pynetcf Documentation

Release 0.4.0

TU Wien

Jan 31, 2023

CONTENTS

This is the documentation of **pynetcf**.

Note: This is the main page of your project's Sphinx documentation. It is formatted in reStructuredText. Add additional pages by creating rst-files in `docs` and adding them to the toctree below. Use then references in order to link them from this page, e.g. *Contributors* and *Changelog*.

It is also possible to refer to the documentation of other Python packages with the Python domain syntax. By default you can reference the documentation of Sphinx, Python, NumPy, SciPy, matplotlib, Pandas, Scikit-Learn. You can add more by extending the `intersphinx_mapping` in your Sphinx's `conf.py`.

The pretty useful extension autodoc is activated by default and lets you include documentation from docstrings. Docstrings can be written in Google style (recommended!), NumPy style and classical style.

CHAPTER

ONE

CONTENTS

1.1 pynetcf

Basic python classes that map to netCDF files on disk written according to the Climate and Forecast metadata conventions

This is a first draft which has a lot of room for improvements, this is especially true for the time series based representations.

1.1.1 Citation

If you use the software in a publication then please cite it using the Zenodo DOI. Be aware that this badge links to the latest package version.

Please select your specific version at https://doi.org/10.5281/zenodo.846767 to get the DOI of that version. You should normally always use the DOI for the specific version of your record in citations. This is to ensure that other researchers can access the exact research artefact you used for reproducibility.

You can find additional information regarding DOI versioning at http://help.zenodo.org/#versioning

1.1.2 Installation

This package should be installable through pip:

```
pip install pynetcf
```

1.1.3 Contribute

We are happy if you want to contribute. Please raise an issue explaining what is missing or if you find a bug. We will also gladly accept pull requests against our master branch for new features or bug fixes.

Development setup

For Development we also recommend a `conda` environment. You can create one including test dependencies and debugger by running `conda env create -f environment.yml`. This will create a new `pynetcf` environment which you can activate by using `source activate pynetcf`.

Guidelines

If you want to contribute please follow these steps:

- Fork the pynetcf repository to your account
- make a new feature branch from the pynetcf master branch
- Add your feature
- Please include tests for your contributions in one of the test directories. We use py.test so a simple function called test_my_feature is enough
- submit a pull request to our master branch

1.1.4 Note

This project has been set up using PyScaffold 4.2.3. For details and usage information on PyScaffold see https://pyscaffold.org/.

1.2 License

1.3 Contributors

- Sebastian Hahn <sebastian.hahn@geo.tuwien.ac.at>
- Christoph Paulik <cpaulik@vandersat.com>

1.4 Changelog

1.4.1 Version 0.4.0

- Replace read_ts, write_ts with read, write
- Replace read_all_ts with read_all
- Replace write_ts_all_loc with write_all
- Pep8 fix

1.4.2 Version 0.3.0

- Code refactor of time series classes
- Add destructor closing files
- Minor documentation update
- Fix warnings in tests
- Update copyright date

1.4.3 Version 0.2.2

- Change numpy.num2date behavior to return datetime.datetime

1.4.4 Version 0.2.1

- Force date type conversion to numpy.datetime64
- Update copyright date

1.4.5 Version 0.2.0

- Package no longer Python 2.7 compatible
- Update pyscaffold v3.2.3
- Remove build_script and changes.md
- Update environment.yml
- Package name changed from pynetCF to pynetcf
- Updating travis settings

1.4.6 Version 0.1.19

- Update readme.
- Remove unnecessary dimensions while reading point data.
- Pin squinx version to fix rtd
- Update netcdf4 requirement to v1.4.2

1.4.7 Version 0.1.18

- Update installation to pyscaffold 2.5.x to fix https://github.com/blue-yonder/pyscaffold/issues/148
- Restrict netcdf4 package to versions <=1.2.8 because of https://github.com/Unidata/netcdf4-python/issues/784

1.4.8 Version 0.1.17

- Allow writing and reading of PointData in append mode.
- Set default filenames for GriddedPointData to include .nc ending.

