feedinlib Documentation

Release beta

Uwe Krien, oemof developing group

Jun 28, 2017

Contents

Contents:

CHAPTER 1

Getting started

Important: The feedinlib as of version 0.0.12 will receive an API breaking overhaul. After that we'll move to 0.1.x and initially focus on developing the following features:

- a standardized/unified weather object containing standardized names for measured variables and the ability to convert itself to pvlib and windpowerlib compatible dataframes
- easy access to at least one freely available weather dataset and
- if possible a unified interface to access functionality of technically more specialized libraries like pvlib and windpowerlib.

If you want to stay informed follow issue #29.

The feedinlib is designed to calculate feedin timeseries of photovoltaic and wind power plants. It is part of the oemof group but works as a standalone application.

The feedinlib is ready to use but may have some teething troubles. It definitely has a lot of space for further development, new and improved models and nice features.

Table of contents

Introduction

Having weather data sets you can use the feedinlib to calculate the electrical output of common pv or wind power plants. Basic parameters for many manufacturers are provided with the library so that you can start directly using one

of these parameter sets. Of course you are free to add your own parameter set.

The parameter sets for pv modules are provided by Sandia Laboratories and can be found here:

- https://sam.nrel.gov/sites/default/files/sam-library-sandia-modules-2015-6-30.csv
- https://sam.nrel.gov/sites/default/files/sam-library-cec-modules-2015-6-30.csv

The cp-values for the wind turbines are provided by the Reiner Lemoine Institut and can be found here:

- http://vernetzen.uni-flensburg.de/~git/cp_values.csv

If just want to use the feedinlib to calculate pv systems, you should think of using the pvlib directly. At the moment the pv part of the feedinlib is just a high level (easy to use) interface for same pvlib functionalities.

Actual Release

Download/Install: https://pypi.python.org/pypi/feedinlib/

Documentation: http://pythonhosted.org/feedinlib/

Developing Version

Clone/Fork: https://github.com/oemof/feedinlib

Documentation: http://feedinlib.readthedocs.org/en/latest/

As the feedinlib is part of the oemof developer group we use the same developer rules: http://oemof.readthedocs.io/en/stable/developer_notes.html

Installation

Using the Feedinlib

So far, the feedinlib is mainly tested on python 3.4 but seems to work down to 2.7.

Install the feedinlib using pip3 (or pip2).

```
sudo pip3 install feedinlib
```

Developing the Feedinlib

If you have push rights clone this repository to your local system.

```
git clone git@github.com:oemof/feedinlib.git
```

If you do not have push rights, fork the project at github, clone your personal fork to your system and send a pull request.

If the project is cloned you can install it using pip3 (or pip2) with the -e flag. Using this installation, every change is applied directly.

```
sudo pip3 install -e <path/to/the/feedinlib/root/dir>
```

Optional Packages

To see the plots of the example file one should install the matplotlib package.

Matplotlib can be installed using pip but some Linux users reported that it is easier and more stable to use the pre-built packages of your Linux distribution.

http://matplotlib.org/users/installing.html

Example

Download the example file and execute it:

http://vernetzen.uni-flensburg.de/~git/feedinlib_example.py

Basic Usage

You need three steps to get a time series.

Warning

Be accurate with the units. In the example all units are given without a prefix.

- pressure [Pa]
- wind speed [m/s]
- irradiation [W/m^2]
- peak power [W]
- installed capacity [W]
- nominal power [W]
- area [m^2]

You can also use kW instead of W but you have to make sure that all units change in the same way.

1. Initialise your Turbine or Module

To initialise your specific module or turbine you need a dictionary that contains your basic parameters.

The most import parameter is the name of the module or turbine to get technical parameters from the provided libraries.

