Introduction to netCDF formats, data models, and utilities

Russ Rew, UCAR Unidata
Workshop on Using Unidata Technologies with Python
17-19 October 2016

Introduction to netCDF

Covered

- Overview
- Formats
- Data models
- Utilities
- Exercises

Not covered

- Building and installing
- Library architecture
- Application programming interfaces for C, Fortran. Python, Java, ...
- CF metadata conventions
- Compression and chunking
- Diskless files
- Parallel I/O, HPC issues
- Future plans

See most recent netCDF workshop for subjects on right

Overview of netCDF

More than just a file format ...

At its simplest, it's also:

- A data model
- An application programming interface (API)
- Software implementing the API

Together, the format, data model, API, and software support the creation, access, and sharing of scientific data.

What is netCDF, really?

- Four format variants
  - **Classic** format and **64-bit offset** format for netCDF-3
  - **NetCDF-4** format and **netCDF-4 classic model** format
- Two data models
  - **Classic** model (for netCDF-3)
  - **Enhanced** model (for netCDF-4)
- Many language APIs
  - **C-based** (Python, C, Fortran, C++, R, Ruby, ...)
  - **Java-based** (Java, MATLAB, ...)
- Unidata and 3rd-party software (NCO, NCL, CDO, ...)

Do users have to know about these complications?
Not usually, thanks to ...

Version compatibility and transparency

You mostly don't need to be aware of version complications, because **new** versions of Unidata netCDF software continue to support

- All previous netCDF formats and their variants
- All previous netCDF data models
- All previous APIs[1]
- NetCDF data written using one language API is readable through other language APIs. [2]

[1] Disclaimer: exceptions for bugs, early releases, documented experiments

[2] Exception: currently some advanced netCDF-4 features are only available from C and Fortran APIs

NetCDF Compatibility Certificate

To ensure future access to existing data archives, Unidata is committed to compatibility of:

- **Data access**: new versions of netCDF software will provide read and write access to previously stored netCDF data.
- **Programming interfaces**: C and Fortran programs using documented netCDF interfaces from previous versions will work without change* with new versions of netCDF software.
- **Future versions**: Unidata will continue to support both data access compatibility and program compatibility in future netCDF releases*.

*See reverse side of this slide for disclaimers and exceptions.

unidata

Summary: netCDF formats, data models, APIs

- **File formats:** for making data & metadata portable
  - Supports sharing array-oriented scientific data and metadata
  - Provides data that is *self-describing, portable, scalable, extendible, remotely accessible, archivable*, and *structured*
- **Data models:** for faithfully representing science data
  - Classic: simplest model -- *dimensions*, *variables*, and *attributes*
  - Enhanced: more complex and powerful model – adds *groups* and *user-defined types*
- **APIs:** for developing applications and services
  - Unidata supports and maintains C and Java APIs.
  - Community developers support and maintain Python API and others.
  - Unidata provides best-effort support for Fortran and C++ APIs.

NetCDF formats

Characteristics of netCDF formats I

NetCDF data is:

- **Self-describing:** You can include metadata as well as data, name variables, locate data in time and space, store units of measure, conform to metadata standards.
- **Portable:** You can write on one platform and read it on other platforms.
- **Scalable:** You can access small subsets of large datasets efficiently.
- **Extendible:** You can add new data efficiently without copying existing data. You can add new metadata without breaking existing programs.

Characteristics of netCDF formats 2

- **Remotely accessible:** You can access subsets of data in netCDF and other formats from remote servers using OPeNDAP protocols.
- **Archivable:** You can access earlier versions of netCDF formats using current and future versions of software.
- **Structured:** You can use a variety of types and data structures to capture the meaning in your data.

NetCDF-3 *classic formats*

Strengths

- ✓ Simple to understand and explain
- ✓ Supported by many applications
- ✓ Standardized for used in many archives, data projects
- ✓ Mature conventions and best practices (e.g. CF) available

Limitations

- No support for efficient compression
- Schema changes slowed by copying data
- Some 4 GiB limits on variable sizes
- Performance issues reading data in different order than written

NetCDF-4 *formats*

Strengths

- ✓ Efficient compression (using zlib, HDF5 storage)
- ✓ Efficient access using HDF5 chunking
- ✓ Efficient schema changes supported
- ✓ Variables can be huge
- ✓ Good testing and support for high performance computing platforms

Limitations

- ▪ Zlib compression sometimes not competitive
- ▪ Chunking defaults may need careful tuning
- ▪ Complexity of format discourages multiple implementations
- ▪ Workarounds required to handle lack of HDF5 support for shared, named dimensions

NetCDF-4 classic-model transitional “format”

netCDF-3

- Compatible with existing applications
- Simplest data model and API

netCDF-4 classic model

- Can be accessed through netCDF-3 APIs, compatible with classic data model
- Uses netCDF-4/HDF5 storage for performance: compression, chunking, …
- Transparent access after library update

