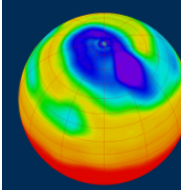




**National Centre for
Atmospheric Science**
NATURAL ENVIRONMENT RESEARCH COUNCIL



**Centre for Environmental
Data Analysis**
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

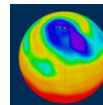
Reading NetCDF files with Python

Thanks to all contributors:

Stephen Pascoe, Jeff Whittaker



**National Centre for
Atmospheric Science**
NATURAL ENVIRONMENT RESEARCH COUNCIL



**Centre for Environmental
Data Analysis**
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

So many options!

There are many options for working with NetCDF files in Python. In this example we have chosen to highlight the use of the **netCDF4-python** module.

The **netCDF4-python** module is useful because:

- It implements the basic “classic” model as well as more advanced features.
- It provides a simple interface to the NetCDF structure.
- It has been used as the underlying NetCDF I/O layer for many more advanced packages.

Opening a netCDF file

To open a netCDF file from python, you simply call the `Dataset()` constructor as follows:

```
>>> from netCDF4 import Dataset
>>> dataset = Dataset('data.nc')

>>> print dataset.file_format
NETCDF4_CLASSIC
```

Working with “classic” NetCDF

The `netCDF4` module can read in any netCDF format.

This tutorial will focus exclusively on the NetCDF-
“classic” data model using: `NETCDF4_CLASSIC`

The “classic” data model is made up of dimensions, variables and attributes (as discussed earlier).

Interrogating dimensions

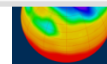
You can interrogate dimensions using simple dictionary calls:

```
>>> print dataset.dimensions.keys()  
['time', 'latitude', 'bound', 'longitude']  
  
>>> print dataset.dimensions['time']  
<type 'netCDF4.Dimension'> (unlimited):  
    name = 'time', size = 1
```

Interrogating variables

You can interrogate variables using simple dictionary calls:

```
>>> print dataset.variables.keys()  
['tcc', 'time', 'latitude', 'longitude']  
  
>>> tcc_var = dataset.variables['tcc']  
>>> print tcc_var  
<type 'netCDF4.Variable'>  
float32 tcc(time, latitude, longitude  
    missing_value: 9.999e+20  
    name: tcc  
    title: Total cloud cover ((0-1))  
unlimited dimensions: time  
current shape = (1, 181, 360)  
filling off
```



Global attributes

Global attributes are available as attributes of the python dataset instance:

```
# Get conventions attribute  
>>> print dataset.Conventions  
CF-1.5
```

```
# Or find all NetCDF global attributes  
>>> for attr in dataset.ncattrs():  
...     print attr, '=', getattr(dataset, attr)  
...  
Conventions = CF-1.0  
history = Written in a hurry on a Tuesday!
```

Variable attributes

Variable attributes are available as attributes of the python variable instance:

```
# Get units attribute  
>>> print myvar.units  
m/s
```

```
# Or find all variable attributes  
>>> for attr in myvar.ncattrs():  
...     print attr, '=', getattr(windspeed, attr)  
...  
long_name = Wind speed  
units = m/s
```


Accessing the data

Variables contain data, which you can access:

```
# Get a whole variable array
>>> arr = dataset.variables['tcc'][:]
>>> arr.shape
(1, 181, 360)
>>> type(arr)
<type 'numpy.ndarray'>
```

This is awesome!
It returns a
NumPy array

```
# Get a sub-slice of the array
>>> subset = dataset.variables['tcc'][0,10:15,5]
>>> subset.shape
(5,)
```

For NetCDF variable has missing data - a Masked Array!

Further reading

Python-netCDF4:

<http://netcdf4-python.googlecode.com/svn/trunk/docs/netCDF4-module.html>