The other parameters are related to location of the plant like orientation of the pv module or the hub height of the wind turbine. The existing models need the following parameters:

Wind Model

- h_hub: height of the hub in meters
- d_rotor: diameter of the rotor in meters
- wind_conv_type: Name of the wind converter according to the list in the csv file

PV Model

- azimuth: Azimuth angle of the pv module in degree
- tilt: Tilt angle of the pv module in degree
- module_name: According to the sandia module library (see the link above)
- albedo: Albedo value

```
your_wind_turbine = plants.WindPowerPlant(model=SimpleWindModel, **your_parameter_set)
your_pv_module = plants.Photovoltaic(model=PvlibBased, **your_parameter_set)
```

If you do not pass a model the default model is used. So far we only have one model, so the follwing lines will have the same effect than the lines above.

```
your_wind_turbine = plants.WindPowerPlant(**your_parameter_set)
your_pv_module = plants.Photovoltaic(**your_parameter_set)
```

2. Initialise a weather object

A weather object contains one weather data set and all its necessary meta data. You can define it passing all the information from your weather data source to the FeedinWeatehr class.

```
my_weather_a = weather.FeedinWeather(
    data=my_weather_pandas_DataFrame,
    timezone='Continent/City',  # e.g. Europe/Berlin or America/Caracas
    latitude=x,  # float
    longitude=y,  # float
    data_heigth=coastDat2  # Dictionary, for the data heights (see below).
    )
```

Depending on the model you do not need all of the optional parameters. For example the standard wind model does not need the longitude. If the DataFrame has a full time index with a time zone you don't have to set the time zone.

For wind and pv calculations the DataFrame needs to have radiation, temperature and wind speed for the pv model and pressure, wind speed, temperature and the roughness length for the wind model.

The data_height dictionary should be of the following form.

```
coastDat2 = {
    'dhi': 0,
    'dirhi': 0,
    'pressure': 0,
    'temp_air': 2,
    'v_wind': 10,
    'Z0': 0}
```

If your DataFrame has different column names you have to rename them. This can easily be done by using a conversion dictionary:

```
name_dc = {
    'your diffuse horizontal radiation': 'dhi',
    'your direct horizontal radiation': 'dirhi',
    'your pressure data set': 'pressure',
    'your ambient temperature': 'temp_air',
    'your wind speed': 'v_wind',
    'your roughness length': 'z0'}
```

```
your_weather_DataFrame.rename(columns=name_dc)
```

3. Get your Feedin Time Series

To get your time series you have to pass the weather object to your model. If you pass only the weather object, you get the electrical output of the turbine or module specified by your parameters. You can use optional parameters to calculated more than one module or turbine.

The possible parameters are *number* and *installed capacity* for wind turbines and *number*, *peak_power* and *area* for pv modules.

```
feedin_series_pv1 = your_pv_module.feedin(weather=my_weather_df)  # One Module
feedin_series_wp1 = your_wind_turbine.feedin(data=my_weather_df, number=5)
```

You always should know the nominal power, area or peak_power of your plant. An area of two square meters (area=2) of a specific module that has an area of 1.5 sqm per module might not be realistic.

4. Using your own model

If you use your own model it is safer to pass a list of the required parameters but you don't have to:

```
own_wind_model = models.YourWindModelClass(required=[parameter1, parameter2])
own_pv_model = models.YourPVModelClass()

your_wind_turbine = plants.WindPowerPlant(model=own_wind_model, **your_parameter_set)
your_pv_module = plants.Photovoltaic(model=own_pv_model, **your_parameter_set)

feedin_series_wp1 = your_wind_turbine.feedin(data=my_weather_df, number=5)
feedin_series_pv1 = your_pv_module.feedin(data=my_weather_df)  # One Module
```

CHAPTER 2

What's New

These are new features and improvements of note in each release

Releases

v0.0.11 (November 22, 2016)

New features

- Using model of windpowerlib instead of internal model. This will be the future of the feedinlib.