1.4.9 Version 0.1.16

- Translate RuntimeError of older versions of the netCDF4 library to IOError.
- Avoid race conditions when creating directories for new files.

1.4.10 Version 0.1.15

- Fix bug that lost the datatype during writing of timeseries with pandas > 0.17.1
- Fix Python 3 compability.

1.4.11 Version 0.1.14

- fix bug that read the wrong timeseries if a non existing location id was given.
- fix bug when reading not completely filled files in read_bulk mode in contigous ragged.

1.4.12 Version 0.1.13

- Catch RuntimeError and IOError to be compatible with older netCDF4 versions.
- Fix compression of variables in point_data.

1.4.13 Version 0.1.12

- IndexedRaggedTs are now compatible with numpy record arrays and dictionaries.

1.4.14 Version 0.1.11

- IndexedRaggedTs.write_ts can now write data for multiple grid points at once.
- Add interface to write a complete cell to GriddedNcIndexedRaggedTs

1.4.15 Version 0.1.10

- No changes in functionality
- Fix setup.py for correct installation

1.4.16 Version 0.1.9

- Fix n_loc bug
- Add recarray for point data
- Excluding pandas==0.19.0

1.4.17 Version 0.1.8

- Deprecate pynetcf.time_series.GriddedTs please use pynetcf.time_series.GriddedNcTs in the future. Be aware that the __init__ arguments have changed slightly to path, grid, ioclass.

1.4.18 Version 0.1.7

- Add support of read/write netCDF point data following CF conventions
- Add support for disabling automatic masking during reading. Useful if the data has fill values but needs to be scaled to a datatype that does not support NaN values.

1.4.19 Version 0.1.6

- Add support for disabling automatic scaling in base netCDF4 library.
- Add support for dtype conversion before scaling and offset.

1.4.20 Version 0.1.5

- Add classes for gridded datasets based on pygeobase
- improve test coverage
- make compatible with newest netCDF4 releases
- support read_bulk keyword for all dataset types

1.4.21 Version 0.1.4

- fix open/closing of netCDF file

1.4.22 Version 0.1.2

- fixed issue #9

1.4.23 Version 0.1.3

- fixed issue #10

1.4.24 Version 0.1.1

- fixed issue #4

1.4.25 Version 0.1

- moved netcdf classes out of rs data readers

1.5 pynetcf

1.5.1 pynetcf package

Submodules

pynetcf.base module

Base classes for reading and writing time series and images in NetCDF files using Climate Forecast Metadata Conventions (http://cfconventions.org/).

class pynetcf.base.**Dataset**(*filename*, *name=None*, *file_format='NETCDF4'*, *mode='r'*, *zlib=True*, *complevel=4*, *autoscale=True*, *automask=True*)

Bases: object

NetCDF file wrapper class that makes some things easier

Parameters

- **filename** (*string*) – filename of netCDF file. If already exiting then it will be opened as read only unless the append keyword is set. if the overwrite keyword is set then the file will be overwritten
- **name** (*string, optional*) – will be written as a global attribute if the file is a new file
- **file_format** (*string, optional*) – file format
- **mode** (*string, optional*) – access mode. default "r" "r" means read-only; no data can be modified. "w" means write; a new file is created, an existing file with the

  same name is deleted.

  "a" and "r+" mean append (in analogy with serial files); an existing
  file is opened for reading and writing.

  Appending s to modes w, r+ or a will enable unbuffered shared access to NETCDF3_CLASSIC or NETCDF3_64BIT formatted files. Unbuffered access may be useful even if you don"t need shared access, since it may be faster for programs that don"t access data sequentially. This option is ignored for NETCDF4 and NETCDF4_CLASSIC formatted files.
- **zlib** (*boolean, optional*) – Default True if set netCDF compression will be used
- **complevel** (*int, optional*) – Default 4 compression level used from 1(low compression) to 9(high compression)
- **autoscale** (*bool, optional*) – If disabled data will not be automatically scaled when reading and writing
- **automask** (*bool, optional*) – If disabled data will not be masked during reading. This means Fill Values will be used instead of NaN.

add_global_attr(*name*, *value*)

Add global attribute.

