h5netcdf: A Python interface for the netCDF4 file-format based on h5py

Contents

Release: 1.7.3
Date: Oct 31, 2025 Print to PDF

h5netcdf is an open source project and Python package that provides an interface for the netCDF4 file-format that reads and writes local or remote HDF5 files directly via h5py or h5pyd, without relying on the Unidata netCDF library.

Why h5netcdf?

- It has one less binary dependency (netCDF C). If you already have h5py installed, reading netCDF4 with h5netcdf may be much easier than installing netCDF4-Python.
- We've seen occasional reports of better performance with h5py than netCDF4-python, though in many cases performance is identical. For one workflow, h5netcdf was reported to be almost **4x faster** than netCDF4-python.
- Anecdotally, HDF5 users seem to be unexcited about switching to netCDF – hopefully this will convince them that netCDF4 is actually quite sane!

- Finally, side-stepping the netCDF C library (and Cython bindings to it) gives us an easier way to identify the source of performance issues and bugs in the netCDF libraries/specification.

Install

Ensure you have a recent version of h5py installed (I recommend using conda or the community effort conda-forge). At least version 3.0 is required. Then:

```
$ pip install h5netcdf
```

Or if you are already using conda:

```
$ conda install h5netcdf
```

Note:

From version 1.2. h5netcdf tries to align with a nep29-like support policy with regard to it's upstream dependencies.

Usage

h5netcdf has two APIs, a new API and a legacy API. Both interfaces currently reproduce most of the features of the netCDF interface, including the ability to write NETCDF4 and NETCDF4_CLASSIC formatted files. Support for operations that rename or delete existing objects is still missing, and patches would be very welcome.

New API

The new API supports direct hierarchical access of variables and groups. Its design is an adaptation of h5py to the netCDF data model. For example:

```python
import h5netcdf
import numpy as np

with h5netcdf.File("mydata.nc", "w") as f:
    # set dimensions with a dictionary
    f.dimensions = {"x": 5}
    # and update them with a dict-like interface
    # f.dimensions['x'] = 5
    # f.dimensions.update({'x': 5})

    v = f.create_variable("hello", ("x",), float)
    v[:] = np.ones(5)

    # you don't need to create groups first
    # you also don't need to create dimensions first if you supply data
    # with the new variable
    v = f.create_variable("/grouped/data", ("y",), data=np.arange(10))

    # access and modify attributes with a dict-like interface
    v.attrs["foo"] = "bar"

    # you can access variables and groups directly using a hierarchical
    # keys like h5py
    print(f["/grouped/data"])

    # add an unlimited dimension
    f.dimensions["z"] = None
    # explicitly resize a dimension and all variables using it
    f.resize_dimension("z", 3)
```

Notes:

- Automatic resizing of unlimited dimensions with array indexing is not available.
- Dimensions need to be manually resized with `Group.resize_dimension(dimension, size)`.
- Arrays are returned padded with `fillvalue` (taken from underlying hdf5 dataset) up to current size of variable's dimensions. The behaviour is equivalent to netCDF4-python's `Dataset.set_auto_mask(False)`.

Legacy API

The legacy API is designed for compatibility with netCDF4-python. To use it, import `h5netcdf.legacyapi`:

```
import h5netcdf.legacyapi as netCDF4

# everything here would also work with this instead:
# import netCDF4
import numpy as np

with netCDF4.Dataset("mydata.nc", "w") as ds:
    ds.createDimension("x", 5)
    v = ds.createVariable("hello", float, ("x",))
    v[:] = np.ones(5)

    g = ds.createGroup("grouped")
    g.createDimension("y", 10)
    g.createVariable("data", "i8", ("y",))
    v = g["data"]
    v[:] = np.arange(10)
    v.foo = "bar"
    print(ds.groups["grouped"].variables["data"])
```

The legacy API is designed to be easy to try-out for netCDF4-python users, but it is not an exact match. Here is an incomplete list of functionality we don't include:

- Utility functions `chartostring`, `num2date`, etc., that are not directly necessary for writing netCDF files.
- h5netcdf variables do not support automatic masking or scaling (e.g., of values matching the `_FillValue` attribute). We prefer to leave this functionality to client libraries (e.g., xarray), which can implement their exact desired scaling behavior. Nevertheless arrays are returned padded with `fillvalue` (taken from underlying hdf5 dataset) up to current size of variable's dimensions. The behaviour is equivalent to netCDF4-python's `Dataset.set_auto_mask(False)`.