Bug fixes

- removed 'vernetzen'-server because it is down

Contributors

- Uwe Krien

v0.0.10 (November 18, 2016)

Other changes

Move wind power calculations to windpowerlib Allow installation of windpowerlib for python versions >3.4 Import requests package instead of urllib5

Contributors

- Uwe Krien
- Stephen Bosch
- Birgit Schachler

v0.0.9 (August 23, 2016)

Bug fixes

- Adapt API due to changes in the pvlib
- Avoid pandas future warning running the pv model

Contributors

- Uwe Krien

v0.0.8 (Mai 2, 2016)

New features

- add a geometry attribute for shapely.geometry objects to the weather class
- add lookup table for the sandia pv modules

Documentation

- add link to the developer rules of oemof

Bug fixes

- Adapt url to sandia's module library

Contributors

- Uwe Krien

v0.0.7 (October 20, 2015)

New features

- add a weather class to define the structure of the weather data input
- add example file to pass your own model class to the feedinlib

Documentation

- correct some typos
- some distribtions are clearer now
- describe the used units

Testing

- add more doctests
- removed obsolete tests

Bug fixes

- does not overwrite class attributes (issue 7)

Other changes

- rename classes to more describing names
- initialisation of a power plant changed (see README for details)

Contributors

- Uwe Krien
- Stephan Günther
- Cord Kaldemeyer

CHAPTER 3

Model Description

So far only two simple models are provided but there will improvements and a larger variety in future version hopefully with the help of the github community.

PV Model

The pv model is based on the pvlib library. http://pvlib-python.readthedocs.org/en/latest/

Solar Position and Incident Angle

To calculate the position of the sun the ephemeris model of the pvlib is used (pvlib.solarposition.get_solarposition). A lot of weather data sets contain the hourly mean of each parameter. The algorithms that calculate the position of the sun will do that for the given time. Therefore the feedinlib calculates the hourly mean of the sun angles based on 5-minute values.

To avoid nan-values the feedinlib cut values that are lower than 0° and higher than 90°.

Base on the position of the sun and the orientation of the module the angle of incident is determined (pvlib.irradiance.aoi).

In Plane Irradiation

The pvlib needs the direct normal irradiation (dni) so the feedinlib uses a geometric equation to calculate it from the direct horizontal irradiation (dirhi) provided by the weather data set and the zenith angle (α_{zenith}).

$$dni = \frac{dirhi}{\sin\left(\pi/2 - \alpha_{zenith}\right)}$$

Small differences between the solar position model and the weather model (or measurement) can cause a big deviation for dni if the sun is near the horizon ($\alpha_{zenith} > 88$). caused by a very small denominator in the equation above. Assuming that the irradiation near the horizon is low and the effect on the in-plane-irradiation is very small, the dni is set to dirhi ($dni = dirhi$) for zenith angles greater than 88°.

To determine the diffuse radiation from the sky the Perez Model is used. Therefore the following functions from the pvlib are used:

- pvlib.irradiance.extraradiation
- pvlib.atmosphere.relativeairmass
- pvlib.irradiance.perez

To calculate the ground reflection the pvlib.irradiance.grounddiffuse is used.

The global in plane function sums up the different fractions (pvlib.irradiance.globalinplane).

Electrical Output

The next step calculate the electrical output is to determine the temperature of the solar cell (pvlib.pvsystem.sapm_celltemp). Knowing the cell temperature the feedinlib uses the Sandia PV Array Performance Model (SAPM) to determine the electrical output of the module (pvlib.pvsystem.sapm).

Wind Model

The wind model is a simple model based on the cp-values of a specific type of a wind turbine. The cp-values are provided by the manufacturer of the wind turbine as a list of cp-values for discrete wind speeds in steps of 0.5 or 1 m/s. The feedinlib makes an linear interpolation of these values to get a continuous cp-curve over the wind speeds.

$$P_{wpp} = \frac{1}{8} \cdot \rho_{air,hub} \cdot d_{rotor}^2 \cdot \pi \cdot v_{wind,hub}^3 \cdot cp\left(v_{wind,hub}\right)$$

with d_{rotor} the diameter of the rotor in meters, $\rho_{air,hub}$ the density of the air at hub height, $v_{wind,hub}$ the wind speed at hub height and cp the cp-values against the wind speed.