Parameters

- **name** (*str*) – Name.
- **value** (*str or number*) – Value.

append_var(*name*, *data*, ***kwargs*)

append data along unlimited dimension(s) of variable

Parameters

- **name** (*string*) – Name of variable to append to.
- **data** (*numpy.array*) – Numpy array of correct dimension.

Raises

IOError – if appending to variable without unlimited dimension

close()

Flush and close file.

create_dim(*name*, *n*)

Create dimension for NetCDF file. if it does not yet exist

Parameters

- **name** (*str*) – Name of the NetCDF dimension.
- **n** (*int*) – Size of the dimension.

flush()

Flush data to disk.

read_var(*name*)

reads variable from netCDF file

Parameters
name (*string*) – name of the variable

write_var(*name*, *data=None*, *dim=None*, *attr={}*, *dtype=None*, *zlib=None*, *complevel=None*, *chunksizes=None*, ***kwargs*)

Create or overwrite values in a NetCDF variable. The data will be written to disk once flush or close is called

Parameters

- **name** (*str*) – Name of the NetCDF variable.
- **data** (*np.ndarray, optional*) – Array containing the data. if not given then the variable will be left empty
- **dim** (*tuple, optional*) – A tuple containing the dimension names.
- **attr** (*dict, optional*) – A dictionary containing the variable attributes.
- **dtype** (*data type, string or numpy.dtype, optional*) – if not given data.dtype will be used
- **zlib** (*boolean, optional*) – explicit compression for this variable if not given then global attribute is used
- **complevel** (*int, optional*) – explicit compression level for this variable if not given then global attribute is used
- **chunksizes** (*tuple, optional*) – chunksizes can be used to manually specify the HDF5 chunksizes for each dimension of the variable.

exception pynetcf.base.**DatasetError**

Bases: Exception

pynetcf.image module

Image class definition for NetCDF files.

class pynetcf.image.**ArrayStack**(*filename*, *grid=None*, *times=None*, *mode='r'*, *name=''*)

Bases: *OrthoMultiTs*

Class for writing stacks of arrays (1D) into netCDF. Array stacks are basically orthogonal multidimensional array representation netCDF files.

write(*gpi*, *data*)

Write a time series into the imagestack for gpi.

Parameters

- **self** (*type*) – description
- **gpi** (*int or numpy.array*) – grid point indices to write to
- **data** (*dictionary*) – dictionary of int or numpy.array for each variable that should be written shape must be (len(gpi), len(times))

class pynetcf.image.**ImageStack**(*filename*, *grid=None*, *times=None*, *mode='r'*, *name=''*)

Bases: *Dataset*

Class for writing stacks of 2D images into NetCDF.

init_variable(*var*)

Initialize variable.

write(*gpi*, *data*)

Write a time series into the imagestack for gpi.

Parameters

- **self** (*type*) – description
- **gpi** (*int or numpy.array*) – grid point indices to write to
- **data** (*dictionary*) – dictionary of int or numpy.array for each variable that should be written shape must be (len(gpi), len(times))

pynetcf.point_data module

Classes for reading and writing point data in NetCDF files using Climate Forecast Metadata Conventions (http://cfconventions.org/).

class pynetcf.point_data.**GriddedPointData**(**args*, ***kwargs*)

Bases: GriddedBase

GriddedPointData class using GriddedBase class as parent and PointData as i/o class.

to_point_data(*filename*, ***kwargs*)

Re-write gridded point data into single file.

