutants: [PollutantType.NOx]
```

Note that different Strategies support different pollutants. Find out what pollutants a Strategy supports on the Strategy's documentation page.

4.3 Strategy and functions

Specify a *Strategy* in the configuration file: `strategy:  path.to.a.Strategy`

Along with the Strategy you need to specify functions to work with the data required for the Strategy. Possible functions are:

- `load_berlin_format_data_function`: The path to a function that converts data in `berlin_format` to data in `yeti_format` and saves the constructed `yeti_format` data to disc. This argument is only necessary for the `mode` `berlin_format`.
- `load_yeti_format_data_function`: The path to a function that loads data in `yeti_format` from disc into memory.
- `validation_function`: Optional. The path to a function that validates the given data. Validation includes: Check that all necessary files are present, Check the column names, check that categorical columns use the right categories, check mappings between files, and check that percentage columns contain values between 0 and 1. Which validation function you want to use depends on the `mode`. Select a `validation_function` that fits the data format you are working with. If you don't specify a `validation_function` in the configuration file, validation is skipped.

Take a look at the docs page for the strategy you want to use to see which functions you should specify in the configuration file or consult the example configs in `example/example_configs/`.

Example:

```
strategy:                       code.copert_cold_strategy.CopertColdStrategy.
↪CopertColdStrategy
load_berlin_format_data_function:    code.copert_cold_strategy.load_berlin_format_
↪data.load_copert_cold_berlin_format_data
load_yeti_format_data_function:   code.copert_cold_strategy.load_yeti_format_data.load_
↪copert_cold_yeti_format_data
validation_function:            code.copert_hot_strategy.validate.file_paths_are_valid
```

4.4 Filenames

This section of the configuration file contains paths to the files you intend to use for the model run. What files are required depends on the Strategy used for the run and on the `mode`. Find out more on the Strategy's documentation page.

4.5 Output folder(s)

You need to specify an `output_folder`. The Model output will be saved in this folder.

You may also specify an `output_folder_for_yeti_format_data`. If the `mode` is `berlin_format`, the `yeti_format` files generated by YETI will be saved in the `output_folder_for_yeti_format_data`. If no `output_folder_for_yeti_format_data` is given, the `yeti_format` files will be saved in the `output_folder`. If you are using `mode` `yeti_format`, this argument is ignored.

Example:

```
output_folder:                        emission_output/
output_folder_for_yeti_format_data:   yeti_format_data_new/
```

4.6 Optional config arguments

links_to_use You may specify a list of `LinkID` s that should be used for this run. All links with IDs that are not in the given list will be ignored. Example:

```
links_to_use:    [42_123, 64_586]    # 42_123 and 64_586 are two link IDs from the link␣
↪data
```

use_n_traffic_data_rows An integer that specifies how many rows of the traffic data should be used for the run. This config option is particularly useful for quick test runs. Example:

```
use_n_traffic_data_rows:     100
```

4.7 Strategy-specific config arguments

Each Strategy comes with a set of required and optional configuration arguments. For example the `CopertColdStrategy` requires that a `temperature` is specified in the config.

You can find out which configuration arguments are needed or possible for each Strategy on the Strategy's documentation page.

CHAPTER 5

Data Requirements

YETI is a street level model. This means that the road network you want to calculate emissions for needs to be divided into street links.

Find example datasets in `example/example_berlin_format_data` and `example/example_yeti_format_data`.

5.1 The two data classes

We differentiate between `berlin_format` and `yeti_format`.

`berlin_format` is data in the format that we were using at the start of this project. It is not ideal for the calculations and needs to be transformed to a different format more suitable for the emissions calculation.

`yeti_format` is data in a unified format. It defines a layer of abstraction between the `berlin_format` data and the emission calculation. We provide functions to transform `berlin_format` data to `yeti_format` data for all Strategies.

The data that you are working with is likely in a different format than our `berlin_format`, however chances are that you can tranform your data to fit the `yeti_format`. If this is the case, you only need to write a function to convert your data to `yeti_format`. Once this is done you can use YETI with your data and don't need to adapt any other part of the system.

5.2 Data requirements depend on Strategy

The data requirements depend on how you want to calculate emissions. For example calculating emissions with the COPERT methodology requires different input data than a calculation with the HBEFA methodology.

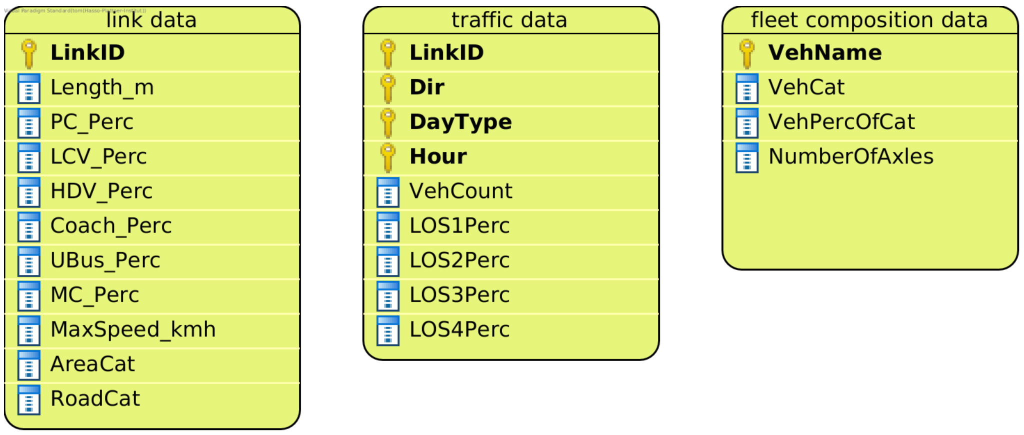
Take a look at the docs page of the Strategy you want to use to find out about the data requirements for that Strategy.

5.3 File format

All data files are csv files. They use comma (' , ') as seperator and the dot (' . ') for decimal points.

5.4 berlin_format

All Strategies need the following three files in `berlin_format`:



how-to-read-er

link data

This is a file with data about the street links in the region you are examining. Each street link corresponds to one line in the link data file. These are the columns:

- *LinkID*: The unique ID for the street link
- *Length_m*: The length of the link in meters
- *AreaCat*: The area category for the street link. Currently allowed area categories are: `0` for rural and `1` for urban. If your input data uses different categories, you can *change the data loading behaviour* to support other categories.
- *RoadCat*: The road category for the street link. Currently allowed area categories are listed below. If your input data uses different categories, you can *change the data loading behaviour* to support other categories.

```
6 = Motorway-Nat.
5 = Motorway-City
9 = Trunk road / Primary-Nat.
8 = Trunk road / Primary-City
1 = Distributer / Secondary
3 = Loacal / Collector
0 = Access-residential
```

- *MaxSpeed_kmh*: The legal maximum speed for the street link in km/h
- *PC_Perc*: The percentage of passenger cars of total traffic at the link
- *LCV_Perc*: The percentage of light commercial vehicles of total traffic at the link
- *HDV_Perc*: The percentage of heavy duty vehicles of total traffic at the link
- *Coach_Perc*: The percentage of coaches of total traffic at the link

- *UBus_Perc*: The percentage of urban buses of total traffic at the link
- *MC_Perc*: The percentage of motorcycles and mopeds of total traffic at the link

Example:

LinkID	Area-Cat	Road-Cat	MaxSpeed_kmh	PC_Perc	LCV_Perc	HDV_Perc	Coach_Perc	UBus_Perc	MC_Perc
42_123	0	5	100	0.4	0.2	0.05	0.15	0.1	0.1
65_485	1	3	30	0.8	0.05	0	0	0.1	0.05

---

traffic data

This file contains the total vehicle count at for each street link, direction, day type and hour. It also contains the distribution of traffic across the different levels of service. It has these columns:

- *LinkID*: The unique ID for the street link. The LinkIDs need to match the LinkIDs in the link data.
- *Dir*: The traffic direction for the given street link:

```
0 = left
1 = right
```

- *DayType*: One of four day types:

```
1 = Monday to Thursday
2 = Friday
3 = Saturday
7 = Sunday or Holiday
```

- *Hour*: A number between 0 and 23.
- *VehCount*: The total number of cars at the given street link for the given day type, hour and direction.
- *LOSxPerc*: The percentage of traffic attributed to the x level of service. Currently implemented levels of service: 1 (Freeflow), 2 (Heavy), 3 (Satur.), and 4 (St+Go).

Example:

LinkID	Dir	Day-Type	Hour	Ve-hCount	LOS1Percentage	LOS2Percentage	LOS3Percentage	LOS4Percentage
42_123	0	1	0	4	1	0	0	0
65_485	1	7	9	80	0.2	0.5	0.4	0.1

---

fleet composition data

This file contains data on the composition of the fleet. The fleet composition applies to all links. It contains these columns:

- *VehName*: The name of a vehicle class. For example `PC petrol <1.4L Euro-1`.
- *VehCat*: The category of the vehicle. Possible vehicle categories are:

```
- P = passenger cars
- L = light commercial vehicles
- S = heavy duty vehicles
- R = coaches
- B = urban buses
- Moped = mopeds
- M = motorcycles
```

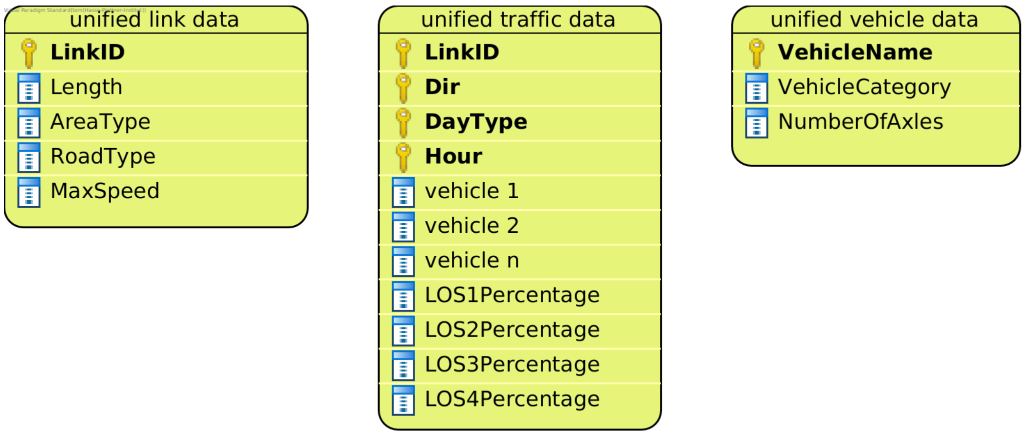
- *VehPercOfCat*: The percentage of the vehicle with regard to all vehicles of its category. Should be between 0 and 1.
- *NumberOfAxles*: The number of axles for the given vehicle. This is only relevant for vehicles belonging to one of these categories: `S`, `R`, or `B`. This column needs to be present, however its contents are only used for the Strategy `PMNonExhaustStrategy`. So if you don't have the number of axles data, you can leave the column blank for all other Strategies.

Example:

VehName	VehCat	VehPercOfCat	NumberOfAxles
PC petrol <1.4L Euro-1	P	0.2	
LCV diesel M+N1-I Euro-2	L	0.003	

5.5 yeti_format

All Strategies work with these `yeti_format` datasets:



how-to-read-er

yeti_format link data

This is a file with data about the street links in the region you are examining. Each street link corresponds to one line in the yeti_format link data file. It has these columns:

- *LinkID*: The unique ID for the street link
- *Length*: The length of the link in kilometers
- *RoadType*: The road type for the street link. Possible road categories are:

```
RoadType.MW_Nat = Motorway-Nat.
RoadType.MW_City = Motorway-City
RoadType.Trunk_Nat = Trunk road / Primary-Nat.
RoadType.Trunk_City = Trunk road / Primary-City
RoadType.Distr = Distributer / Secondary
RoadType.Local = Loacal / Collector
RoadType.Access = Access-residential
```

- *AreaType*: The area type for the street link. Possible area categories are:

```
AreaType.Urban = rural
AreaType.Rural = urban
```

- *MaxSpeed*: The legal maximum speed for the street link in km/h.

Example:

LinkID	AreaType	RoadType	MaxSpeed
42_123	AreaType.Rural	RoadType.MW_City	100
65_485	AreaType.Urban	RoadType.Local	30

---

yeti_format traffic data

This file contains vehicle count data for every vehicle in yeti_format vehicle data and for each street link, direction, day type and hour. It has these columns:

- *LinkID*: The unique ID for the street link. The LinkIDs need to match the LinkIDs in the yeti_format link data.
- *Dir*: The traffic direction for the given street link:

```
Dir.R = right
Dir.L = left
```

- *DayType*: One of four day types:

```
DayType.MONtoTHU = Monday to Thursday
DayType.FRI = Friday
DayType.SAT = Saturday
DayType.SUN = Sunday or Holiday
```

- *Hour*: A number between 0 and 23.
- *vehicle i*: The number of vehicles belonging to class `vehicle i` at the street link for the given day type, hour and direction. The file needs to contain vehicle count columns for all vehicle names in the yeti_format vehicle data.
- *LOSxPercentage*: The percentage of traffic attributed to the x level of service. Currently implemented levels of service: 1 (Freeflow), 2 (Heavy), 3 (Satur.), and 4 (St+Go).

Example

LinkID	Dir	DayType	Hour	PC petrol <1.4L Euro-1	LCV diesel M+N1-I Euro-2
42_123	Dir.L	DayType.MONtoTHU	0	0.32	0.0023999999
65_485	Dir.R	DayType.SUN	9	12.8	0.012

---

yeti_format vehicle data

A dataset linking each vehicle class to its category.

- *VehicleName*: The name of a vehicle class. For example `PC petrol <1.4L Euro-1`.
- *VehicleCategory*: The vehicle category the vehicle class belongs to. These are the possible categories:

```
VehicleCategory.PC = passenger cars
VehicleCategory.LCV = light commercial vehicles
VehicleCategory.HDV = heavy duty vehicles
VehicleCategory.COACH = coaches
VehicleCategory.UBUS = urban buses
VehicleCategory.MOPED = mopeds
VehicleCategory.MC = motorcycles
```

- *NumberOfAxles*: The number of axles for the given vehicle. This is only relevant for vehicles belonging to one of these categories: `VehicleCategory.HDV`, `VehicleCategory.COACH`, or `VehicleCategory.UBUS`. This column needs to be present, however its contents are only used for the Strategy `PMNonExhaustStrategy`. So if you don't have the number of axles data, you can leave the column blank for all other Strategies.

Example

VehicleName	VehicleCategory	NumberOfAxles
PC petrol <1.4L Euro-1	VehicleCategory.PC	
LCV diesel M+N1-I Euro-2	VehicleCategory.LCV	

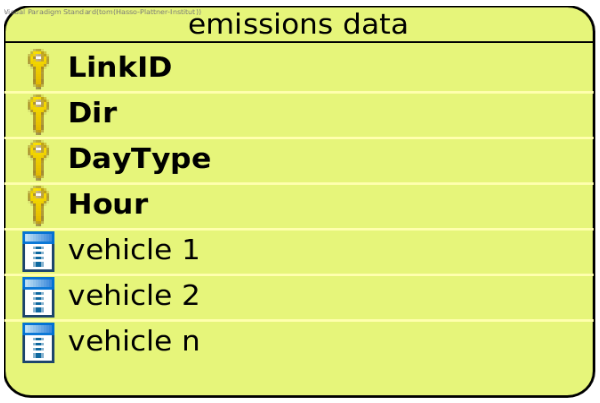
CHAPTER 6

Output data

6.1 Data format

The model outputs one or multiple csv files. How many files are generated depends on the Strategy used for the run.

The output files will be in this format:



how-to-read-er

- *LinkID*: The ID for the street link this row contains emissions data for.
- *Dir*: The traffic direction. `Dir.R` or `Dir.L`.
- *DayType*: One of four day types. `DayType.MONtoTHU`, `DayType.FRI`, `DayType.SAT`, or `DayType.SUN`.
- *Hour*: A number between 0 and 23.
- *vehicle i*: The emissions for the vehicle class named 'vehicle i' at the street link and for the given direction, day type and hour. A real number.

Note that the emission columns match the vehicle names given in the fleet composition data file (if using data in `berlin_format`) or the vehicle data file (if using data in `yeti_format`).

6.2 Output location

In the config file you specify an `output_folder`. The model outputs will be in the specified `output_folder`.

6.3 files in yeti_format

If you are using mode `berlin_format`, YETI generates files in unifed_data format during the run. These files are saved in `output_folder`. If you specify `output_folder_for_yeti_format_data` in the config file, the `yeti_format` files will be in `output_folder_for_yeti_format_data` instead.

CHAPTER 7

What is a Strategy?

A Strategy implements a specific methodology/algorithm for calculating traffic emissions. For example the `CopertHotStrategy` implements emission calculation with the COPERT methodology for hot emissions.

Each model run uses a specific Strategy to calculate emissions. Almost all configuration parameters are dependent on the Strategy used. For instance the data requirements are different for all Strategies and most Strategies need additional configuration parameters.

The use of Strategies makes it easy to extend YETI and include new ways of calculating emissions. If you want to use a custom Strategy, look *here*.

7.1 List of Strategies

Take a look at the individual Strategy pages for further details.

- *CopertColdStrategy*
- *CopertHotStrategy*
- *CopertHotFixedSpeedStrategy*
- *HbefaHotStrategy*
- *PMNonExhaustStrategy*

CHAPTER 8

CopertStrategy

The CopertStrategy implements emission calculation for hot and (optionally) cold emissions focusing on calculation with the Copert methodology.

Note: The possible configurations for this Strategy can become quite complex. Please have a look at the example config files `copert_config.yaml`, `copert_hot_config.yaml`, `copert_cold_config.yaml`, and `copert_hot_fixed_speed_config.yaml` in the folder `example/example_configs/`. They contain example configurations for the `CopertStrategy` and may help you to understand how the `CopertStrategy` can be used.

8.1 hot emissions

The Copert methodology for hot emissions is used to calculate hot emissions. By default the `CopertHotStrategy` is used to calculate hot emissions. If the config argument `fixed_speed` is set to True, the `CopertHotFixedSpeedStrategy` is used to calculate hot emissions using fixed speeds instead.

8.2 cold emissions

By default the `CopertColdStrategy` is used to calculate cold emissions. In the config file the argument `cold_strategy` may be used to specify the path to a different Strategy. If a `cold_strategy` is given in the config file it will be used instead of the `CopertColdStrategy`. Note that cold emissions are only calculated if the config argument `only_hot` is not set to True.

8.3 A note on the data loading process

The conversion of berlin_format data to yeti_format data is conducted independently for the hot and the cold Strategies. This is done to keep the dependencies between the hot and cold Strategies to a minimum. It enables the

`CopertStrategy` to work with any cold Strategy you want.

The consequence is that some computations may be done twice, because they are done by both the hot and the cold `load_berlin_format_data_function` s. The constructed yeti_format files will be in two subfolders of the output folder (the output folder is specified in the config file).

8.4 Data Requirements

What data is required for the CopertStrategy depends on the config arguments. There are several cases:

1. **`only_hot` is set to `yes` in the config file.**
   (a) `fixed_speed` is set to `yes` in the config file. Then the Data Requirements are the same as for the *CopertHotFixedSpeedStrategy*.
   (b) `fixed_speed` is set to `no` or not given in the config file. Then the DataRequirements are the same as for the *CopertHotStrategy*.
2. **`only_hot` is set to `no` or not given in the config file.**
   (a) The argument `cold_strategy` is given in the config file. Then the Data Requirements are the same as for the given `cold_strategy` and the `CopertHotStrategy` (or `CopertHotFixedSpeedStrategy` depending on the config argument `fixed_speed`).
   (b) The argument `cold_strategy` is not given in the config file. Then the Data Requirements are the same as for the *CopertColdStrategy* and the `CopertHotStrategy` (or `CopertHotFixedSpeedStrategy` depending on the config argument `fixed_speed`).

8.5 What to put in the config.yaml

If you want to use the `CopertStrategy` for your calculations, you need to set the following options in your config.yaml. Don't forget to add the parameters specified here: *Configuring YETI*

8.5.1 If using mode `berlin_format`:

```
mode:                               berlin_format
strategy:                           code.copert_strategy.CopertStrategy.CopertStrategy
load_berlin_format_data_function:   code.copert_strategy.load_berlin_format_data.load_
↪copert_berlin_format_data
load_yeti_format_data_function:     code.copert_strategy.load_yeti_format_data.load_
↪copert_yeti_format_data

only_hot:            no   # or yes. Default is no
fixed_speed:         no   # or yes. Default is no

[..]  # add the file locations for the data required by the CopertHotStrategy (or␣
↪CopertHotFixedSpeedStrategy depending on fixed_speed)
[..]  # add any additional args that you want to pass to the CopertHotStrategy (or␣
↪CopertHotFixedSpeedStrategy depending on fixed_speed)

# if only_hot is yes, the following arguments may be omitted.
cold_strategy:                           path.to.ColdStrategy
cold_load_berlin_format_data_function: path.to.load_berlin_format_data_function.for.
↪cold_strategy
```

```
cold_load_yeti_format_data_function:   path.to.load_yeti_format_data_function.for.
↪cold_strategy

[..]  # add the file locations of any additional files needed for the cold_strategy
[..]  # add any additional args that you want to pass to the cold_strategy
```

8.5.2 If using mode `yeti_format`:

```
mode:                                yeti_format
strategy:                            code.copert_strategy.CopertStrategy.CopertStrategy
load_yeti_format_data_function: code.copert_strategy.load_yeti_format_data.load_
↪copert_yeti_format_data

only_hot:            no   # or yes. Default is no
fixed_speed:         no   # or yes. Default is no

[..]  # add the file locations for the data required by the CopertHotStrategy (or␣
↪CopertHotFixedSpeedStrategy depending on fixed_speed)
[..]  # add any additional args that you want to pass to the CopertHotStrategy (or␣
↪CopertHotFixedSpeedStrategy depending on fixed_speed)


# if only_hot is yes, the following arguments may be omitted.
cold_strategy:                       path.to.ColdStrategy
cold_load_yeti_format_data_function: path.to.load_yeti_format_data_function.for.cold_
↪strategy

[..]  # add the file locations of any additional files needed for the cold_strategy
[..]  # add any additional args that you want to pass to the cold_strategy
```

8.5.3 How to deal with naming conflicts

Naming conflicts between the config arguments for the hot Strategy and the arguments for the cold Strategy are a possible issue. For example `berlin_format_emission_factors` is a config argument for the `CopertHotStrategy` and for the `HbefaColdStrategy`, however the two Strategies require input data in a different format. How do we deal with this issue when we want to use the `HbefaColdStrategy` to calculate cold emissions with the `CopertStrategy`?

We solve this naming issue by prefixing the argument that should go to the hot Strategy with `hot_[..]`. The argument that should go to the cold Strategy is prefixed with `cold_[..]`.

In our example for `berlin_format_emission_factors` we would add these lines to the config:

```
hot_berlin_format_emission_factors:    path/to/ef_data_for_hot_strategy.csv
cold_berlin_format_emission_factors:   path/to/ef_data_for_cold_strategy.csv
```

If the two Strategies require the same config argument, there is no need to add prefixes. For example the config argument `berlin_format_link_data` is required for the `CopertHotStrategy` and the `HbefaColdStrategy`. However both Strategies require the exact same data. Therefore it is sufficient to specify it once:

```
berlin_format_link_data:                path/to/berlin_format_link_data.csv
```

8.5.4 A note on the validation_function

We currently don't provide a dedicated validation function for this Strategy. However in most cases you can use a validation function for a different Strategy. Consider these cases:

1. **`only_hot` is set to `yes` in the config file.**
   (a) `fixed_speed` is set to `yes` in the config file. You can use the validation function for the *CopertHotFixedSpeedStrategy*.
   (b) `fixed_speed` is set to `no` or not given in the config file. You can use the validation function for the *CopertHotStrategy*.
2. **`only_hot` is set to `no` or not given in the config file.**
   (a) `cold_strategy` is given in the config file. In this case we don't provide a valiation function that you can use out of the box. If you want to use validation, you will have to *write your own valiation function*.
   (b) `cold_strategy` is not given in the config file. You can use the validation function for the *CopertColdStrategy*.

8.6 Output

The output of this Strategy depends on the config arguments. There are three cases:

1. `only_hot` is set to True. Then the output is the same as for the `CopertStrategy` or the `CopertHotFixedSpeedStrategy` (depending on the the value of the `fixed_speed` config argument)
2. `only_hot` is not set to True and no `cold_strategy` is given in the config file. Then the output is the same as for the `CopertColdStrategy`.
3. `only_hot` is not set to True and a `cold_strategy` is given in the config file. Then the output consists of the files generated by the `CopertHotStrategy` or `CopertHotFixedSpeedStrategy` (prefixed with `hot_[..]`) and the files produced by the `cold_strategy` (prefixed with `cold_[..]`).

CHAPTER 9

CopertHotStrategy

Note: It is recommended to use the *CopertStrategy* with the config argument `only_hot: yes` instead of this Strategy.

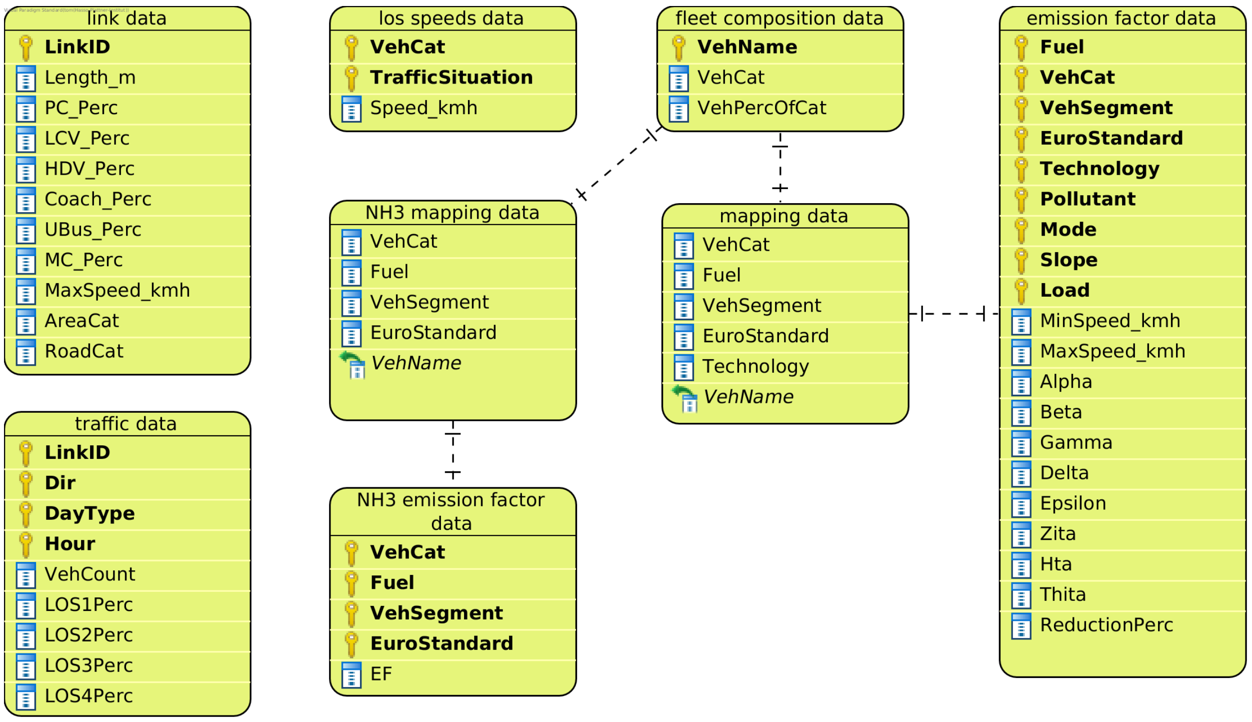
The `CopertHotStrategy` implements emission calculation with the EEA Tier 1 methodology for hot exhaust emissions using COPERT emission factors.

This Strategy uses speed-dependent emission factors with flexible speeds dependent on the level of service. When developing YETI we had access to speed data that depended on the level of service, so we chose to get the speed used for the COPERT emission factor calculation from there. If your speed data has a different format, you can *add your own Strategy* to work with your speed data.

9.1 Data requirements

What data the `CopertHotStrategy` requires depends on the `mode` set in the configuration file for the run.

9.1.1 Data requirements for mode `berlin_format`



how-to-read-er

---

link data Just like the link data required for the other Strategies. Look *here*.

Make sure that the combination of AreaCat, RoadCat and MaxSpeed_kmh matches a traffic situation in the los speeds data.

---

traffic data Just like the traffic data required for the other Strategies. Look *here*.

---

fleet composition data Just like the fleet composition data required for the other Strategies. Look *here*.

---

los speeds data This dataset contains speed data for vehicle categories and traffic situations. It is used to find out the speed of a vehicle for a particular level of service.

- *VehCat*: A vehicle category. Valid categories are

```
pass. car
LCV
coach
urban bus
motorcycle
HGV
```

- *TrafficSituation*: Each value in this column is a String describing a particular traffic situation in the format `{area type}/{road type}/{max speed}/{level of service}`. Acceptable area types are `URB` and `RUR`. For level of service choose one of these values `Freeflow`, `Heavy`, `Satur.`, or `St+Go`.

Possible road types are as follows. Please note that the possible road types are hard coded. If you are using different road definitions, the easiest would be to map them to the definitions used by YETI and work with the YETI definitions.