The wind speed at hub height is determined by the following equation, assuming a logarithmic wind profile.

$$v_{wind,hub} = v_{wind,data} \cdot \frac{\ln\left(\frac{h_{hub}}{z_0}\right)}{\ln\left(\frac{h_{wind,data}}{z_0}\right)}$$

with $v_{wind,hub}$ the wind speed at the height of the weather model or measurement, h_{hub} the height of the hub and $h_{wind,data}$ the height of the wind speed measurement or the height of the wind speed within the weather model.

The density of the air is calculated assuming a temperature gradient of -6.5 K/km and a pressure gradient of -1/8 hPa/m.

$$T_{hub} = T_{air,data} - 0.0065 \cdot (h_{hub} - h_{T,data})$$

with $T_{air,data}$ the temperature at the height of the weather model or measurement, h_{hub} the height of the hub and $h_{T,Data}$ the height of temperature measurement or the height of the temperature within the weather model.

$$\rho_{air,hub} = \left(p_{data}/100 - (h_{hub} - h_{p,data}) * \frac{1}{8}\right) / \left(2.8706 \cdot T_{hub}\right)$$

with p_{data} the pressure at the height of the weather model or measurement, T_{hub} the temperature of the air at hub height, h_{hub} the height of the hub and $h_{p,data}$ the height of pressure measurement or the height of pressure within the weather model.

Weather Data

The weather data manly used by the oemof developing group is the coastDat2 of the Helmholtz-Zentrum Geesthacht

http://wiki.coast.hzg.de/pages/viewpage.action?pageId=4751533

Due to licence problems we are not allowed to ship the dataset with the feedinlib. Please contact the Helmholtz-Zentrum Geesthacht (HZG), the data set for Europe might be free for non commercial use. If you have a licence agreement with the HZG for the coastDat2 data set we might be able to provide a ready-to-use database dump for a postgis database (about 70 GB).

CHAPTER 4

feedinlib package

Submodules

feedinlib.models module

@author: oemof development group

class feedinlib.models.**Base**(***kwargs*)
Bases: abc.ABC

The base class of feedinlib models.

Parameters **required** (*list of strings, optional*) – Containing the names of the required parameters to use the model.

required
The (names of the) parameters this model requires in order to calculate the feedin.

As this is an abstract property, you have to override it in a subclass so that the model can be instantiated. This forces implementors to make the required parameters for a model explicit, even if they are empty, and gives them a good place to document them.

By default, this property is settable and its value can be specified via and argument on construction. If you want to keep this functionality, simply delegate all calls to the superclass.

class feedinlib.models.**PvlibBased**(***kwargs*)
Bases: *feedinlib.models.Base*

Model to determine the output of a photovoltaik module

The calculation is based on the library pvlib.[1]

Parameters **(list of strings, optional)** (PvlibBased.required) – List of required parameters of the model

[1] pvlib on github

Notes

For more information about the photovoltaic model check the documentation of the pvlib library.

https://readthedocs.org/projects/pvlib-python/

References

Examples

```
>>> from feedinlib import models
>>> pv_model = models.PvlibBased()
```

See also:

Base, *SimpleWindTurbine*

angle_of_incidence(*data*, ***kwargs*)

Determine the angle of incidence using the pvlib aoi funktion.[4]

Parameters

- **data** (*pandas.DataFrame*) – Containing the timeseries of the azimuth and zenith angle
- **tilt** (*float*) – Tilt angle of the pv module (horizontal=0°).
- **azimuth** (*float*) – Azimuth angle of the pv module (south=180°).

Returns Angle of incidence in degrees.

Return type pandas.Series

See also:

solarposition_hourly_mean(), *solarposition()*

References

feedin(***kwargs*)

Feedin time series for the given pv module.

In contrast to *turbine_power_output* it returns just the feedin series instead of the whole DataFrame.

Parameters **see** – *turbine_power_output*

Returns A time series of the power output for the given pv module.

