

---

# **pynetcf Documentation**

***Release 0.2.2***

**TU Wien**

**Jun 27, 2022**



---

## Contents

---

<b>1</b>	<b>How to use</b>	<b>3</b>
<b>2</b>	<b>Contents</b>	<b>5</b>
2.1	License . . . . .	5
2.2	Contributors . . . . .	5
2.3	Changelog . . . . .	6
2.4	pynetcf . . . . .	9
<b>3</b>	<b>Indices and tables</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>



This is the documentation of **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.



# CHAPTER 1

---

## How to use

---

For now please see the tests for examples on how to use the classes.





### 2.1 License

MIT License

Copyright (c) 2020 TU Wien

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### 2.2 Contributors

- Sebastian Hahn <[sebastian.hahn@geo.tuwien.ac.at](mailto:sebastian.hahn@geo.tuwien.ac.at)>
- Christoph Paulik <[cpaulik@vandersat.com](mailto:cpaulik@vandersat.com)>

## 2.3 Changelog

### 2.3.1 Version 0.2.2

- Change numpy.num2date behavior to return datetime.datetime

### 2.3.2 Version 0.2.1

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

### 2.3.3 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

### 2.3.4 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

### 2.3.5 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>

### 2.3.6 Version 0.1.17

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

### 2.3.7 Version 0.1.16

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

### 2.3.8 Version 0.1.15

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

### 2.3.9 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 contiguous ragged.

### 2.3.10 Version 0.1.13

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

### 2.3.11 Version 0.1.12

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

### 2.3.12 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

### 2.3.13 Version 0.1.10

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

### 2.3.14 Version 0.1.9

- Fix n\_loc bug
- Add recarray for point data
- Excluding pandas==0.19.0

### 2.3.15 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.

### **2.3.16 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.

### **2.3.17 Version 0.1.6**

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

### **2.3.18 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

### **2.3.19 Version 0.1.4**

- fix open/closing of netCDF file

### **2.3.20 Version 0.1.2**

- fixed issue #9

### **2.3.21 Version 0.1.3**

- fixed issue #10

### **2.3.22 Version 0.1.1**

- fixed issue #4

### **2.3.23 Version 0.1**

- moved netcdf classes out of rs data readers

## 2.4 pynetcf

### 2.4.1 pynetcf package

#### Submodules

#### pynetcf.base module

Base classes for reading and writing time series and images in NetCDF files according to the Climate Forecast Meta-data 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 existing 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)
```

```
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()`

`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()`

`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, chunk-sizes=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

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

Bases: `pynetcf.time_series.OrthoMultiTs`

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

`write_ts(gpi, data)`

write a time series into the imagestack at the given 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: *pynetcf.base.Dataset*

Class for writing stacks of 2D images into netCDF.

**init\_variable** (*var*)

**write\_ts** (*gpi, data*)

write a time series into the imagestack at the given 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

Module for reading and writing point data in NetCDF format according to the Climate Forecast Metadata Conventions (<http://cfconventions.org/>).

**class** pynetcf.point\_data.**GriddedPointData** (*\*args, \*\*kwargs*)

Bases: *pygeobase.io\_base.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 existing 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

Classes for reading and writing time series in NetCDF files according to the 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: `pynetcf.time_series.OrthoMultiTs`

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 existing 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\_ts** (*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.**GriddedNcContiguousRaggedTs** (\*args, \*\*kwargs)  
Bases: *pynetcf.time\_series.GriddedNcTs*

**class** pynetcf.time\_series.**GriddedNcIndexedRaggedTs** (\*args, \*\*kwargs)  
Bases: *pynetcf.time\_series.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: *pynetcf.time\_series.GriddedNcTs*

**class** pynetcf.time\_series.**GriddedNcTs** (\*args, \*\*kwargs)  
Bases: *pygeobase.io\_base.GriddedTsBase*

**class** pynetcf.time\_series.**IndexedRaggedTs** (*filename, n\_loc=None, obs\_loc\_lut='locationIndex', \*\*kwargs*)  
Bases: *pynetcf.time\_series.ContiguousRaggedTs*