Invalid netCDF files

h5py implements some features that do not (yet) result in valid netCDF files:

- **Data types:**
  - Booleans
  - Reference types
- **Arbitrary filters:**
  - Scale-offset filters

By default [1], h5netcdf will not allow writing files using any of these features, as files with such features are not readable by other netCDF tools.

However, these are still valid HDF5 files. If you don't care about netCDF compatibility, you can use these features by setting `invalid_netcdf=True` when creating a file:

```
# avoid the .nc extension for non-netcdf files
f = h5netcdf.File("mydata.h5", invalid_netcdf=True)
...

# works with the legacy API, too, though compression options are not exposed
ds = h5netcdf.legacyapi.Dataset("mydata.h5", invalid_netcdf=True)
...
```

In such cases the *_NCProperties* attribute will not be saved to the file or be removed from an existing file. A warning will be issued if the file has *.nc*-extension.

Footnotes

[1] h5netcdf we will raise `h5netcdf.CompatibilityError`.

Decoding variable length strings

h5py 3.0 introduced new behavior for handling variable length string. Instead of being automatically decoded with UTF-8 into NumPy arrays of `str`, they are required as arrays of `bytes`.

The legacy API preserves the old behavior of h5py (which matches netCDF4), and automatically decodes strings.

The new API matches h5py behavior. Explicitly set `decode_vlen_strings=True` in the `h5netcdf.File` constructor to opt-in to automatic decoding.

Datasets with missing dimension scales

By default [2] h5netcdf raises a `ValueError` if variables with no dimension scale associated with one of their axes are accessed. You can set `phony_dims='sort'` when opening a file to let h5netcdf invent phony dimensions according to netCDF behaviour.

```
# mimic netCDF-behaviour for non-netcdf files
f = h5netcdf.File("mydata.h5", mode="r", phony_dims="sort")
...
```

Note, that this iterates once over the whole group-hierarchy. This has affects on performance in case you rely on laziness of group access. You can set `phony_dims='access'` instead to defer phony dimension creation to group access time. The created phony dimension naming will differ from netCDF behaviour.

```
f = h5netcdf.File("mydata.h5", mode="r", phony_dims="access")
...
```

Footnotes

[2] Keyword default setting `phony_dims=None` for backwards compatibility.

Track Order

As of h5netcdf 1.1.0, if h5py 3.7.0 or greater is detected, the `track_order` parameter is set to `True` enabling order tracking for newly created netCDF4 files. This helps ensure that files created with the h5netcdf library can be modified by the netCDF4-c and netCDF4-python implementation used in other software stacks. Since this change should be transparent to most users, it was made without deprecation.

Since track_order is set at creation time, any dataset that was created with `track_order=False` (h5netcdf version 1.0.2 and older except for 0.13.0) will continue to opened with order tracker disabled.

The following describes the behavior of h5netcdf with respect to order tracking for a few key versions:

- Version 0.12.0 and earlier, the `track_order` parameter`order was missing and thus order tracking was implicitly set to `False`.
- Version 0.13.0 enabled order tracking by setting the parameter `track_order` to `True` by default without deprecation.
- Versions 0.13.1 to 1.0.2 set `track_order` to `False` due to a bug in a core dependency of h5netcdf, h5py upstream bug which was resolved in h5py 3.7.0 with the help of the

h5netcdf team.

- In version 1.1.0, if h5py 3.7.0 or above is detected, the `track_order` parameter is set to `True` by default.

History

The project was started in early 2015. The first commit was made on 7th of April in 2015 by Stephan Hoyer. The first *official* `h5netcdf` announcement was made by Stephan on the xarray issue tracker only one day later.

The library evolved constantly over the years (fixing bugs and adding enhancements) and gained contributions from 21 other Contributors so far. The library is widely used, especially as backend within xarray.

Early 2020 Kai Mühlbauer started to add contributions and after some time he volunteered to help in maintaining `h5netcdf`. Two years later in January 2022 Stephan handed the project-lead over to Kai. `h5netcdf` version 1.0 was released on 31st of March 2022.

License

3-clause BSD

Indices and tables

- Index
- Module Index
- Search Page