Return type pandas.Series

fetch_module_data(*lib='sandia-modules'*, ***kwargs*)

Fetch the module data from the Sandia Module library

The file is saved in the ~/.oemof folder and loaded from there to save time and to make it possible to work if the server is down.

Parameters **module_name** (*string*) – Name of a pv module from the sam.nrel database[9].

[4] pvlib angle of incidence.
[9] module library.

Returns The necessary module data for the selected module to use the pvlib sapm pv model.[8]

Return type dictionary

Examples

```
>>> from feedinlib import models
>>> pvmodel = models.PvlibBased()
>>> name = 'Yingli_YL210__2008__E__'
>>> print(pvmodel.fetch_module_data(module_name=name).Area)
1.7
```

See also:

pv_module_output()

get_pv_power_output(***kwargs*)

Output of the given pv module. For the theoretical background see the pvlib documentation[11].

Parameters **weather** (*feedinlib.weather.FeedinWeather object*) – Instance of the feedinlib weather object (see class *FeedinWeather* for more details)

Notes

See *method required* for all required parameters of this model.

Returns The DataFrame contains the following new columns: p_pv_norm, p_pv_norm_area and all timeseries calculated before.

Return type pandas.DataFrame

References

See also:

pv_module_output(), *feedin()*

global_in_plane_irradiation(*data*, ***kwargs*)

Determine the global irradiaton on the tilted surface.

This method determines the global irradiation in plane knowing the direct and diffuse irradiation, the incident angle and the orientation of the surface. The method uses the pvlib.irradiance.globalinplane function[5] and some other functions of the pvlib.atmosphere[6] and the pvlib.solarposition[2] module to provide the input parameters for the globalinplane function.

Parameters

- **data** (*pandas.DataFrame*) – Containing the time index of the location and columns with the following timeseries: (dirhi, dhi, zenith, azimuth, aoi)
- **tilt** (*float*) – Tilt angle of the pv module (horizontal=0°).
- **azimuth** (*float*) – Azimuth angle of the pv module (south=180°).

[8] pvlib pv-system.
[11] pvlib documentation.
[5] pvlib globalinplane.
[6] pvlib atmosphere.
[2] pvlib solarposition.

- **albedo** (*float*) – Albedo factor around the module

Returns The DataFrame contains the following new columns: poa_global, poa_diffuse, poa_direct

Return type pandas.DataFrame

References

See also:

solarposition_hourly_mean(), *solarposition()*, angle_of_incidenc()

pv_module_output (*data*, ***kwargs*)

Determine the output of pv-system.

Using the pvlib.pvsystem.sapm function of the pvlib[8].

Parameters

- **module_name** (*string*) – Name of a pv module from the sam.nrel database[9].
- **data** (*pandas.DataFrame*) – Containing the time index of the location and columns with the following timeseries: (temp_air [K], v_wind, poa_global, poa_diffuse, poa_direct, airmass, aoi)
- **method** (*string, optional*) – Method to calulate the position of the sun according to the methods provided by the pvlib function (default: 'ephemeris') 'pvlib.solarposition.get_solarposition'.[10]

Returns The DataFrame contains the following new columns: p_pv_norm, p_pv_norm_area

Return type pandas.DataFrame

References

See also:

global_in_plane_irradiation()

required

The parameters this model requires to calculate a feedin.

In this feedin model the required parameters are:

Modul_name (string) - name of a pv module from the sam.nrel database[12]

Tilt (float) - tilt angle of the pv module (horizontal=0°)

Azimuth (float) - azimuth angle of the pv module (south=180°)

Albedo (float) - albedo factor arround the module

solarposition (*location*, *data*, ***kwargs*)

Determine the position of the sun unsing the time of the time index.

Parameters

- **location** (*pvlib.location.Location*) – A pvlib location object containing the longitude, latitude and the timezone of the location

[10] pvlib get_solarposition.
[12] module library.