```
# possible road types
MW-Nat.
MW-City
Trunk-Nat.
Trunk-City
Distr
Local
Access
```

```
# Example traffic situations
URB/MW-City/70/Freeflow   # Freeflow is the first level of service (LOS 1)
URB/MW-City/70/Heavy      # Heavy is the second level of service (LOS 2)
URB/MW-City/70/Satur.     # Satur. is the third level of service (LOS 3)
URB/MW-City/70/St+Go      # St+Go is the fourth level of service (LOS 4)
```

- *Speed_kmh*: The speed for a vehicle of category VehCat driving in the corresponding TrafficSituation. Unit: km/h

The los speeds data should contain values for all vehicle categies and for traffic situation strings describing all street links and all los types.

Example:

VehCat	TrafficSituation	Speed_kmh
pass. car	URB/MW-City/70/Freeflow	70
pass. car	URB/MW-City/70/Heavy	55
pass. car	URB/MW-City/70/Satur	40
pass. car	URB/MW-City/70/St+Go	30

---

emission factor data This dataset contains the necessary attributes to calculate speed dependent emission factors to be used in the calculation of hot exhaust emissions according to EEA methodology.

Note that this dataset needs to contain values for the pollutants you are using, otherwise you will encounter errors in the emission calculation.

Example:

Fuel	VehCat	VehSegment	EuroStandard	Technology	Pollutant	Mode	Slope	Load	MinSpeed_kmh	MaxSpeed_kmh	Alpha	Beta	Gamma	Delta	Epsilon	Zita	Hta	Thita	ReductionPerc
Petrol	Passenger Cars	Small	Euro 4	GDI	CO				5	130	0.651	16.6	0.468	-0.48	10.186	79.57	3	-0.79	0.3
Petrol	Passenger Cars	Small	Euro 4	GDI	NOx				5	130	0.896	86.5	0.167	-0.74	6.3214	761	97.	-0.55	0

---

mapping data This file is used to map vehicle names to their emission factor attributes in the emission factor data.

The values in the columns `VehCat`, `Fuel`, `VehSegment`, `EuroStandard`, and `Technology` need to match the values in the columns of the emission factor data exactly.

The vehicle names in the column `VehName` need to match the the vehicle names in `fleet composition data >> VehName` exactly. More precisely, each vehicle in `fleet composition data >> VehName` needs one corresponding row in the mapping data.

Example:

Say we want to construct a mapping between the following two files:

fleet composition data

VehName	VehCat	VehPercOfCat	NumberOfAxles
PC petrol <1.4L Euro-1	P	0.2	
LCV diesel M+N1-I Euro-2	L	0.003	

emission factor data

Fuel	VehCat	VehSegment	EuroStandard	Technology	Pollutant	Mode	Slope	Load	MinSpeed_kmh	MaxSpeed_kmh	Alpha	Beta	Gamma	Delta	Epsilon	Zita	Hta	Thita	ReductionPerc
Petrol	Passenger Cars	Small	Euro 4	GDI	CO				5	130	0.651	16.6	0.468	-0.48	10.1867	9.57	3	-0.79	0.3
Diesel	Light Commercial Vehicles	M+N1-I	Euro 2		NOx				5	100	0.896	86.5	0.167	-0.74	6.32147	761	97.	-0.55	0

Our mapping file would look like this:

mapping data

VehCat	Fuel	VehSegment	EuroStandard	Technology	VehName
Passenger Cars	Petrol	Small	Euro 4	GDI	PC petrol <1.4L Euro-1
Light Commercial Vehicles	Diesel	M+N1-I	Euro 2		LCV diesel M+N1-I Euro-2

---

nh3 mapping data [OPTIONAL] Like the mapping data described above, but without the column `Technology`. Also this file is used to construct a mapping between the fleet composition data and the nh3 emission factor data.

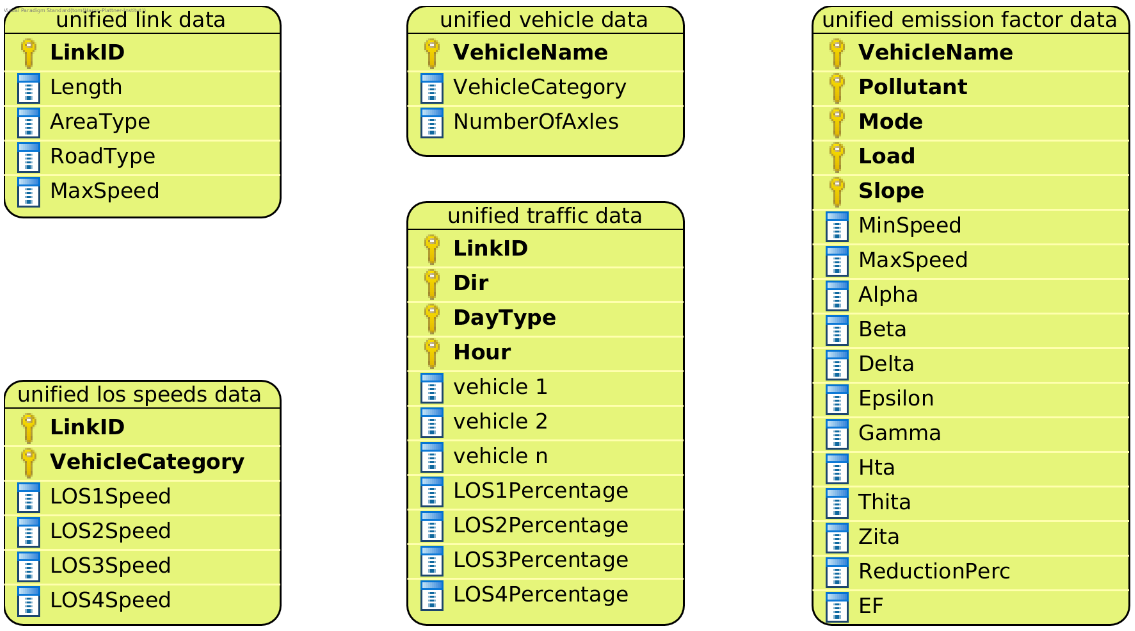
---

nh3 emission factor data [OPTIONAL] This file contains Tier 2 emission factor values for the pollutant NH3.

Example:

VehCat	Fuel	VehSegment	EuroStandard	EF
Passenger Cars	Petrol	Small	Euro 4	0.8
Light Commercial Vehicles	Diesel	M+N1-I	Euro 2	4.95

9.1.2 Data requirements for mode `yeti_format`



how-to-read-er

---

yeti_format link data Just like the yeti_format link data required for the other Strategies. See *here*.

Make sure that the combination of AreaType, RoadType and MaxSpeed matches a traffic situation in the los speeds data.

---

yeti_format vehicle data Just like the yeti_format vehicle data required for the other Strategies. See *here*.

---

yeti_format traffic data Just like the yeti_format traffic data required for the other Strategies. See *here*.

---

yeti_format los speeds data This dataset contains data about the speeds associated with the levels of service for the links and vehicle categories used.

- *LinkID*: The ID of a street link. Needs to match the link IDs in `yeti_format link data >> LinkID`.
- *VehicleCategory*: One of the following vehicle categories:

```
VehicleCategory.PC
VehicleCategory.LCV
VehicleCategory.HDV
VehicleCategory.COACH
VehicleCategory.UBUS
VehicleCategory.MOPED
VehicleCategory.MC
```

- *LOSxSpeed*: The average speed of vehicles belonging to the given vehicle category at the given link for the x level of service. Currently implemented levels of service: `1` (Freeflow), `2` (Heavy), `3` (Satur.), and `4` (St+Go).

Example:

LinkID	VehicleCategory	LOS1Speed	LOS2Speed	LOS3Speed	LOS4Speed
123_87	VehicleCategory.PC	44.9160	36.996669	30.752666	12.756747
123_87	VehicleCategory.LCV	44.9160	36.996669	30.752666	12.756747
123_87	VehicleCategory.HDV	39.8291	30.092407	28.670288	11.770976

---

yeti_format emission factor data

This dataset contains emission factor attributes used in the emission factor calculation with the copert methodology for all vehicles in the fleet.

It can contain the optional column EF giving you the option to use fixed emission factors that are independent of speed. Values in EF will be used as the emission factor for the given vehicle and pollutant and will take precedence over the emission factor calculation with the copert methodology. If you want to use fixed emission factors for some vehicles and speed-depend emission factors for other vehicles, you can leave the EF blank for the vehicles that you want to use speed-dependent copert emission factors for.

Note that this dataset needs to contain values for the pollutants you are using, otherwise you will encounter errors in the emission calculation.

- *VehicleName*: The name of a vehicle class. Needs to match the vehicle names in `yeti_format vehicle data >> VehicleName` exactly.
- *Pollutant*: One of the following pollutants:

```
PollutantType.NOx
PollutantType.CO
PollutantType.NH3
PollutantType.VOC
PollutantType.PM_Exhaust
```

- *Mode*: The mode, as used by the copert methodology.
- *Load*: The load, as used by the copert methodology. Note that the load is only used to filter the yeti_format emission factor data. Only rows with load 0 or blank will be considered for the emission factor calculation.
- *Slope*: The slope, as used by the copert methodology. Note that the slope is only used to filter the yeti_format emission factor data. Only rows with slope 0 or blank will be considered for the emission factor calculation.
- *EF*: [OPTIONAL] A fixed emission factor to be used for the given vehicle name and pollutant. If not blank the EF takes precedence over the emission calculation with the copert methodology.
- The other columns contain the attributes used in the copert emission factor calculation.

Example:

VehicleName	Pollutant	Mode	Slope	Load	MinSpeed	MaxSpeed	Alpha	Beta	Gamma	Delta	Epsilon	Zita	Hta	Thita	ReductionPerc	EF
PC petrol <1.4L Euro-1	CO		0	0	5	130	0.651	16.6	0.468	-0.48	10.1869	7.57	3	-0.79	0.3	
LCV diesel M+N1-I Euro-2	NOx				5	100	0.896	86.5	0.167	-0.74	6.3214	7761	97.	-0.55	0	3

9.2 Supported pollutants

CopertHotStrategy supports these pollutants:

```
PollutantType.NOx
PollutantType.CO
PollutantType.NH3
PollutantType.VOC
PollutantType.PM_Exhaust
```

Set the pollutants for a run in your config file. For example:

```
pollutants:           [PollutantType.CO, PollutantType.NOx]
```

Make sure to include emission factors for the pollutants you are using in the emission factor data.

9.3 What to put in the config.yaml

If you want to use the `CopertHotStrategy` for your calculations, you need to set the following options in your `config.yaml`. Don't forget to add the parameters specified here: *Configuring YETI*

9.3.1 If using mode `berlin_format`:

```
strategy:                       code.copert_hot_strategy.CopertHotStrategy.
↪CopertHotStrategy
load_berlin_format_data_function:     code.copert_hot_strategy.load_berlin_format_
↪data.load_copert_hot_berlin_format_data
load_yeti_format_data_function:   code.copert_hot_strategy.load_yeti_format_data.load_
↪copert_hot_yeti_format_data
validation_function:            code.script_helpers.validate_files.validate_copert_
↪berlin_format_files

berlin_format_link_data:              path/to/link_data.csv
berlin_format_fleet_composition:      path/to/fleet_composition_data.csv
berlin_format_emission_factors:       path/to/emission_factor_data.csv
berlin_format_los_speeds:             path/to/los_speeds_data.csv
berlin_format_traffic_data:           path/to/traffic_data.csv
```

```
berlin_format_vehicle_mapping:        path/to/vehicle_mapping_data.csv

use_nh3_tier2_ef:              yes or no
# if you set use_nh3_tier2_ef to yes, also add these lines:
berlin_format_nh3_emission_factors:   path/to/nh3_emission_factor_data.csv
berlin_format_nh3_mapping:            path/to/nh3_mapping_data.csv
```

You may have data on Tier 2 emission factors for NH3. If you set `use_nh3_tier2_ef: yes` in the config file, YETI will read them from the specified files and use them in the emission calculation for pollutant `PollutantType.NH3`.

9.3.2 If using mode `yeti_format`:

```
strategy:                      code.copert_hot_strategy.CopertHotStrategy.
↪CopertHotStrategy
load_yeti_format_data_function:    code.copert_hot_strategy.load_yeti_format_data.load_
↪copert_hot_yeti_format_data
validation_function:           code.script_helpers.validate_files.validate_copert_yeti_
↪format_files

yeti_format_emission_factors:      path/to/yeti_format_ef_data.csv
yeti_format_los_speeds:            path/to/yeti_format_los_speed_data.csv
yeti_format_vehicle_data:          path/to/yeti_format_vehicle_data.csv
yeti_format_link_data:             path/to/yeti_format_link_data.csv
yeti_format_traffic_data:          path/to/yeti_format_traffic_data.csv
```

CHAPTER 10

CopertHotFixedSpeedStrategy

Note: It is recommended to use the *CopertStrategy* with the config arguments `only_hot: yes` and `fixed_speed: yes` instead of this Strategy.

The `CopertHotFixedStrategy` implements emission calculation with the COPERT methodology for hot emissions at the street level. It uses speed-dependent emission factors with fixed speed values.

There are two ways to specify the fixed speed:

1. Add the parameter `v: {speed, an integer}` to your configuration file. The specified speed is used for all emission calculations.
2. Add a speed column (`Speed_kmh`) to the input file with the link data to specify a fixed speed for all emission calculations at a particular street link.

If you include `v` in the config file and add a speed column to the link data, the speed value in `v` will take precedence.

10.1 Data requirements

What data the `CopertHotFixedSpeedStrategy` requires depends on the `mode` set in the configuration file for the run.

10.1.1 Data requirements for mode `berlin_format`

The same data in `berlin_format` as `CopertHotStrategy` is required with these modifications:

- los speeds data is not required.
- The link data may contain the column `Speed_kmh` with fixed speeds for each street link.
- The traffic data does not need to contain los percentage columns (LOS1Perc . . . LOS4Perc).

10.1.2 Data requirements for mode `yeti_format`

The same data in `yeti_format` as `CopertHotStrategy` is required with these modifications:

- yeti_format los speeds data is not required.
- The yeti_format link data may contain the column `Speed` with fixed speeds for each street link.
- The yeti_format traffic data does not need to contain los percentage columns (LOS1Percentage … LOS4Percentage)

10.2 Supported pollutants

`CopertHotFixedSpeedStrategy` supports these pollutants:

```
PollutantType.NOx
PollutantType.CO
PollutantType.NH3
PollutantType.VOC
PollutantType.PM_Exhaust
```

Set the pollutants for a run in your config file. For example:

```
pollutants:          [PollutantType.CO, PollutantType.NOx]
```

Make sure to include emission factors for the pollutants you are using in the emission factor data.

10.3 What to put in the config.yaml

If you want to use the `CopertHotFixedSpeedStrategy` for the your calculations, you need to set the following options in your `config.yaml`. Don't forget to add the parameters specified here: *Configuring YETI*.

10.3.1 If using mode `berlin_format`:

```
strategy:                          code.copert_hot_fixed_speed_strategy.
↪CopertHotFixedSpeedStrategy.CopertHotFixedSpeedStrategy
load_berlin_format_data_function:          code.copert_hot_fixed_speed_strategy.load_
↪berlin_format_data.load_copert_fixed_speed_berlin_format_data
load_yeti_format_data_function:        code.copert_hot_fixed_speed_strategy.load_yeti_
↪format_data.load_copert_fixed_speed_yeti_format_data
validation_function:               code.copert_hot_fixed_speed_strategy.validate.
↪validate_copert_fixed_speed_berlin_format_files

# if you want to use a global speed for all links, include this:
v:                              50

berlin_format_link_data:              path/to/link_data.csv
berlin_format_fleet_composition:      path/to/fleet_composition_data.csv
berlin_format_emission_factors:       path/to/emission_factor_data.csv
berlin_format_traffic_data:           path/to/traffic_data.csv
berlin_format_vehicle_mapping:        path/to/vehicle_mapping_data.csv

use_nh3_tier2_ef:               yes or no
```

```
# if you set use_nh3_tier2_ef to yes, also add these lines:
berlin_format_nh3_emission_factors:   path/to/nh3_emission_factor_data.csv
berlin_format_nh3_mapping:            path/to/nh3_mapping_data.csv
```

You may have data on Tier 2 emission factors for NH3. If you set `use_nh3_tier2_ef: yes` in the config file, YETI will read them from the specified files and use them in the emission calculation for pollutant `PollutantType.NH3`.

10.3.2 If using mode `yeti_format`:

```
strategy:                           code.copert_hot_fixed_speed_strategy.
↪CopertHotFixedSpeedStrategy.CopertHotFixedSpeedStrategy
load_yeti_format_data_function:         code.copert_hot_fixed_speed_strategy.load_yeti_
↪format_data.load_copert_fixed_speed_yeti_format_data
validation_function:                code.copert_hot_fixed_speed_strategy.validate.
↪validate_copert_fixed_speed_yeti_format_files

# if you want to use a global speed for all links, include this:
v:                               50

yeti_format_emission_factors:     path/to/yeti_format_ef_data.csv
yeti_format_vehicle_data:         path/to/yeti_format_vehicle_data.csv
yeti_format_link_data:            path/to/yeti_format_link_data.csv
yeti_format_traffic_data:         path/to/yeti_format_traffic_data.csv
```

CHAPTER 11

CopertColdStrategy

Warning: This Strategy cannot be used by itself. It can only be used as a `cold_strategy` with the `CopertStrategy` or the `HbefaStrategy`.

The `CopertColdStrategy` implements emission calculation with the COPERT methodology for cold start emissions. It uses global assumptions about the average length of a trip (in km/h) and the average ambient temperature (in degrees Celsius) that are set in the configuration file.

Also in the configuration file you may specify some road types and area types that will be excluded from the cold emission calculation. This means that cold emissions will be zero for all vehicles at links that belong to the road types or area types that you want to exclude. Possible road types to be excluded are: `MW_Nat`, `MW_City`, `Trunk_Nat`, `Trunk_City`, `Distr`, `Local`, and `Access`. Area types that can be excluded are `Rural` and `Urban`.

The `CopertColdStrategy` takes the emissions from the hot strategy that is used in the `CopertStrategy` or `HbefaStrategy` as input. It uses the hot emissions to derive hot emission factors and to calculate total emissions.

Output of a model run with this Strategy are two csv files per pollutant:

- cold start emissions
- total emissions

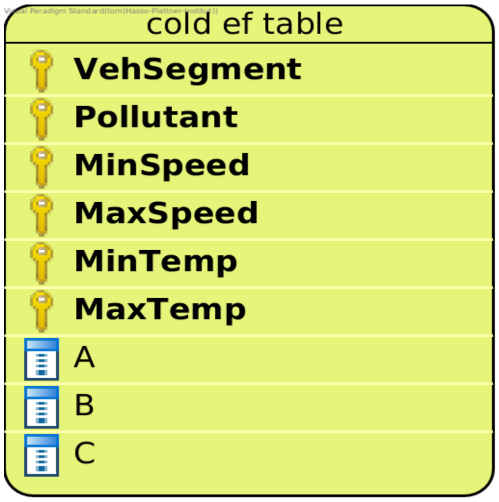
11.1 Data requirements

What data the `CopertColdStrategy` requires depends on the `mode` set in the configuration file for the run.

11.1.1 Data requirements for mode `berlin_format`

All input data in `berlin_format` required by the `CopertHotStrategy` is also required for the `CopertColdStrategy`.

Additionally a file with **cold emission factors** is required:



how-to-read-er

This is the cold ef table as provided by EEA recommendations.

- *Pollutant*: One of the following: `CO`, `NOx`, or `VOC`.
- *VehSegment*: A vehicle segment. One of the following: `Mini`, `Small`, `Medium`, or `Large-SUV-Executive`.
- *MinSpeed*: The minimum speed the A,B and C values in this row are valid for.
- *MaxSpeed*: The maximum speed the A,B and C values in this row are valid for.
- *MinTemp*: The minimum temperature the A,B and C values in this row are valid for.
- *MaxTemp*: The maximum temperature the A,B and C values in this row are valid for.
- *A*: The A parameter for the cold ef calculation.
- *B*: The B parameter for the cold ef calculation.
- *C*: The C parameter for the cold ef calculation.

Example:

Pollutant	VehSegment	MinSpeed	MaxSpeed	MinTemp	MaxTemp	A	B	C
CO	Mini	5	25	-20	15	0.563	-0.895	4.964
CO	Mini	26	45	-20	15	0.842	-0.349	3.485
CO	Mini	5	45	15		0.222	-0.876	10.12

11.1.2 Data requirements for mode `yeti_format`

All data in `yeti_format` required by the `CopertHotStrategy` is also required for the `CopertColdStrategy`.

Additional requirements:

- A file with cold emission factors is required, as *described above*.
- A vehicle mapping file is required. It needs to be in *this format*.

11.2 Supported Pollutants

`CopertColdStrategy` supports these pollutants:

```
PollutantType.CO
PollutantType.NOx
PollutantType.VOC
```

Set the pollutants for a run in your config file. For example:

```
pollutants:           [PollutantType.CO, PollutantType.NOx]
```

Make sure to include emission factors for the pollutants you are using in the emission factor data.

11.3 What to put in the config.yaml

If you want to use the `CopertColdStrategy` for your calculations, you need to set the following options in your `config.yaml`. Don't forget to add the parameters specified here: *Configuring YETI*

11.3.1 If using mode `berlin_format`:

```
strategy:                      code.copert_strategy.CopertStrategy.CopertStrategy
load_berlin_format_data_function:     code.copert_strategy.load_berlin_format_data.
↪load_copert_berlin_format_data
load_yeti_format_data_function:   code.copert_strategy.load_yeti_format_data.load_
↪copert_yeti_format_data
validation_function:           code.copert_cold_strategy.validate.validate_copert_cold_
↪berlin_format_files

berlin_format_link_data:              path/to/link_data.csv
berlin_format_fleet_composition:      path/to/fleet_composition_data.csv
berlin_format_emission_factors:       path/to/emission_factor_data.csv
berlin_format_los_speeds:             path/to/los_speeds_data.csv
berlin_format_traffic_data:           path/to/traffic_data.csv
berlin_format_vehicle_mapping:        path/to/vehicle_mapping_data.csv
berlin_format_cold_ef_table:          path/to/cold_ef_table.csv

ltrip:                         12  # the average length of a trip in km/h
temperature:                   15  # the average ambient temperature in °C
exclude_road_types:            [MW_City]  # Exclude multiple road types like this: [MW_
↪City, Trunk_City]
exclude_area_types:            [Rural]    # Or: [Urban]
```

11.3.2 If using mode `yeti_format`:

```
strategy:                      code.copert_strategy.CopertStrategy.CopertStrategy
load_yeti_format_data_function:   code.copert_strategy.load_yeti_format_data.load_
↪copert_yeti_format_data
validation_function:           code.copert_cold_strategy.validate.validate_copert_cold_
↪yeti_format_files

yeti_format_emission_factors:     path/to/yeti_format_ef_data.csv
```

```
yeti_format_los_speeds:          path/to/yeti_format_los_speed_data.csv
yeti_format_vehicle_data:        path/to/yeti_format_vehicle_data.csv
yeti_format_link_data:           path/to/yeti_format_link_data.csv
yeti_format_traffic_data:        path/to/yeti_format_traffic_data.csv
yeti_format_cold_ef_table:       path/to/cold_ef_table.csv
yeti_format_vehicle_mapping:     path/to/vehicle_mapping_data.csv

ltrip:                       12  # the average length of a trip in km/g
temperature:                 15  # the average ambient temperature in °C
exclude_road_types:          [MW_City]  # Exclude multiple road types like this: [MW_
↪City, Trunk-City]
exclude_area_types:          [Rural]    # Or: [Urban]
```

CHAPTER 12

HbefaStrategy

The HbefaStrategy implements emission calculation for hot and (optionally) cold emissions focusing on calculation with the Hbefa methodology.

Note: The possible configurations for this Strategy can become quite complex. Please have a look at the example config files `hbefa_config.yaml`, `hbefa_hot_config.yaml`, and `hbefa_hot_and_copert_cold_config.yaml` in the folder `example/example_configs/`. They contain example configurations for the `HbefaStrategy` and may help you to understand how the `HbefaStrategy` can be used.

12.1 hot emissions

The `HbefaHotStrategy` is used to calculate hot emissions.

12.2 cold emissions

By default the `HbefaColdStrategy` is used to calculate cold emissions. In the config file the argument `cold_strategy` may be used to specify the path to a different Strategy. If a `cold_strategy` is given in the config file it will be used instead of the `HbefaColdStrategy`. Note that cold emissions are only calculated if the config argument `only_hot` is not set to True.

12.3 A note on the data loading process

The conversion of berlin_format data to yeti_format data is conducted independently for the hot and the cold Strategies. This is done to keep the dependencies between the hot and cold Strategies to a minimum. It enables the `HbefaStrategy` to work with any cold Strategy you want.

The consequence is that some computations may be done twice, because they are done by both the hot and the cold `load_berlin_format_data_function` s. The constructed yeti_format files will be in two subfolders of the output folder (the output folder is specified in the config file).

12.4 Data Requirements

The data required for the `HbefaHotStrategy` is required for the `HbefaStrategy` in all cases.

If `only_hot` is set to `no` or not given in the config file additional data is required depending on the config argument `cold_strategy`:

- The argument `cold_strategy` is not given in the config file. Then the data required for the `CopertColdStrategy` is also required.
- The argument `cold_strategy` is given in the config file. Then the data required for the given `cold_strategy` is also required.

12.5 What to put in the config.yaml

If you want to use the `HbefaStrategy` for your calculations, you need to set the following options in your config.yaml. Don't forget to add the parameters specified here: *Configuring YETI*

12.5.1 If using mode `berlin_format`:

```
mode:                              berlin_format
strategy:                          code.hbefa_strategy.HbefaStrategy.HbefaStrategy
load_berlin_format_data_function: code.hbefa_strategy.load_berlin_format_data.load_
↪hbefa_berlin_format_data
load_yeti_format_data_function:    code.hbefa_strategy.load_yeti_format_data.load_
↪hbefa_yeti_format_data

only_hot:             no  # or yes. Default is no

[..]  # add the file locations for the data required by the HbefaHotStrategy

# if only_hot is yes, the following arguments may be omitted.
cold_strategy:                          path.to.ColdStrategy
cold_load_berlin_format_data_function: path.to.load_berlin_format_data_function.for.
↪cold_strategy
cold_load_yeti_format_data_function:    path.to.load_yeti_format_data_function.for.
↪cold_strategy

[..]  # add the file locations of any additional files needed for the cold_strategy
[..]  # add any additional args that you want to pass to the cold_strategy
```

12.5.2 If using mode `yeti_format`:

```
mode:                            yeti_format
strategy:                        code.copert_strategy.HbefaStrategy.HbefaStrategy
load_yeti_format_data_function: code.hbefa_strategy.load_yeti_format_data.load_hbefa_
↪yeti_format_data
```

```
only_hot:            no  # or yes. Default is no

[..]  # add the file locations for the data required by the HbefaHotStrategy

# if only_hot is yes, the following arguments may be omitted.
cold_strategy:                     path.to.ColdStrategy
cold_load_yeti_format_data_function: path.to.load_yeti_format_data_function.for.cold_
↪strategy