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

**write\_ts** (*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: `pynetcf.base.Dataset`

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 existing 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\_ts 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*) –
- **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 an 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** *np.ndarray*

**read\_all\_ts** (*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

**read\_ts** (*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

**write\_ts** (*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\_ts\_all\_loc** (*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

## Module contents



## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### p

- `pynetcf`, [17](#)
- `pynetcf.base`, [9](#)
- `pynetcf.image`, [10](#)
- `pynetcf.point_data`, [11](#)
- `pynetcf.time_series`, [13](#)



## A

add\_global\_attr() (pynetcf.base.Dataset method), 9  
 append\_var() (pynetcf.base.Dataset method), 9  
 ArrayStack (class in pynetcf.image), 10

## C

close() (pynetcf.base.Dataset method), 10  
 close() (pynetcf.point\_data.PointData method), 12  
 ContiguousRaggedTs (class in pynetcf.time\_series), 13  
 create\_dim() (pynetcf.base.Dataset method), 10

## D

Dataset (class in pynetcf.base), 9  
 DatasetError, 10

## E

extend\_time() (pynetcf.time\_series.OrthoMultiTs method), 15

## F

flush() (pynetcf.base.Dataset method), 10  
 flush() (pynetcf.point\_data.PointData method), 12

## G

get\_time\_variable\_overlap()  
     (pynetcf.time\_series.OrthoMultiTs method), 15  
 GriddedNcContiguousRaggedTs (class in  
     pynetcf.time\_series), 14  
 GriddedNcIndexedRaggedTs (class in  
     pynetcf.time\_series), 14  
 GriddedNcOrthoMultiTs (class in pynetcf.time\_series),  
     14  
 GriddedNcTs (class in pynetcf.time\_series), 14  
 GriddedPointData (class in pynetcf.point\_data), 11

## I

ImageStack (class in pynetcf.image), 11  
 IndexedRaggedTs (class in pynetcf.time\_series), 14  
 init\_variable() (pynetcf.image.ImageStack method), 11

## O

OrthoMultiTs (class in pynetcf.time\_series), 14

## P

PointData (class in pynetcf.point\_data), 11  
 pynetcf (module), 17  
 pynetcf.base (module), 9  
 pynetcf.image (module), 10  
 pynetcf.point\_data (module), 11  
 pynetcf.time\_series (module), 13

## R

read() (pynetcf.point\_data.PointData method), 12  
 read\_all\_ts() (pynetcf.time\_series.OrthoMultiTs method),  
     16  
 read\_dates() (pynetcf.time\_series.OrthoMultiTs method),  
     16  
 read\_time() (pynetcf.time\_series.ContiguousRaggedTs  
     method), 13  
 read\_time() (pynetcf.time\_series.OrthoMultiTs method),  
     16  
 read\_ts() (pynetcf.time\_series.OrthoMultiTs method), 16  
 read\_var() (pynetcf.base.Dataset method), 10

## T

to\_point\_data() (pynetcf.point\_data.GriddedPointData  
     method), 11

## W

write() (pynetcf.point\_data.PointData method), 12  
 write\_cell() (pynetcf.time\_series.GriddedNcIndexedRaggedTs  
     method), 14  
 write\_ts() (pynetcf.image.ArrayStack method), 10  
 write\_ts() (pynetcf.image.ImageStack method), 11  
 write\_ts() (pynetcf.time\_series.ContiguousRaggedTs  
     method), 13  
 write\_ts() (pynetcf.time\_series.IndexedRaggedTs  
     method), 14  
 write\_ts() (pynetcf.time\_series.OrthoMultiTs method), 16

`write_ts_all_loc()` (pynetcf.time\_series.OrthoMultiTs  
method), [16](#)

`write_var()` (pynetcf.base.Dataset method), [10](#)