- **data** (*pandas.DataFrame*) – Containing the timeseries of the weather data and the time index of the location.
- **method** (*string, optional*) – Method to calulate the position of the sun according to the methods provided by the pvlib function (default: 'ephemeris') 'pvlib.solarposition.get_solarposition'.[2]

Returns The DataFrame contains the following new columns: azimuth, zenith, elevation

Return type pandas.DataFrame

Notes

This method is not used in favour to solarposition_hourly_mean.

Examples

```
>>> import pvlib
>>> import pandas as pd
>>> from feedinlib import models
>>> loc = pvlib.location.Location(52, 13, 'Europe/Berlin')
>>> pvmodel = models.PvlibBased()
>>> data = pd.DataFrame(index=pd.date_range(pd.datetime(2010, 1, 1, 0),
... periods=8760, freq='H', tz=loc.tz))
>>> elevation = pvmodel.solarposition(loc, data).elevation
>>> print(round(elevation[12], 3))
14.968
```

See also:

solarposition_hourly_mean() calculates the position of the sun as an hourly mean.

solarposition_hourly_mean(*location*, *data*, ***kwargs*)

Determine the position of the sun as an hourly mean of all angles above the horizon.

Parameters

- **location** (*pvlib.location.Location*) – A pvlib location object containing the longitude, latitude and the timezone of the location
- **data** (*pandas.DataFrame*) – Containing the time index of the location.
- **method** (*string, optional*) – Method to calulate the position of the sun according to the methods provided by the pvlib function (default: 'ephemeris') 'pvlib.solarposition.get_solarposition'.[2]
- **freq** (*string, optional*) – The time interval to create the hourly mean (default: '5Min').
- **pandas.DataFrame** – The DataFrame contains the following new columns: azimuth, zenith, elevation

Notes

Combining hourly values for irradiation with discrete values for the position of the sun can lead to unrealistic results. Using hourly values for the position minimizes these errors.

References

See also:

solarposition() calculates the position of the sun at a given time

class feedinlib.models.**SimpleWindTurbine**(***kwargs*)
Bases: *feedinlib.models.Base*

Model to determine the output of a wind turbine

Parameters **required** (*list of strings*) – Containing the names of the required parameters to use the model.

Examples

```
>>> from feedinlib import models
>>> required_ls = ['h_hub', 'd_rotor', 'wind_conv_type', 'data_height']
>>> wind_model = models.SimpleWindTurbine(required=required_ls)
```

See also:

Base, *PvlibBased*

feedin(***kwargs*)
Alias for turbine_power_output.

required
The parameters this model requires to calculate a feedin.

In this feedin model the required parameters are:

H_hub (float) - Height of the hub of the wind turbine

D_rotor (float) - 'Diameter of the rotor [m]',

Wind_conv_type (string) - Name of the wind converter type. Use self.get_wind_pp_types() to see a list of all possible wind converters.

feedinlib.powerplants module

@author: oemof developing group

Classes in this module correspond to specific types of powerplants.

Powerplant objects act as data holders for the attributes making up a powerplants specification. These objects should only contain little logic. Computing the actual feedin provided by a powerplant is done using it's *model* attribute.'

class feedinlib.powerplants.**Base**(***attributes*)
Bases: abc.ABC

The base class of feedinlib powerplants.

The most prominent shared functionality between powerplants is the fact that they instantiate their model class upon construction, in order to get a unique model instance for each powerplant instance.

Parameters

- **model** (*A model class or an instance of one*) – If a class (or in general, any instance of `type`) is provided, it is used to create the model instance encapsulating the actual mathematical model used to calculate the feedin provided by this powerplant.

  In any other case, the provided object is used directly. Note though, that a reference to this powerplant is saved in the provided object, so sharing model instances between two powerplant objects is not a good idea, as the second powerplant will overwrite the reference the reference to the first.

  The non-class version is only provided for users who need the extra flexibility of controlling model instantiation and who know what they are doing. In general, you'll want to provide a class for this parameter or just go with the default for the specific subclass you are using.

- ****attributes** – The remaining attributes providing the technical specification of the powerplant. They will be added as regular attributes to this object, with keys as attribute names and initialized to the corresponding values.