Parameters

filename (*str*) – File name.

class pynetcf.point_data.**PointData**(*filename*, *mode='r'*, *file_format='NETCDF4'*, *zlib=True*, *complevel=4*, *n_obs=None*, *obs_dim='obs'*, *add_dims=None*, *loc_id_var='location_id'*, *time_units='days since 1900-01-01 00:00:00'*, *time_var='time'*, *lat_var='lat'*, *lon_var='lon'*, *alt_var='alt'*, ***kwargs*)

Bases: object

PointData class for reading and writing netCDF files following the CF conventions for point data.

Parameters

- **filename** (*str*) – Filename of netCDF file. If already exiting then it will be opened as read only unless the append keyword is set.
- **mode** (*str, optional*) – access mode. default "r" "r" means read-only; no data can be modified. "w" means write; a new file is created, an existing file with the

  same name is deleted.

  "a" and "r+" mean append (in analogy with serial files); an existing
  file is opened for reading and writing.

  Appending s to modes w, r+ or a will enable unbuffered shared access to NETCDF3_CLASSIC or NETCDF3_64BIT formatted files. Unbuffered access may be useful even if you don"t need shared access, since it may be faster for programs that don"t access data sequentially. This option is ignored for NETCDF4 and NETCDF4_CLASSIC formatted files.
- **zlib** (*boolean, optional*) – If set netCDF compression will be used. Default True
- **complevel** (*int, optional*) – Compression level used from 1(low compression) to 9(high compression). Default: 4
- **n_obs** (*int, optional*) – Number of observations. If None, unlimited dimension will be used. Default: None
- **obs_dim** (*str, optional*) – Observation dimension name. Default: "obs"
- **add_dims** (*dict, optional*) – Additional dimensions. Default: None
- **loc_id_var** (*str, optional*) – Location id variable name. Default: "location id"
- **time_units** (*str, optional*) – Time unit.
- **time_var** (*str, optional*) – Time variable name. Default "time"
- **lat_var** (*str, optional*) – Latitude variable name. Default "lat"
- **lon_var** (*str, optional*) – Longitude variable name. Default: "lon"
- **alt_var** (*str, optional*) – Altitude variable name. Default: "alt"

close()

Close file.

flush()

Flush data.

read(*loc_id*)

Read variable from netCDF file for given location id.

Parameters
loc_id (*int*) – Location id.

Returns
data – Dictionary containing variable names as a key and data as items.

Return type
dict

write(*loc_id*, *data*, *lon=None*, *lat=None*, *alt=None*, *time=None*, ***kwargs*)

Write data for specified location ids.

Parameters

- **loc_id** (*numpy.ndarray*) – Location id.
- **data** (*dict of numpy.ndarray or numpy.recarray*) – Dictionary containing variable names as keys and data as items.
- **lon** (*numpy.ndarray, optional*) – Longitude information. Default: None
- **lat** (*numpy.ndarray, optional*) – Latitude information. Default: None
- **alt** (*numpy.ndarray, optional*) – Altitude information. Default: None
- **time** (*numpy.ndarray, optional*) – Time information. Default: None

pynetcf.time_series module

Abstract class providing an interface for reading and writing time series in NetCDF files using Climate Forecast Metadata Conventions (http://cfconventions.org/).

class pynetcf.time_series.**ContiguousRaggedTs**(*filename*, *n_loc=None*, *n_obs=None*, *obs_loc_lut='row_size'*, *obs_dim_name='obs'*, ***kwargs*)

Bases: *DatasetTs*

Class that represents a Contiguous ragged array representation of time series according to NetCDF CF-conventions 1.6.