[..]  # add the file locations of any additional files needed for the cold_strategy
[..]  # add any additional args that you want to pass to the cold_strategy
```

12.5.3 How to deal with naming conflicts

Naming conflicts between the config arguments for the hot Strategy and the arguments for the cold Strategy are an issue that you will certainly encounter. For example berlin_format_emission_factors is a config argument for the HbefaHotStrategy and for the HbefaColdStrategy, however the two Strategies require input data in a different format. How do we deal with this issue when we want to use the HbefaColdStrategy to calculate cold emissions with the HbefaStrategy?

We solve this naming issue by prefixing the argument that should go to the hot Strategy with hot_[..]. The argument that should go to the cold Strategy is prefixed with cold_[..].

In our example for berlin_format_emission_factors we would add these lines to the config:

```
hot_berlin_format_emission_factors:     path/to/ef_data_for_hot_strategy.csv
cold_berlin_format_emission_factors:    path/to/ef_data_for_cold_strategy.csv
```

If the two Strategies require the same config argument, there is no need to add prefixes. For example the config argument berlin_format_link_data is required for the HbefaHotStrategy and the HbefaColdStrategy. However both Strategies require the exact same data. Therefore it is sufficient to specify it once:

```
berlin_format_link_data:                path/to/berlin_format_link_data.csv
```

12.5.4 A note on the validation_function

We currently don't provide a dedicated validation function for this Strategy. If you are only calculating hot emissions (set only_hot: yes in the config file) you can use the validation function for the HbefaHotStrategy.

12.6 Output

The output of this Strategy depends on the config arguments. There are three cases:

1. only_hot is set to True. Then the output is the same as for the HbefaHotStrategy.
2. only_hot is not set to True and no cold_strategy is given in the config file. Then the output consists of the files generated by the HbefaHotStrategy (prefixed with hot_[..]) and the files produced by the HbefaColdStrategy (prefixed with cold_[..]).
3. only_hot is not set to True and a cold_strategy is given in the config file. Then the output consists of the files generated by the HbefaHotStrategy (prefixed with hot_[..]) and the files produced by the cold_strategy (prefixed with cold_[..]).

CHAPTER 13

HbefaHotStrategy

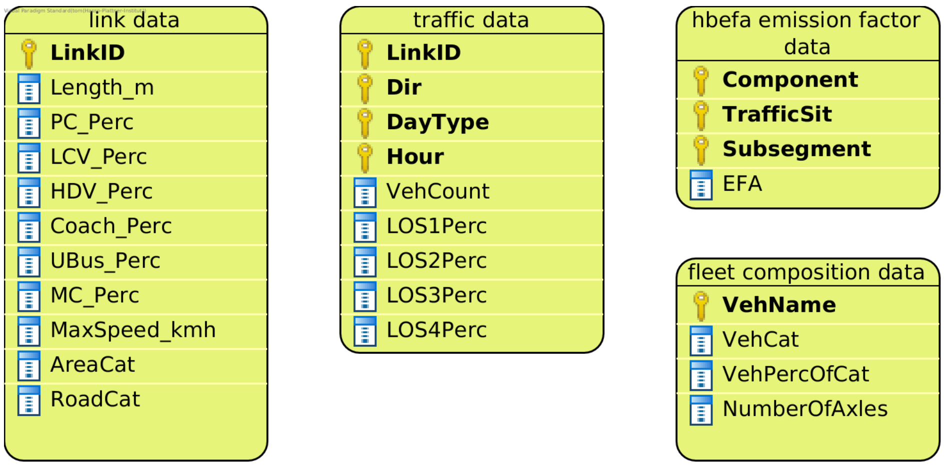
`HbefaHotStrategy` implements emission calculation with the HBEFA methodology for hot exhaust emissions. It uses emission factors that are dependent on the vehicle and the traffic situation.

We used HBEFA emission factors and values from the database v3.3.

13.1 Data Requirements

What data the `HbefaHotStrategy` requires depends on the `mode` set in the configuration file for the run.

13.1.1 Data requirements for mode `berlin_format`



how-to-read-er

---

link data Just like the link data required for the other Strategies. See *here*.

---

traffic data Just like the traffic data required for the other strategies. See *here*.

---

fleet composition data Just like the fleet composition data required for the other Strategies. See *here*.

---

hbefa emission factor data A dataset with HBEFA emission factors.

Note that this dataset needs to contain values for the pollutants you are using, otherwise you will encounter errors in the emission calculation.

- *Component*: A pollutant. Accepted pollutants are:

```
NOx
CO
NH3
VOC
PM Exhaust
PM  # interpreted as PM Exhaust
```

- *TrafficSit*: A String describing a particular traffic situation in the format `{area type}/{road type}/{max speed}/{level of service}`. Acceptable area types are `URB` and `RUR`. For level of service choose one of these values `Freeflow`, `Heavy`, `Satur.`, or `St+Go`. Possible road types are

```
MW-Nat.
MW-City
Trunk-Nat.
Trunk-City
Distr
Local
Access
```

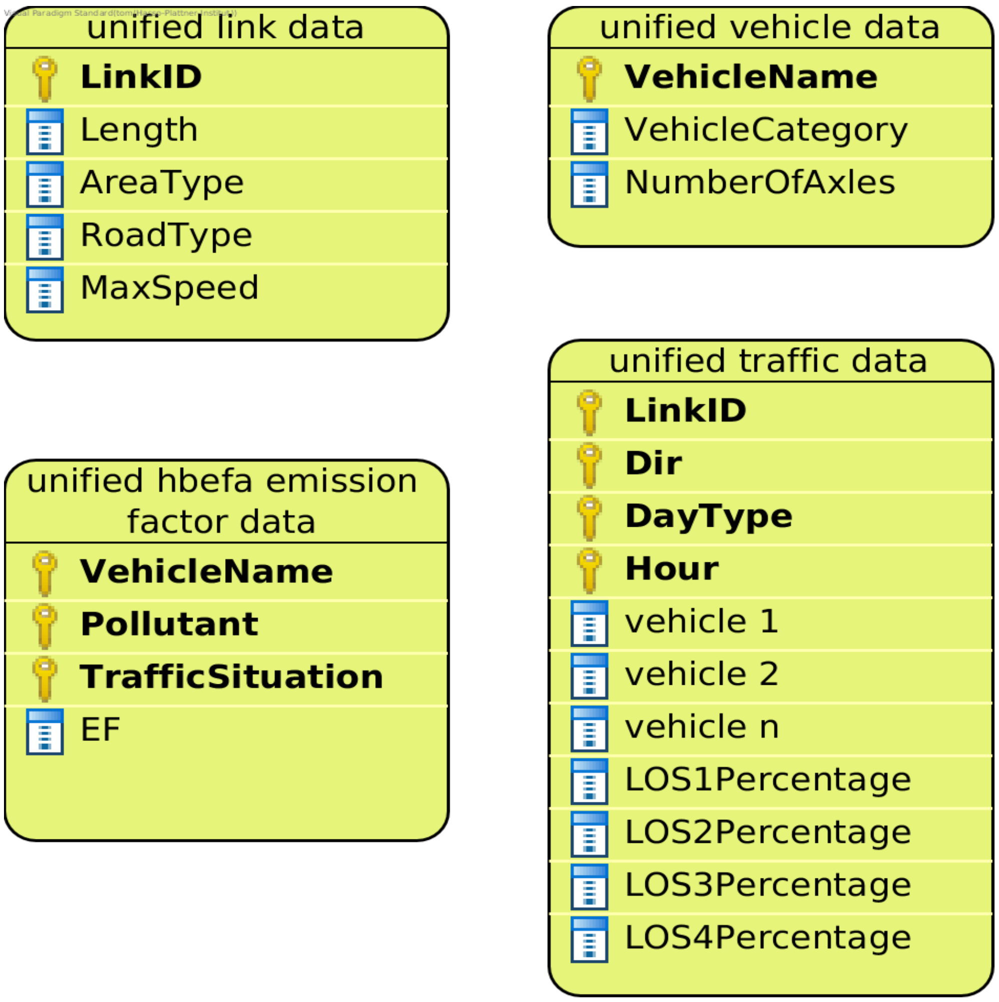
```
# Example traffic situations
URB/MW-City/70/Freeflow   # Freeflow is the first level of service (LOS 1)
URB/MW-City/70/Heavy      # Heavy is the second level of service (LOS 2)
URB/MW-City/70/Satur.     # Satur. is the third level of service (LOS 3)
URB/MW-City/70/St+Go      # St+Go is the fourth level of service (LOS 4)
```

- *Subsegment*: A vehicle name. The values in Subsegment need to match the values in `fleet composition data >> VehName` exactly.
- *EFA*: The emission factor for the component, traffic situation and subsegment in the same row.

Example

Component	TrafficSit	Subsegment	EFA
NOx	URB/MW-City/70/Freeflow	PC petrol <1.4L Euro-1	1.5
NOx	URB/MW-City/70/Heavy	PC petrol <1.4L Euro-1	1.6
NOx	URB/MW-City/70/Satur.	PC petrol <1.4L Euro-1	1.7
NOx	URB/MW-City/70/St+Go	PC petrol <1.4L Euro-1	1.8

13.1.2 Data requirements for mode `yeti_format`



how-to-read-er

---

yeti_format link data Just like the yeti_format link data required for the other Strategies. See *here*.

---

yeti_format traffic data Just like the yeti_format traffic data required for the other strategies. See *here*.

---

yeti_format vehicle data Just like the yeti_format link data required for the other Strategies. See *here*.

---

yeti_format hbefa emission factor data A dataset with HBEFA emission factors.

Note that this dataset needs to contain values for the pollutants you are using, otherwise you will encounter errors in the emission calculation.

- *Pollutant*: A pollutant. Accepted pollutants are:

```
PollutantType.NOx
PollutantType.CO
PollutantType.NH3
PollutantType.VOC
PollutantType.PM_Exhaust
```

- *TrafficSituation*: Just like the column TrafficSit in the hbefa emission factor data for mode `berlin_format`
- *VehicleName*: A vehicle name. The values in this column need to match the values in `yeti_format vehicel data >> VehicleName` exactly.
- *EF*: The emission factor for the pollutant, traffic situation and vehicle name in the same row.

Example

Pollutant	TrafficSituation	VehicleName	EF
PollutantType.NOx	URB/MW-City/70/Freeflow	PC petrol <1.4L Euro-1	1.5
PollutantType.NOx	URB/MW-City/70/Heavy	PC petrol <1.4L Euro-1	1.6
PollutantType.NOx	URB/MW-City/70/Satur	PC petrol <1.4L Euro-1	1.7
PollutantType.NOx	URB/MW-City/70/St+Go	PC petrol <1.4L Euro-1	1.8

13.2 Supported pollutants

`HbefaHotStrategy` supports these pollutants:

```
PollutantType.NOx
PollutantType.CO
PollutantType.NH3
PollutantType.VOC
PollutantType.PM_Exhaust
```

Set the pollutants for a run in your config file. For example:

```
pollutants:           [PollutantType.CO, PollutantType.NOx]
```

Make sure to include emission factors for the pollutants you are using in the emission factor data.

13.3 What to put in the config.yaml

If you want to use the `HbefaHotStrategy` for your calculations, you need to set the following options in your `config.yaml`. Don't forget to add the parameters specified here: *Configuring YETI*

13.3.1 If using mode `berlin_format`:

```
strategy:                    code.hbefa_hot_strategy.HbefaHotStrategy.
↪HbefaHotStrategy
load_berlin_format_data_function:     code.hbefa_hot_strategy.load_berlin_format_data.
↪load_hbefa_hot_berlin_format_data
load_yeti_format_data_function:    code.hbefa_hot_strategy.load_yeti_format_data.load_
↪hbefa_hot_yeti_format_data
validation_function:         code.hbefa_hot_strategy.validate.validate_hbefa_berlin_
↪format_files
```

```
berlin_format_link_data:           path/to/link_data.csv
berlin_format_fleet_composition:   path/to/fleet_composition_data.csv
berlin_format_emission_factors:    path/to/hbefa_emission_factor_data.csv
berlin_format_traffic_data:        path/to/traffic_data.csv
```

13.3.2 If using mode `yeti_format`:

```
strategy:                      code.hbefa_hot_strategy.HbefaHotStrategy.
↪HbefaHotStrategy
load_yeti_format_data_function:    code.hbefa_hot_strategy.load_yeti_format_data.load_
↪hbefa_hot_yeti_format_data
validation_function:           code.hbefa_hot_strategy.validate.validate_hbefa_yeti_
↪format_files