  An error is raised if the attributes required by the given model are not contained in this hash.

Raises `AttributeError` – in case the attribute listed in the given model's required attribute are not present in the *attributes* parameter.

See also:

feedinlib.models for and explanation of the model needed to actually calculate the feedin.

feedin(***kwargs*)

Calculates the amount of energy fed in by this powerplant into the energy system.

This method delegates the actual computation to the *feedin()* method of this objects `model` while giving you the opportunity to override some of the inputs used to calculate the feedin.

Parameters ****kwargs** – Keyword arguments. If not specified, all the paramters needed to calculate the feedin are taken from this object. If any keyword argument is present whose key matches one of the parameters needed to calculate the feedin, it takes precedence over a matching attribute of this object.

Returns **feedin** – The feedin provided by this poweplant as a time series represented by a `pandas.DataFrame`.

Return type Pandas dataframe

class feedinlib.powerplants.**Photovoltaic**(*model=<class 'feedinlib.models.PvlibBased'>*, ***attributes*)

Bases: *feedinlib.powerplants.Base*

Photovoltaic objects correspond to powerplants using solar power to generate electricity.

Parameters

- **model** – Used as the *model* parameter for *Base*. Defaults to *feedinlib.models.PvlibBased*.
- ****attributes** (see *Base*) –

feedin(***kwargs*)

class feedinlib.powerplants.**WindPowerPlant**(*model=<class 'feedinlib.models.SimpleWindTurbine'>*, ***attributes*)

Bases: *feedinlib.powerplants.Base*

WindPowerPlant objects correspond to powerplants using wind power to generate electricity.

Parameters

- **model** – Used as the model argument for *Base*. Defaults to *feedinlib.models.SimpleWindTurbine*.
- ****attributes** (See *Base*) –

See also:

feedinlib.models for and explanation of the model needed to actually calculate the feedin.

feedin(***kwargs*)

feedinlib.weather

Created on Fri Oct 9 16:01:02 2015

@author: uwe

class feedinlib.weather.**FeedinWeather**(***kwargs*)

Bases: object

Class, containing all meta informations regarding the weather data set.

Parameters

- **data** (*pandas.DataFrame, optional*) – Containing the time series of the different parameters as columns
- **timezone** (*string, optional*) – Containing the name of the time zone using the naming of the IANA (Internet Assigned Numbers Authority) time zone database[40]
- **longitude** (*float, optional*) – Longitude of the location of the weather data
- **latitude** (*float, optional*) – Latitude of the location of the weather data
- **geometry** (*shapely.geometry object*) – polygon or point representing the zone of the weather data
- **data_height** (*dictionary, optional*) – Containing the heights of the weather measurements or weather model in meters with the keys of the data parameter
- **name** (*string*) – Name of the weather data object

Notes

Depending on the used feedin modell some of the optional parameters might be mandatory.

References

read_feedinlib_csv(*filename*, *overwrite=True*)

Reading a csv-file with a header containg the meta data of the time series.

The header has to contain the time zone and has to end with a blank line. To add data of the data_height dictionary there should be space between the parameter name and the key name (e.g. # data_height v_wind: 10). Further more any number of parameters can be added.

The file should have the following form:

[40] IANA time zone database.

```
# timezone=
# name: NAME
# longitude: xx.xxx
# latitude: yy.yyy
# timezone: Continent/City
# data_height temp_air: zz
# data_height v_wind: vv

,temp_air,v_wind,.....
2010-01-01 00:00:00+01:00,267.599,5.32697,...
2010-01-01 01:00:00+01:00,267.596,5.46199,....
....
```

Parameters

- **filename** (*string*) – The filename with the full path and the suffix of the file.
- **overwrite** (*boolean*) – If False the only class attributes of NoneType will be overwritten with the data of the csv file. If True all class attributes will be overwriten with the data of the csv-file.

Raises `FileNotFoundError` – If the file defined by filename can not be found.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index