Parameters

- **filename** (*string*) – filename of netCDF file. If already exiting then it will be opened as read only unless the append keyword is set. if the overwrite keyword is set then the file will be overwritten
- **n_loc** (*int, optional*) – number of locations that this netCDF file contains time series for only required for new file
- **n_obs** (*int, optional*) – how many observations will be saved into this netCDF file in total only required for new file
- **obs_loc_lut** (*string, optional*) – variable name in the netCDF file that contains the lookup between observations and locations
- **loc_dim_name** (*string, optional*) – name of the location dimension
- **obs_dim_name** (*string, optional*) – name of the observations dimension
- **loc_ids_name** (*string, optional*) – name of variable that has the location id"s stored
- **loc_descr_name** (*string, optional*) – name of variable that has additional location information stored
- **time_units** (*string, optional*) – units the time axis is given in. Default: "days since 1900-01-01 00:00:00"
- **time_var** (*string, optional*) – name of time variable Default: time

- **lat_var** (*string, optional*) – name of latitude variable Default: lat
- **lon_var** (*string, optional*) – name of longitude variable Default: lon
- **alt_var** (*string, optional*) – name of altitude variable Default: alt

read_time(*loc_id*)

Read the time stamps for the given location id in this case it works like a normal time series variable.

Returns
time_var – Time variable.

Return type
np.float64

write(*loc_id*, *data*, *dates*, *loc_descr=None*, *lon=None*, *lat=None*, *alt=None*, *fill_values=None*, *attributes=None*, *dates_direct=False*)

Write time series data, if not yet existing also add location to file.

Parameters

- **loc_id** (*int*) – Location id.
- **data** (*dict*) – Dictionary with variable names as keys and numpy.ndarrays as values.
- **dates** (*numpy.array*) – Array of datetime objects.
- **attributes** (*dict, optional*) – Dictionary of attributes that should be added to the netCDF variables. can also be a dict of dicts for each variable name as in the data dict.
- **dates_direct** (*boolean*) – If true the dates are already converted into floating point number of correct magnitude.

class pynetcf.time_series.**DatasetTs**(*filename*, *n_loc=None*, *loc_dim_name='locations'*, *obs_dim_name='time'*, *loc_ids_name='location_id'*, *loc_descr_name='location_description'*, *time_units='days since 1900-01-01 00:00:00'*, *time_var='time'*, *lat_var='lat'*, *lon_var='lon'*, *alt_var='alt'*, *unlim_chunksize=None*, *read_bulk=False*, *read_dates=True*, ***kwargs*)

Bases: *Dataset*, *ABC*

Abstract class to store common methods for NetCDF time series such as OrthoMulti-, ContiguousRaggedArray- and IndexedRaggedArray-representation. Implemented according to the NetCDF CF-conventions 1.6.

Parameters

- **filename** (*string*) – filename of netCDF file. If already exiting then it will be opened as read only unless the append keyword is set. if the overwrite keyword is set then the file will be overwritten
- **n_loc** (*int, optional*) – number of locations that this netCDF file contains time series for only required for new file
- **loc_dim_name** (*string, optional*) – name of the location dimension
- **obs_dim_name** (*string, optional*) – name of the observations dimension
- **loc_ids_name** (*string, optional*) – name of variable that has the location id"s stored
- **loc_descr_name** (*string, optional*) – name of variable that has additional location information stored
- **time_units** (*string, optional*) – units the time axis is given in. Default: "days since 1900-01-01 00:00:00"

- **time_var** (*string, optional*) – name of time variable Default: time
- **lat_var** (*string, optional*) – name of latitude variable Default: lat
- **lon_var** (*string, optional*) – name of longitude variable Default: lon
- **alt_var** (*string, optional*) – name of altitude variable Default: alt
- **unlim_chunksize** (*int, optional*) – chunksize to use along unlimited dimensions, other chunksizes will be calculated by the netCDF library
- **read_bulk** (*boolean, optional*) – if set to True the data of all locations is read into memory, and subsequent calls to "read" read from the cache and not from disk this makes reading complete files faster#
- **read_dates** (*boolean, optional*) – if false dates will not be read automatically but only on specific request useable for bulk reading because currently the netCDF num2date routine is very slow for big datasets

extend_time(*dates*, *direct=False*)

Extend the time dimension and variable by the given dates

Parameters

- **dates** (*numpy.array of datetime objects or floats*) – Timestamps.
- **direct** (*boolean*) – if true the dates are already converted into floating point number of correct magnitude

get_time_variable_overlap(*dates*)

Figure out if a new date array has a overlap with the already existing time variable.