yeti_format_emission_factors:      path/to/yeti_format_hbefa_ef_data.csv
yeti_format_vehicle_data:          path/to/yeti_format_vehicle_data.csv
yeti_format_link_data:             path/to/yeti_format_link_data.csv
yeti_format_traffic_data:          path/to/yeti_format_traffic_data.csv
```

CHAPTER 14

PMNonExhaustStrategy

The `PMNonExhaustStrategy` implements emission calculation for PM from non-exhaust emissions. Sources for PM non-exhaust emissions are tyre wear, brake wear and road surface emissions.

It uses a global assumption about the load_factor for trucks. You need to specify the assumed load_factor in the configuration file as a number between 0 and 1.

This Strategy calculates TSP, PM 10, and PM 2.5 emissions independently for tyre wear, brake wear, and road surface wear. Then it adds up the emissions from all sources to obtain total PM non-exhaust emissions for TSP, PM 10, and PM 2.5.

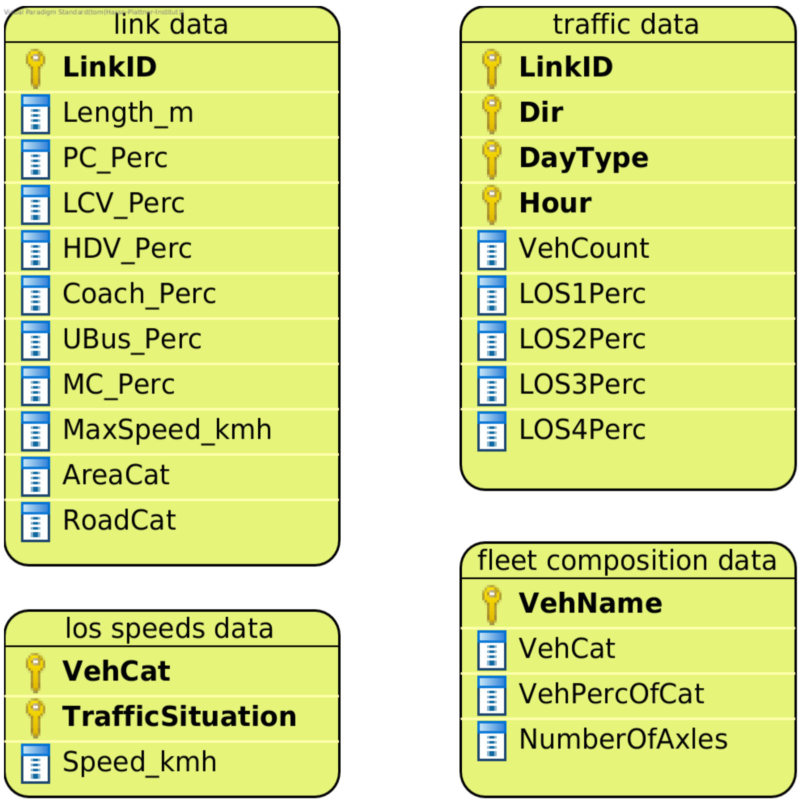
Output of a model run with this Strategy are three csv files:

- TSP emissions
- PM 10 emissions
- PM 2.5 emissions

14.1 Data Requirements

What data the `PMNonExhaustStrategy` requires depends on the `mode` set in the configuration file for the run.

14.1.1 Data requirements for mode `berlin_format`



how-to-read-er

---

link data Just like the link data required for the other Strategies. See *here*.

---

traffic data Just like the traffic data required for the other Strategies. See *here*.

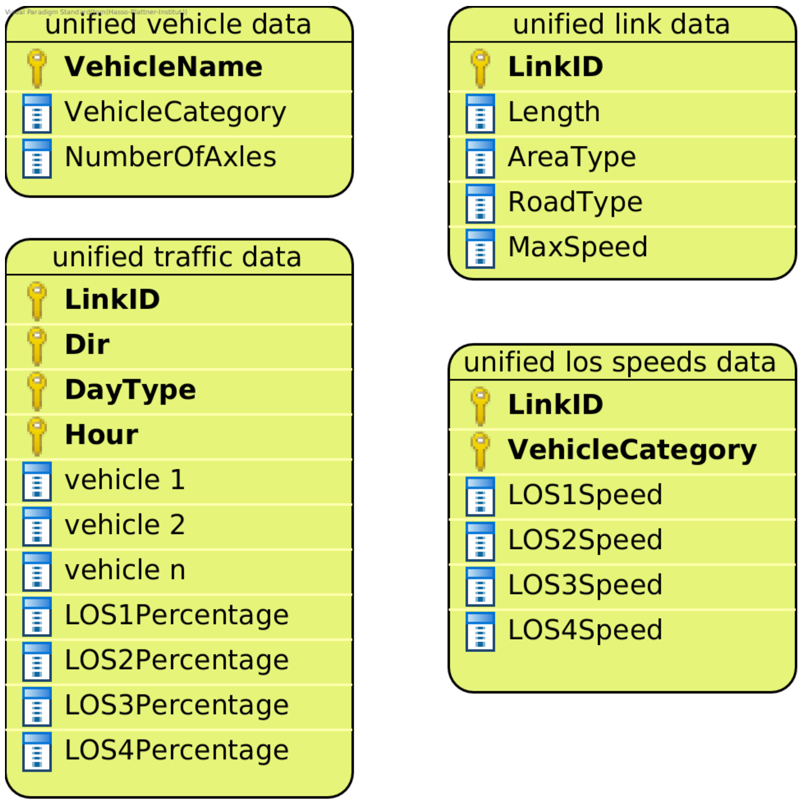
---

fleet composition data Just like the fleet composition data required for the other Strategies. See *here*.

---

los speeds data Just like the los speeds data data required for the `CopertHotStrategy`. See *here*.

14.1.2 Data requirements for mode `yeti_format`



how-to-read-er

---

yeti_format link data Just like the yeti_format link data required for the other Strategies. See *here*.

---

yeti_format vehicle data Just like the yeti_format vehicle data required for the other Strategies. See *here*.

---

yeti_format traffic data Just like the yeti_format traffic data required for the other Strategies. See *here*.

---

yeti_format los speeds data Just like the yeti_format los speeds data data required for the `CopertHotStrategy`. See *here*.

14.2 Supported Pollutants

The only pollutant supported by this Strategy is `PollutantType.PM_Non_Exhaust`. Set it in the config file:

```
pollutants:                    [PollutantType.PM_Non_Exhaust]
```

14.3 What to put in the config.yaml

If you want to use the `PMNonExhaustStrategy` for your calculations, you need to set the following options in your `config.yaml`. Don't forget to add the parameters specified here: *Configuring YETI*

14.3.1 If using mode `berlin_format`:

```
strategy:                       code.pm_non_exhaust_strategy.PMNonExhaustStrategy.
↪PMNonExhaustStrategy
load_berlin_format_data_function:     code.pm_non_exhaust_strategy.load_berlin_format_
↪data.load_pm_non_exhaust_berlin_format_data
load_yeti_format_data_function:   code.pm_non_exhaust_strategy.load_yeti_format_data.
↪load_pm_non_exhaust_yeti_format_data
validation_function:            code.pm_non_exhaust_strategy.validate.validate_pm_non_
↪exhaust_berlin_format_files

berlin_format_link_data:                path/to/link_data.csv
berlin_format_fleet_composition:        path/to/fleet_composition_data.csv
berlin_format_los_speeds:               path/to/los_speeds_data.csv
berlin_format_traffic_data:             path/to/traffic_data.csv

load_factor:                    0.3   # A number between 0 and 1. The assumption about␣
↪the average load of trucks.
```

14.3.2 If using mode `yeti_format`:

```
strategy:                       code.pm_non_exhaust_strategy.PMNonExhaustStrategy.
↪PMNonExhaustStrategy
load_yeti_format_data_function:   code.pm_non_exhaust_strategy.load_yeti_format_data.
↪load_pm_non_exhaust_yeti_format_data
validation_function:            code.pm_non_exhaust_strategy.validate.validate_pm_non_
↪exhaust_yeti_format_files

yeti_format_link_data:            path/to/yeti_format_link_data.csv
yeti_format_vehicle_data:         path/to/yeti_format_vehicle_data.csv
yeti_format_los_speeds:           path/to/yeti_format_los_speed_data.csv
yeti_format_traffic_data:         path/to/yeti_format_traffic_data.csv

load_factor:                    0.3   # A number between 0 and 1. The assumption about␣
↪the average load of trucks.
```

CHAPTER 15

How to support more pollutants, area types, . . .

15.1 About

The docs impose a lot of restrictions on the input data. For example:

1. The number of supported pollutants is quite small.
2. There can only be two directions and two area types in the link data: R/L and 0/1 respectively.
3. There are four day types: 1,2,3,7.
4. The number of vehicle types is limited and their names are set.

This page covers how to work around these restrictions to make YETI work with your data.

15.2 Where do restrictions exist?

The restrictions for the input data are in place because some values are hard-coded. This applies to

- The conversion from berlin_format to yeti_format data for all Strategies.
- Some strategies use hard-coded values for the calculation step, thus imposing restrictions on the input data. Most Strategies don't.

Below are the Strategies that don't impose any restrictions on the input data *for the calculation step*. Remember that these Strategies still impose restrictions when it comes to converting berlin_format data to yeti_format data.

- CopertHotStrategy
- CopertHotFixedSpeedStrategy
- HbefaColdStrategy
- PMNonExhaustStrategy

The following Strategies impose some restrictions on the input data for the calculation step:

- HbefaHotStrategy: Possible road types and area types are hard-coded. Refer to the docs page for HbefaHot-Strategy for more.
- CopertColdStrategy: Possible pollutant types and vehicle categories are hard-coded. Refer to the docs page for CopertColdStrategy for more.

15.3 How to work around the restrictions

If you want to work with data in `berlin_format` you will have to write your own `load_berlin_data_function` that does not enforce the restrictions. The function should convert your data to `yeti_format`. Make sure that the data in `yeti_format` is complete and coherent. Look here for more information: *Support a new data format*

If you are already working with data in yeti_format you don't need to write new data loading functions.

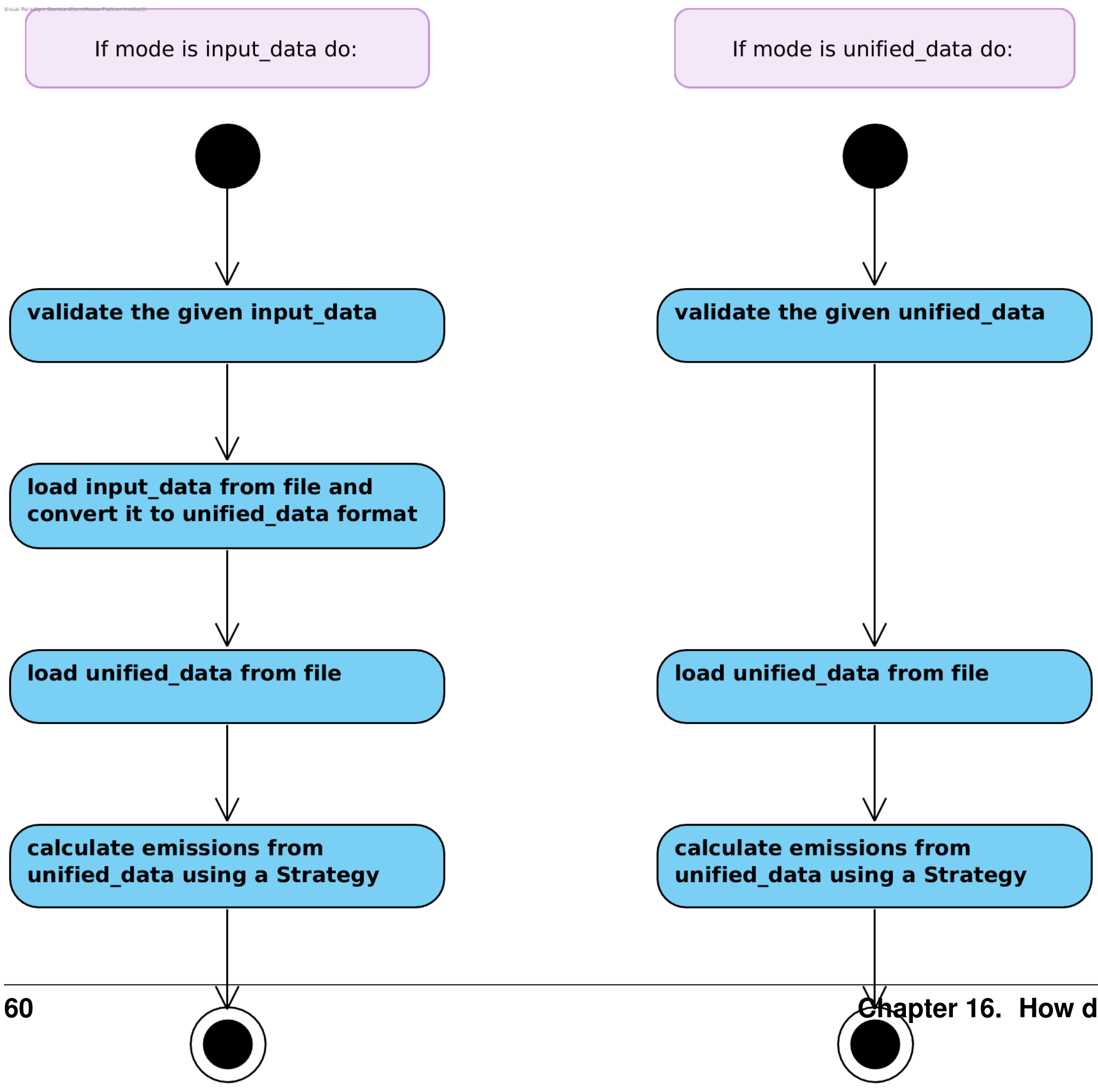
This should be sufficient for the Strategies that enforce restrictions for the calculation step, as listed above. If you want to work with a Strategy that does enforce restrictions for the calculation step, you may need to take additional action:

- If your data does not conflict with the hard-coded values nothing else needs to be done. Refer to the list above to see which values are hard-coded in the Strategies. For example if you want to use different vehicle categories, you don't need to change the HbefaHotStrategy (it only uses hard-coded values for road and area types, not for vehicle categories).
- If your data does conflict with the hard-coded values you need to change code for the Strategy. How exactly you need to change the Strategy depends on the exact format of the data you want to use.

CHAPTER 16

How does it work?

16.1 Overview


If mode is input_data do:
If mode is unified_data do:
validate the given input_data
validate the given unified_data
load input_data from file and convert it to unified_data format
load unified_data from file
load unified_data from file
calculate emissions from unified_data using a Strategy
calculate emissions from unified_data using a Strategy

16.2 Validation

Data validation checks the format of the given files:

- All necessary columns are present
- The mapping files contain all the necessary values and do a correct mapping
- The levels in categorical columns are correct
- Percentage columns contain percentage values

Eather mode can use data validation. Data validation is done by a function designed specifically for data validation. We provide validation functions for some Strategies. If you want to add data validation for your own Strategies and datasets, look here: *Change how data is validated*

Specify the validation function to be used in the config:

```
validation_function: path.to.the.validation_function
```

You may not want to use data validation or no validation function may exists for the desired Strategy. If that is the case simply don't include the above statement in your config to skip validation.

16.3 Load data in berlin_format and convert it to data in yeti_format

This is the part where data in berlin_format is converted to data in yeti_format. It is **only relevant for mode berlin_format**. The conversion is done by a function. Which function you want to use depends on the format of your input data and on which Strategy you are looking to use. We provide functions to load and convert `berlin_format` data for all Strategies.

If you want to work with your own dataset, **you will likely have to write your own function** for this. Look here for information on how to do that: *Support a new data format*

Specify the function to be used in the config:

```
load_berlin_format_data_function: path.to.the.berlin_format_data_loading_function
```

Note that the product of this step are multiple files containing the data in `yeti_format`. The files will be loaded into memory in the next step.

16.4 Load data in yeti_format from file

Load the yeti_format dataset from file. This step is relevant for all modes. It will select the right columns and set the correct data types.

Just like two steps above, this step is done by a function. You can specify the function to be used in the config:

```
load_yeti_format_data_function: path.to.the.yeti_format_data_loading_function
```

If you are working with data that can be converted to fit the yeti_format, you should not have to write your own function for this step. If you do need to adapt how the data in yeti_format is loaded from file, look here: *Change how data in yeti_format is loaded*

16.5 Calculate emissions from data in yeti_format using a Strategy

This step calculates and saves emissions with a Strategy. If you don't know what a Strategy is, look here: *What is a Strategy?*

Specify the Strategy to be used in the config:

```
strategy: path.to.the.Strategy
```

If you want to add a new way to calculate emissions, you need to add a new Strategy to the model. Look here for instructions: *Add a Strategy*

CHAPTER 17

Add a Strategy

Add a new Strategy if you want to support a new methodology/algorithm for emission calculation. Before you add a new Strategy be sure to check that none of the existing Strategies do what you want.

Before you continue reading, make sure you have understood the *high level process* YETI works with.

17.1 Strategies are classes

A Strategy is a Python class that is defined in a `.py` file. Each Strategy needs to contain the function `calculate_emissions`.

```
class MyStrategy:

    def calculate_emissions(self,
                            traffic_and_link_data_row: Dict[str, Any],
                            vehicle_dict: Dict[str, str],
                            pollutants: List[str],
                            **kwargs):

        # Put the emission calculation logic here.
```

The function signature may be confusing for now. That is okay, we will get to the parameters in a second.

Why do we use a class? The answer is that classes can have state. Having the option to store data or results in attributes gives you a lot more flexibility in the implementation.

17.2 Strategies collaborate with data handling functions

The functions `load_berlin_format_data_function` and `load_yeti_format_data_function` (as specified in the config) are used to load the data that is required by the Strategy.

`load_berlin_format_data_function` is a function that reads the data in berlin_format for the Strategy from file, converts it to yeti_format and saves the constructed data in yeti_format to disc. *more*

`load_yeti_format_data_function` has the simple job of reading the required data in yeti_format for the Strategy from disc. *more*

Each Strategy has a corresponding `load_berlin_format_data_function` and `load_yeti_format_data_function`. If you write your own Strategy you may have to also write new data loading functions.

You can access the output of the data loading functions in `calculate_emissions`, as described *here*.

17.3 How are Strategies called?

The Strategy's function `calculate_emissions` is called many times in a model run. The class that calls `calculate_emissions` is the `StrategyInvoker`.

Let's take a look at the parameters of `calculate_emissions`:

17.3.1 `traffic_and_link_data_row`

The `StrategyInvoker` performs an SQL-style inner join on the given yeti_format link data and yeti_format traffic data. It then calls the Strategy's method `calculate_emissions` once per row in the resulting dataframe. The row is passed to `calculate_emissions` as a dictionary.

Let's look at an example. Say your link data and traffic data (belonging to `yeti_format` class) look like this:

link data:

LinkID	AreaType	RoadType	MaxSpeed
42_123	AreaType.Rural	RoadType.MW_City	100
65_485	AreaType.Urban	RoadType.Local	30

traffic_data:

LinkID	Dir	DayType	Hour	PC petrol <1.4L Euro-1	LCV diesel M+N1-I Euro-2
42_123	Dir.L	DayType.MONtoTHU	0	0.32	0.0023999999
65_485	Dir.R	DayType.SUN	9	12.8	0.012

After the SQL-style join we have this dataframe with the traffic and link data:

LinkID	AreaType	RoadType	MaxSpeed	Dir	DayType	Hour	PC petrol <1.4L Euro-1	LCV diesel M+N1-I Euro-2
42_123	AreaType.Rural	RoadType.MW_City	100	Dir.L	DayType.MONtoTHU	0	0.32	0.0023999999
65_485	AreaType.Urban	RoadType.Local	30	Dir.R	DayType.SUN	9	12.8	0.012

The first call to `calculate_emissions` will be with this `traffic_and_link_data_row` :

```
{
"LinkID":    42_123,
"AreaType": "AreaType.Rural",
"RoadType": "RoadType.MW_City",
"MaxSpeed": 100,
"Dir":       "Dir.L",
"DayType":  "DayType.MONtoTHU",
"Hour":      0,
"PC petrol <1.4L Euro-1": 0.32,
"LCV diesel M+N1-I Euro-2": 0.0023999999
}
```

Now the Strategy's job is to take this dictionary and calculate emissions for the two vehicles.

The second call to `calculate_emissions` receives a dictionary with the data from the second traffic and link data row as `traffic_and_link_data_row`.

17.3.2 `vehicle_dict`

This parameter is a dictionary mapping the names of vehicle classes to the corresponding vehicle category. For example `calculate_emissions` may be called with a `vehicle_dict` such as this:

```
{
    "PC petrol <1.4L Euro-1": "VehicleCategory.PC",
    "LCV diesel M+N1-I Euro-2": "VehicleCategory.LCV"
}
```

In `calculate_emissions` you can use the `vehicle_dict` to access the category of a vehicle by its name or use it to iterate over all vehicles. For example:

```python
# MyStrategy.py
class MyStrategy:
    def calculate_emissions(self,
                            traffic_and_link_data_row: Dict[str, Any],
                            vehicle_dict: Dict[str, str],
                            pollutants: List[str],
                            **kwargs):

        ...
        # access the category of a vehicle by its name:
        vehicle_a = ...  # assign some vehicle name to vehicle_a
        category_of_vehicle_a = vehicle_dict[vehicle_a]  # get vehicle_a's category
        ...
        # iterate over all vehicles:
        for vehicle_name, vehicle_category in vehicle_dict.items():
            # do some computation using vehicle_name and/or vehicle_category
        ...
```

The `vehicle_dict` is constructed from the yeti_format vehicle data by the `StrategyInvoker` class.

17.3.3 `pollutants`

A List of Strings. The pollutants as specified in the configuration file.

17.3.4 `**kwargs`

All parameters specified in the configuration file are passed to `calculate_emissions` as keyword arguments. This means that you can use all arguments from the config file in your strategy. You can even define custom config options for your Strategy. An example for using a config parameter in the Strategy:

```
# config.yaml
average_slope:      0.15
```

```
# MyStrategy.py
class MyStrategy:
    def calculate_emissions(self,
                            traffic_and_link_data_row: Dict[str, Any],
                            vehicle_dict: Dict[str, str],
                            pollutants: List[str],
                            **kwargs):

        average_slope = kwargs["average_slope"]
        # You can now use average_slope in the emission calculation.
```

The **return value of the ‘‘load_yeti_format_data_function‘‘** is also passed to `calculate_emissions` as keyword arguments. This means that you can load the required data for the Strategy in the `load_yeti_format_data_function` and then access it in the Strategy. For more details on the `load_yeti_format_data_function` look *here*. An example for using a return value of the `load_yeti_format_data_function` in the Strategy:

```
# function_to_load_yeti_format_data.py
import pandas as pd

def load_yeti_format_data(...):
    ...
    some_pandas_dataframe = pd.read_csv(...) # load the data
    ...
    return {
        "some_dataset": some_pandas_dataframe,
        ...
    }
```

```
# MyStrategy.py
class MyStrategy:
    def calculate_emissions(self,
                            traffic_and_link_data_row: Dict[str, Any],
                            vehicle_dict: Dict[str, str],
                            pollutants: List[str],
                            **kwargs):

        some_dataset = kwargs["some_dataset"]
        # You can now use the dataframe some_dataset for the emission calculation.
```

17.4 What should Strategies return?

As discussed above, the Strategy’s function `calculate_emissions` is called once for each row in a dataframe obtained from joining the link data and the traffic data in an SQL-style fashion.

Each call to `calculate_emissions` should return the emissions for one row in the output emissions dataframe(s) as a dictionary. It is important to note that you should return the emissions for all pollutants.

The `StrategyInvoker` will associate the emissions with the right link ID, day type, hour and direction and save the emissions to disc.

17.4.1 One emissions file per pollutant

Most Strategies want to output a single csv file for each pollutant with emission data for that pollutant. To do so, a Strategy should return one dictionary with emissions per pollutant in the parameter `pollutants` on each call to `calculate_emissions`.

For example:

Let's say `calculate_emissions` was called with this `traffic_and_link_data_row`:

```
{
"LinkID":    42_123,
"AreaType": "AreaType.Rural",
"RoadType": "RoadType.MW_City",
"MaxSpeed": 100,
"Dir":       "Dir.L",
"DayType":   "DayType.MONtoTHU",
"Hour":      0,
"PC petrol <1.4L Euro-1": 0.32,
"LCV diesel M+N1-I Euro-2": 0.0023999999
}
```

Also let's say that the parameter pollutants is `[PollutantType.NOx, PollutantType.CO]`.

The Strategy should then return a dictionary in this format:

```
{
"PollutantType.NOx": {
    "PC petrol <1.4L Euro-1":   some_emissions_value_for_NOx,
    "LCV diesel M+N1-I Euro-2": some_other_emissions_value_for_NOx
    },
"PollutantType.CO": {
    "PC petrol <1.4L Euro-1":   some_emissions_value_for_CO,
    "LCV diesel M+N1-I Euro-2": some_other_emissions_value_for_CO
    }
}
```

This will result in the following rows being added to the emissions dataframes that are saved to disc:

NOx emissions:

LinkID	Dir	DayType	Hour	PC petrol <1.4L Euro-1 LCV	diesel M+N1-I Euro-2
42_123	Dir.L	Day-Type.MONtoTHU	0	some_emissions_value_for_NOx	some_other_emissions_value_for_NOx

CO emissions:

LinkID	Dir	DayType	Hour	PC petrol <1.4L Euro-1 LCV	diesel M+N1-I Euro-2
42_123	Dir.L	Day-Type.MONtoTHU	0	some_emissions_value_for_CO	some_other_emissions_value_for_CO

17.4.2 Multiple emission files per pollutant

Some Strategies want to output multiple emissions files per pollutant. This can be done by adding more dictionaries to the return dictionary.

For example:

Let's say that `calculate_emissions` is called with the same `pollutants` and `traffic_and_link_data` as in the example above. If we want the Strategy to output two emissions files per pollutant, we should return a dictionary like this:

```
{
"PollutantType.NOx_type_A":
    {
    "PC petrol <1.4L Euro-1": some type a emissions value for NOx,
    "LCV diesel M+N1-I Euro-2": some other type a emissions value for NOx,
    ...
    },
"PollutantType.NOx_type_B":
    {
    "PC petrol <1.4L Euro-1": some type b emissions value for NOx,
    "LCV diesel M+N1-I Euro-2": some other type b emissions value for NOx,
    ...
    },
"PollutantType.CO_type_A":
    {
    "PC petrol <1.4L Euro-1": some type a emissions value for CO,
    "LCV diesel M+N1-I Euro-2": some other type a emissions value for CO,
    ...
    },
"PollutantType.CO_type_B":
    {
    "PC petrol <1.4L Euro-1": some type b emissions value for CO,
    "LCV diesel M+N1-I Euro-2": some other type b emissions value for CO,
    ...
    }
}
```

This will create two emissions files per pollutant, one with type a emissions and one with type b emissions. You don't need to stick to the names "type_A" and "type_B". Also you can return as many nested dictionaries as you want to create as many emissions files as you want.

CHAPTER 18

Change how data is validated

Data validation is done by the `validation_function`, as specified in the config. This docs page covers how to write your own `validation_function`.

18.1 How is a `validation_function` called?

A validation function is called with the single argument `kwargs`:

```
def validation_function(**kwargs)

    ...
```

`kwargs` is a Python dictionary that contains all arguments from the config file. For example if the config file contains the line `berlin_format_link_data:  path/to/link_data.csv` the `kwargs` dictionary will contain the key-value pair `"input_link_data":  "path/to/link_data.csv"`.

`kwargs` lets you access the input files that are specified in the config file. You can access these input files and do whatever you want with them. For example you can check if an input file contains all necessary columns.

18.2 Example

```
import pandas as pd
import logging

def validation_function(**kwargs)

    # validate the link data
    # 1. Load link data from file
    link_data_file = kwargs["input_link_data"]
    link_data = pd.read_csv(link_data)
    # 2. Check that link data has the column 'LinkID', 'Length_m', 'MaxSpeed_kmh',
↪'AreaCat', and 'RoadCat'.
    for colname in ["LinkID", "Length_m", "MaxSpeed", "AreaCat", "RoadCat"]:
        if colname not in link_data.columns:
            logging.warning(f"link data is missing the column {colname}")

    # 3. perform other validation operations on the link data
    ...

    # validate other datasets
    ...
```

18.3 Output of the validation_function

What you want the `validation_function` to output is up to you.

The validation functions that we provide print warnings whenever a validation check fails. This means that YETI will keep running even if the validation fails. You can print warnings with `logging.warning("..")`, as shown in the example above.

If you want to stop the YETI run when a validation check fails, you should raise an Error. For example:

```
def validation_function(**kwargs):

    if some_validation_check_does_not_pass():
        raise RuntimeError("validation failed")
```

CHAPTER 19

Support a new data format

This page describes how to adapt YETI to work with a dataset that is not in berlin_format and not in yeti_format. We assume that the dataset you want to use can be converted to fit the yeti_format.

The function `load_berlin_format_data_function` (that you specify in the config file) is responsible for handling input data. It does these four things:

1. Load the input data from file.
2. Convert the input data to fit yeti_format.
3. Save the constructed data in yeti_format to file.
4. Return the locations of the files in yeti_format.

It is likely that you can change the input data loading behaviour without having to write much code yourself. We provide the class `DataLoader` that is extensible and can likely be adapted to your needs without too much effort. More info *below*.

19.1 Example

This as an example `load_berlin_format_data_function`:

```
def load_berlin_format_data_for_my_strategy(**kwargs) -> Dict[str, str]:

    # load the data in berlin_format
    # convert the data to yeti_format
    # save the data in yeti_format to disc

    # then return the locations of the files in yeti_format in a dictionary
    return {
        "yeti_format_link_data": "path/to/yeti_format_link_data.csv",
        "yeti_format_traffic_data": "path/to/yeti_format_traffic_data.csv",
        ...
    }
```

19.2 How is the function called?

The load_berlin_format_data_function is called with a single argument: kwargs. kwargs is a dictionary that contains all arguments from the config file.

For example, if your config file looks like this:

```
# config.yaml
mode:           berlin_format
strategy:                      path.to.Strategy
load_berlin_format_data_function:     path.to.berlin_format_data_load_function
load_yeti_format_data_function:   path.to.yeti_format_data_load_function
output_folder:                 output_folder
berlin_format_link_data:               path/to/link_data.csv
berlin_format_fleet_composition:       path/to/fleet_composition_data.csv
berlin_format_emission_factors:        path/to/emission_factor_data.csv
berlin_format_los_speeds:              path/to/los_speeds_data.csv
berlin_format_traffic_data:            path/to/traffic_data.csv
berlin_format_vehicle_mapping:         path/to/vehicle_mapping_data.csv
```

Then the kwargs dictionary the load_berlin_format_data_function is called with looks like this:

```
{
    "mode":                         "berlin_format",
    "strategy":                     "path.to.Strategy",
    "load_berlin_format_data_function":     "path.to.berlin_format_data_load_function
↪",
    "load_yeti_format_data_function":   "path.to.yeti_format_data_load_function",
    "output_folder":                "output_folder",
    "input_link_data":              "path/to/link_data.csv",
    "input_fleet_composition":      "path/to/fleet_composition_data.csv",
    "input_emission_factors":       "path/to/emission_factor_data.csv",
    "input_los_speeds":             "path/to/los_speeds_data.csv",
    "input_traffic_data":           "path/to/traffic_data.csv",
    "input_vehicle_mapping":        "path/to/vehicle_mapping_data.csv"
}
```

You can work with the kwargs like any other Python dictionary. kwargs gives you access to all configuration arguments. Use them.

19.3 What should the function return?

The load_berlin_format_data_function should return a dictionary containing paths to the files with data in yeti_format that were constructed and saved by the load_berlin_format_data_function.

Example

Let's say the load_berlin_format_data_function creates the four files yeti_format_data/link_data.csv, yeti_format_data/traffic_data.csv, yeti_format_data/vehicle_data.csv, and yeti_format_data/ef_data.csv that contain data in yeti_format. Then the return dictionary of the function should look like this:

```
{
    yeti_format_link_data:      yeti_format_data/link_data.csv
    yeti_format_traffic_data:   yeti_format_data/traffic_data.csv
    yeti_format_vehicle_data:   yeti_format_data/vehicle_data.csv
    yeti_format_ef_data:        yeti_format_data/ef_data.csv
}
```

Note that the `load_yeti_format_data_function` that is specified in the config will be called after the `load_berlin_format_data_function`. The keys in the return dictionary must match the keyword arguments that the `load_yeti_format_data_function` expects as input.

19.4 Use the existing `DataLoader`

As mentioned at the top of the page, there is an easy way to adapt to a new input data format (meaning data in a format different from berlin_format). We provide the class `DataLoader` that is responsible for loading input data from file and converting it to yeti_format. We also provide the function `save_dataframes` to save the data in yeti_format to file and construct the return dictionary.

The `DataLoader` is originally designed to work with data in berlin_format as required by the *CopertHotStrategy* and output data in yeti_format as required by the `CopertHotStrategy`. We will now discuss how to adapt the `DataLoader` to your data requirements.

There are two usage scenarios:

1. One of your berlin_format files has a different format.
2. You don't use all the files in yeti_format that are used by the `CopertHotStrategy`.

We will take a detailed look at the two usage scenarios a bit later on the page. For now we want to look at what they have in common:

You need to subclass the `DataLoader` and use the new class in the `load_berlin_format_data_function`. For example:

Let's say you wrote the `MyDataLoader` that extends the `DataLoader` to fit your needs:

```
from code.data_loading.DataLoader import DataLoader

class MyDataLoader(DataLoader):

    ...
```

Now you want to use `MyDataLoader` in the `load_berlin_format_data_function`:

```
from path.to.MyDataLoader import MyDataLoader
from code.strategy_helpers.helpers import save_dataframes

def load_berlin_format_data(**kwargs):

    output_folder = kwargs["output_folder"]
    loader = MyDataLoader(**kwargs)
    data = loader.load_data(use_nh3_ef=False)
    (link_data, vehicle_data, traffic_data, los_speeds_data, emission_factor_data, _)␣
↪= data

    yeti_format_data_file_paths = save_dataframes(
        output_folder,
        {
            "yeti_format_emission_factors": emission_factor_data,
            "yeti_format_los_speeds": los_speeds_data,
            "yeti_format_vehicle_data": vehicle_data,
            "yeti_format_link_data": link_data,
            "yeti_format_traffic_data": traffic_data
        }
    )
    return yeti_format_data_file_paths
```

Now we will take a look at the two usage scenarios mentioned before.

19.4.1 1. One of your berlin_format files has a different format

This means that you will need to change

1. How the input data is read from file.
2. How one or multiple files in yeti_format are constructed.

For this you will need a `DataLoader` subclass so that you can change the behaviour of the `DataLoader`.

1. Change how the input data is read from file.

The class `FileDataLoader` is responsible for loading input data (e.g. data in berlin_format) from file. To change how input data is loaded you should subclass the `FileDataLoader`, override relevant methods and make your `DataLoader` use the new `FileDataLoader`

First, here is **how to subclass the FileDataLoader:**

```
from code.data_loading.FileDataLoader import FileDataLoader

MyFileDataLoader(FileDataLoader):

    ...  # override the method you would like to change
```

Secondly, these are the **methods you can override:**

```
load_link_data_from_file(self)  # override this method to change how berlin_format␣
↪link data is loaded from file
load_fleet_comp_data_from_file(self)  # override this method to change how berlin_
↪format fleet composition data is loaded from file
load_traffic_count_data_from_file(self)  # override this method to change how berlin_
↪format traffic data is loaded from file
load_emission_factor_data_from_file(self)  # override this method to change how␣
↪berlin_format emission factor data is loaded from file
load_los_speeds_data_from_file(self)  # override this method to change how berlin_
↪format los_speeds data is loaded from file
load_vehicle_mapping_data_from_file(self)  # override this method to change how␣
↪berlin_format vehicle mapping data is loaded from file
load_nh3_ef_data_from_file_if_wanted(self, use_nh3_ef)  # override this method to␣
↪change how berlin_format tier 2 NH3 emission factor data is loaded from file
```

The `self` argument to the functions will give you access to these attributes:

```
self.emission_factor_file
self.los_speeds_file
self.fleet_comp_file
self.link_data_file
self.traffic_file
self.vehicle_name_to_category_mapping
self.nh3_ef_file
self.nh3_mapping_file
```

For example, here is how you would change the way that link data is loaded from file:

```
from code.data_loading.FileDataLoader import FileDataLoader

MyFileDataLoader(FileDataLoader):

    def load_link_data_from_file(self):

        link_data_file_location = self.link_data_file
        link_data = ...  # read the link data from file
        return link_data
```

The last thing you need to do is to **make your DataLoader use the new MyFileDataLoader:**

```
from code.data_loading.DataLoader import DataLoader
from path.to.MyFileDataLoader import MyFileDataLoader

class MyDataLoader(DataLoader):

    # override the method load_berlin_format_data
    def load_berlin_format_data(self, use_nh3_ef: bool):
        return MyFileDataLoader(**self.filenames_dict).load_data(use_nh3_ef)
```

2. Change how one or multiple files in yeti_format are constructed

Every yeti_format file is constructed in a dedicated method by the `DataLoader`. To change how a yeti_format file is constructed, override the method that constructs it.

These are the methods that construct data in yeti_format:

```
# load_traffic_data depends on the berlin_format fleet composition data, link data,␣
↪and traffic data
load_traffic_data(self, fleet_comp_data, link_data, traffic_data)

# load_link_data depends on the berlin_format link data
load_link_data(self, link_data: pd.DataFrame)

# load_vehicle_data depends on the berlin_format fleet composition data
load_vehicle_data(self, fleet_comp_data: pd.DataFrame)

# load_emission_factor_data depends on the berlin_format fleet composition data,␣
↪vehicle mapping data,
# emission factor data, NH3 ef data, and NH3 ef mapping data
load_emission_factor_data(self,
                          use_nh3_ef: bool,
                          fleet_comp_data: pd.DataFrame,
                          vehicle_mapping_data: pd.DataFrame,
                          ef_data: pd.DataFrame,
                          nh3_ef_data: pd.DataFrame,
                          nh3_mapping_data: pd.DataFrame) -> Tuple[pd.DataFrame, pd.
↪DataFrame]

# load_los_speeds_data depends on the berlin_format link data and los speeds data
load_los_speeds_data(self, link_data: pd.DataFrame, los_speeds_data: pd.DataFrame)
```

The comments in the code block above show which methods need to be overridden when which input dataset changes format. For example if the input link data changes, you need to override `load_traffic_data`, `load_link_data`, and `load_los_speeds_data`.

For example if your traffic data format changes, you will need to override `load_traffic_data`:

```
from code.data_loading.DataLoader import DataLoader

class MyDataLoader(DataLoader):

    # override the method load_traffic_data
    def load_traffic_data(self, fleet_comp_data, link_data, traffic_data):

        # construct the traffic data in yeti_format
        yeti_format_traffic_data = ...

        return yeti_format_traffic_data
```

19.4.2 2. You don't use all the files in yeti_format that are used by the `CopertHotStrategy`.

If this is the case you should override the method that constructs the yeti_format file that you don't want to use. Let the method return None. For example say you don't want to use emission factor data:

```
from code.data_loading.DataLoader import DataLoader

class MyDataLoader(DataLoader):

    load_emission_factor_data(self,
                              use_nh3_ef: bool,
                              fleet_comp_data: pd.DataFrame,
                              vehicle_mapping_data: pd.DataFrame,
                              ef_data: pd.DataFrame,
                              nh3_ef_data: pd.DataFrame,
                              nh3_mapping_data: pd.DataFrame):

        return None
```

CHAPTER 20

Change how data in yeti_format is loaded

The `load_yeti_format_data_function` specified in a config file is responsible for loading data in yeti_format from disc. Each `load_yeti_format_data_function` is related to a Strategy that works with the output of the function.

20.1 How is `load_yeti_format_data_function` called?

`load_yeti_format_data_function` is called with the single argument `kwargs`. `kwargs` contains all all arguments from the config file and the key-value pairs returned by the `load_berlin_format_data_function` that was called earlier in the run.

```
def load_yeti_format_data(**kwargs):

    # load data in yeti_format from file

    return {
        # return dict with loaded data in yeti_format
    }
```

Example

Say you want to load los speeds data from file, because your Strategy requires los speeds data:

```
import pandas as pd

def load_yeti_format_data(**kwargs):

    los_speeds_data_file = kwargs["yeti_format_los_speeds_data_file"]
    los_speeds_data = pd.read_csv(los_speeds_file)

    ... # load the other relevant datasets

    return {
        "los_speeds_data": los_speeds_data,
        ...
    }
```

You need to make sure that eather the config file or the return dictionary from the `load_berlin_format_data_function` contains the key `yeti_format_los_speeds_data_file`, so that you can access it in the `kwargs`.

20.2 What should the `load_yeti_format_data_function` return?

The function should return a dictionary. The dictionary should contain all dataframes that will be used by the Strategy. Additionally the return dictionary needs to contain the keys `"traffic_data"`, `"link_data"`, and `"vehicle_data"`. For example:

```
def load_yeti_format_data(**kwargs):

    # load data in yeti_format from file

    return {
        "traffic_data": a_data_frame_with_yeti_format_traffic_data,
        "link_data": a_data_frame_with_yeti_format_link_data,
        "vehicle_data": a_data_frame_with_yeti_format_vehicle_data,
        ...
    }
```

CHAPTER 21

How to update the documentation

The documentation for YETI is written in reStructuredText (rst) using Sphinx. Please familiarize yourself with the basics of rst and Sphinx before updating the docs.

Each page of the documentation corresponds to one rst file in the folder `docs/`. You can look at the rst sourcecode of existing documentation pages by clicking on "View page source" in the top right corner of the page.

When writing documentation, you will want to **look at what you created** and see if it renders the way you want it to. To see what the documentation website will look like with your changes follow these steps:

1. Run `make html` on the command line from the folder `docs/`.
2. Open the file `docs/_build/index.html` in your favourite browser.

If you have firefox installed, you can alternatively run `make open` on the command line from the folder `docs/`.

21.1 Update an existing page

To update an existing page, find the rst file that contains the content of the page. You can find the rst files for the user documentation in `docs/user/` and the rst files for the developer documentation in `docs/developer/`.

Once you have found the right file, make you changes. Then commit them to git and push to GitHub. Make sure to merge the changes into the master branch, otherwise they won't be added to the documentation website.

21.2 Add a new page

21.2.1 1. Create an rst file in `docs/`

Create an rst file in the folder `docs/` or one of its subfolders. You can also create your own subfolder.

21.2.2 2. Add content to the rst file

Add the desired content to the rst file you created. Follow this template:

```
Page Title
==========
Description of docs page

Major section 1
---------------
Some text and/or images

Paragraph 1.1
^^^^^^^^^^^^^
Some text and/or images

Major section 2
---------------
...
```

21.2.3 3. Add the new file to index.rst

To add your documentation page to the docs, you need to add it to the file `docs/index.rst`.

Add the path to your rst file to a `toctree` in `docs/index.rst`. The documentation page will be displayed in the documentation subsection that corresponds to the chosen `toctree` ("User documentation" or "Developer documentation"). Note that the path to your rst file needs to be relative to `docs/index.rst` and you should omit the `.rst` postfix in the path to your rst file.

If you want to create a new documentation subsection, you can add a new `toctree` to `docs/index.rst`. Follow this template:

```
.. toctree::
   :maxdepth: 1
   :caption: Name of the new subsection

   path/to/pageA
   path/to/pageB
   ...
```

21.2.4 4. Commit and Push

When you are happy with the new documentation page, commit your changes to git and push them to GitHub. Make sure to merge the changes into the master branch, otherwise they won't be added to the documentation website.

CHAPTER 22

Contributing to YETI

We are open for collaboration, however we have limited resources to review contributions.

Anyhow, all contributions should follow these guidelines:

- Code should comply with the PEP8 style guide as much as possible.
- All new features should be tested. YETI uses the built-in `unittest` module. If you are new to testing in Python, this website is a good starting point: unittest introduction.
- We follow a green build policy. This means that all the tests should succeed on the test server before a Pull Request is merged.