Return the index of the existing time variable where the new dates should be located.

At the moment this only handles cases where all dates are new or none are new.

Parameters
dates (*list*) – List of datetime objects

Returns
indexes – Array of indexes that overlap

Return type
numpy.ndarray

read(*variables*, *loc_id*, *dates_direct=False*)

reads time series of variables

Parameters

- **variables** (*list or string*) –
- **loc_id** (*int*) – location_id
- **dates_direct** (*boolean, optional*) – if True the dates are read directly from the netCDF file without conversion to datetime

read_all(*loc_id*, *dates_direct=False*)

read a time series of all time series variables at a given location id

Parameters

- **loc_id** (*int*) – id of location, can be a grid point id or some other id
- **dates_direct** (*boolean, optional*) – if True the dates are read directly from the netCDF file without conversion to datetime

Returns
time_series – keys of var and time with numpy.arrays as values

Return type
dict

read_dates(*loc_id*)

Read time stamps and convert them.

read_time(*loc_id*)

Read the time stamps for the given location id in this case the location id is irrelevant since they all have the same timestamps

abstract write(*loc_id*, *data*, *dates*, *loc_descr=None*, *lon=None*, *lat=None*, *alt=None*, *fill_values=None*, *attributes=None*, *dates_direct=False*)

Write time series data, if not yet existing also add location to file for this data format it is assumed that in each write/append cycle the same amount of data is added.

Parameters

- **loc_id** (*int*) – Location id.
- **data** (*dict*) – Dictionary with variable names as keys and numpy.ndarrays as values.
- **dates** (*numpy.ndarray*) – Array of datetime objects.
- **attributes** (*dict, optional*) – Dictionary of attributes that should be added to the netCDF variables. can also be a dict of dicts for each variable name as in the data dict.
- **dates_direct** (*boolean*) – If true the dates are already converted into floating point number of correct magnitude.

write_all(*loc_ids*, *data*, *dates*, *loc_descrs=None*, *lons=None*, *lats=None*, *alts=None*, *fill_values=None*, *attributes=None*, *dates_direct=False*)

Write time series data in bulk, for this the user has to provide a 2D array with dimensions (self.nloc, dates) that is filled with the time series of all grid points in the file.

Parameters

- **loc_ids** (*numpy.ndarray*) – location ids along the first axis of the data array
- **data** (*dict*) – dictionary with variable names as keys and 2D numpy.arrays as values
- **dates** (*numpy.ndarray*) – Array of datetime objects with same size as second dimension of data arrays.
- **attributes** (*dict, optional*) – Dictionary of attributes that should be added to the netCDF variables. can also be a dict of dicts for each variable name as in the data dict.
- **dates_direct** (*boolean*) – If true the dates are already converted into floating point number of correct magnitude

class pynetcf.time_series.**GriddedNcContiguousRaggedTs**(**args*, ***kwargs*)

Bases: *GriddedNcTs*

class pynetcf.time_series.**GriddedNcIndexedRaggedTs**(**args*, ***kwargs*)

Bases: *GriddedNcTs*

write_cell(*cell*, *gpi*, *data*, *datefield*)

Write complete data set into cell file.

Parameters

- **cell** (*int*) – Cell number.
- **gpi** (*numpy.ndarray*) – Location ids.
- **data** (*dict or numpy record array*) – dictionary with variable names as keys and numpy.arrays as values
- **datefield** (*string*) – field in the data dict that contains dates in correct format

class pynetcf.time_series.**GriddedNcOrthoMultiTs**(**args*, ***kwargs*)

Bases: *GriddedNcTs*

class pynetcf.time_series.**GriddedNcTs**(**args*, ***kwargs*)

Bases: GriddedTsBase

class pynetcf.time_series.**IndexedRaggedTs**(*filename*, *n_loc=None*, *obs_loc_lut='locationIndex'*, ***kwargs*)

Bases: *DatasetTs*

Class that represents a Indexed ragged array representation of time series according to NetCDF CF-conventions 1.6.

read_time(*loc_id*)

Read the time stamps for the given location id in this case it works like a normal time series variable.

Returns
time_var – Time variable.

Return type
np.float64

write(*loc_id*, *data*, *dates*, *loc_descr=None*, *lon=None*, *lat=None*, *alt=None*, *fill_values=None*, *attributes=None*, *dates_direct=False*)

write time series data, if not yet existing also add location to file

Parameters

- **loc_id** (*int or numpy.ndarray*) – location id, if it is an array the location ids have to match the data in the data dictionary and in the dates array. In this way data for more than one point can be written into the file at once.
- **data** (*dict or numpy.recarray*) – dictionary with variable names as keys and numpy.arrays as values
- **dates** (*numpy.array*) – array of datetime objects
- **attributes** (*dict, optional*) – dictionary of attributes that should be added to the netCDF variables. can also be a dict of dicts for each variable name as in the data dict.
- **dates_direct** (*boolean*) – if true the dates are already converted into floating point number of correct magnitude

class pynetcf.time_series.**OrthoMultiTs**(*filename*, *n_loc=None*, *loc_dim_name='locations'*, *obs_dim_name='time'*, *loc_ids_name='location_id'*, *loc_descr_name='location_description'*, *time_units='days since 1900-01-01 00:00:00'*, *time_var='time'*, *lat_var='lat'*, *lon_var='lon'*, *alt_var='alt'*, *unlim_chunksize=None*, *read_bulk=False*, *read_dates=True*, ***kwargs*)

Bases: *DatasetTs*

Implementation of the Orthogonal multidimensional array representation of time series according to the NetCDF CF-conventions 1.6.

Parameters

- **filename** (*string*) – filename of netCDF file. If already exiting then it will be opened as read only unless the append keyword is set. if the overwrite keyword is set then the file will be overwritten
- **n_loc** (*int, optional*) – number of locations that this netCDF file contains time series for only required for new file
- **loc_dim_name** (*string, optional*) – name of the location dimension
- **obs_dim_name** (*string, optional*) – name of the observations dimension
- **loc_ids_name** (*string, optional*) – name of variable that has the location id"s stored
- **loc_descr_name** (*string, optional*) – name of variable that has additional location information stored
- **time_units** (*string, optional*) – units the time axis is given in. Default: "days since 1900-01-01 00:00:00"
- **time_var** (*string, optional*) – name of time variable Default: time
- **lat_var** (*string, optional*) – name of latitude variable Default: lat
- **lon_var** (*string, optional*) – name of longitude variable Default: lon
- **alt_var** (*string, optional*) – name of altitude variable Default: alt
- **unlim_chunksize** (*int, optional*) – chunksize to use along unlimited dimensions, other chunksizes will be calculated by the netCDF library
- **read_bulk** (*boolean, optional*) – if set to True the data of all locations is read into memory, and subsequent calls to "read" read from the cache and not from disk this makes reading complete files faster#
- **read_dates** (*boolean, optional*) – if false dates will not be read automatically but only on specific request useable for bulk reading because currently the netCDF num2date routine is very slow for big datasets

write(*loc_id*, *data*, *dates*, *loc_descr=None*, *lon=None*, *lat=None*, *alt=None*, *fill_values=None*, *attributes=None*, *dates_direct=False*)

Write time series data, if not yet existing also add location to file for this data format it is assumed that in each write/append cycle the same amount of data is added.

Parameters

- **loc_id** (*int*) – Location id.
- **data** (*dict*) – Dictionary with variable names as keys and numpy.ndarrays as values.
- **dates** (*numpy.ndarray*) – Array of datetime objects.
- **attributes** (*dict, optional*) – Dictionary of attributes that should be added to the netCDF variables. can also be a dict of dicts for each variable name as in the data dict.
- **dates_direct** (*boolean*) – If true the dates are already converted into floating point number of correct magnitude.

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

T

W