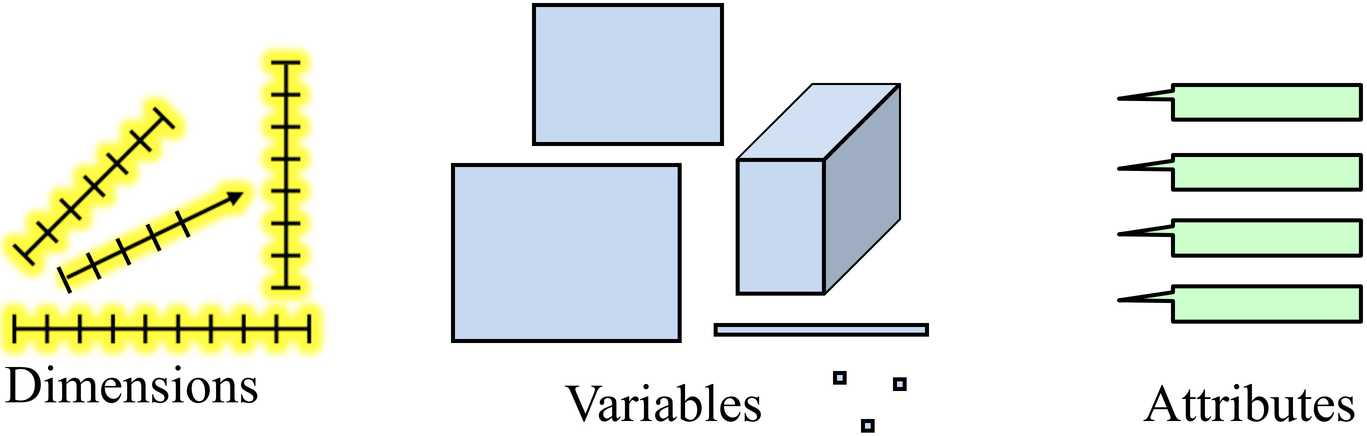
netCDF-4

- Requires netCDF-4 APIs to access enhanced model features, such as strings, groups, and user-defined types
- Good for new data and applications

NetCDF data models

The netCDF classic data model

- NetCDF data has named ***dimensions***, ***variables***, and ***attributes***.
- ***Dimensions*** are for specifying shapes of variables
- ***Variables*** are for data, attributes are for metadata
- ***Attributes*** may apply to a whole dataset or to a variable
- Variables may share dimensions, indicating a common grid.
- One dimension may be of unlimited length.
- Each variable or attribute has a ***type***: char, byte, short, int, float, double



The netCDF classic data model (UML)

NetCDF Data has
- **Dimensions** *(lat, lon, level, time, ...)*
- **Variables** *(temperature, pressure, ...)*
- **Attributes** *(units, valid_range, ...)*

Each dimension has
- Name, length

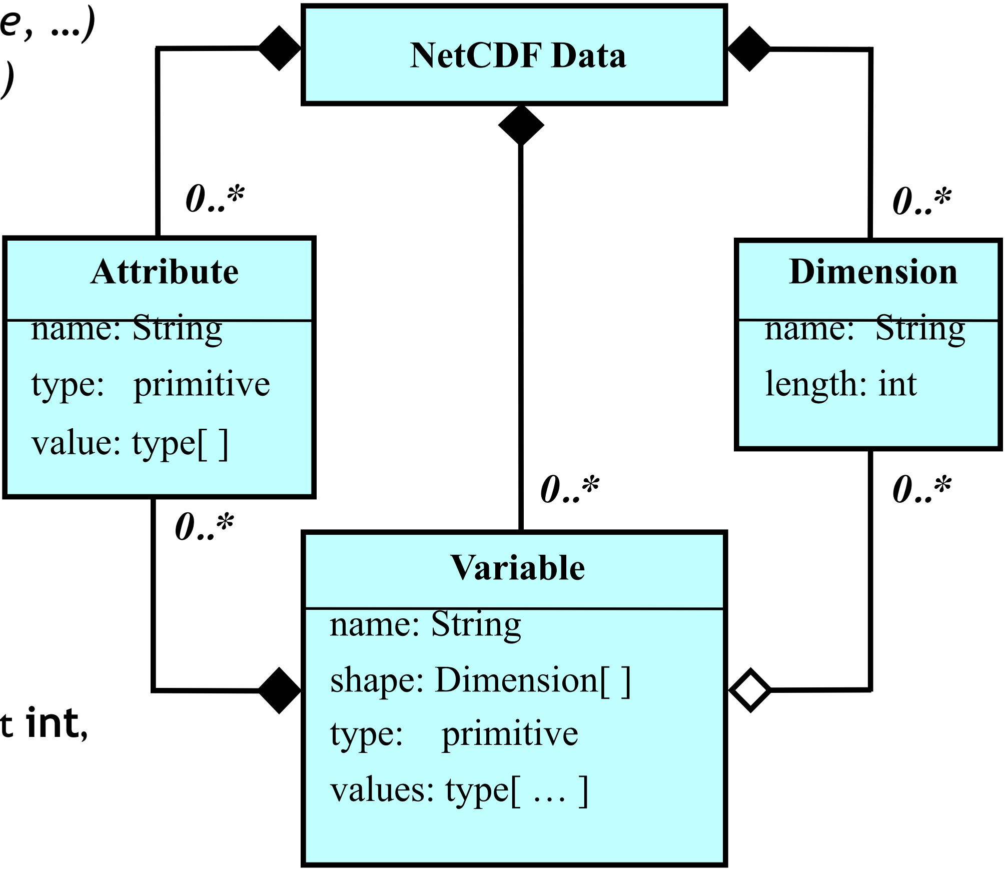
Each variable has
- Name, shape, type, attributes
- N-dimensional array of values

Each attribute has
- Name, type, value(s)

Variables *may share* dimensions
- Represents shared coordinates, grids

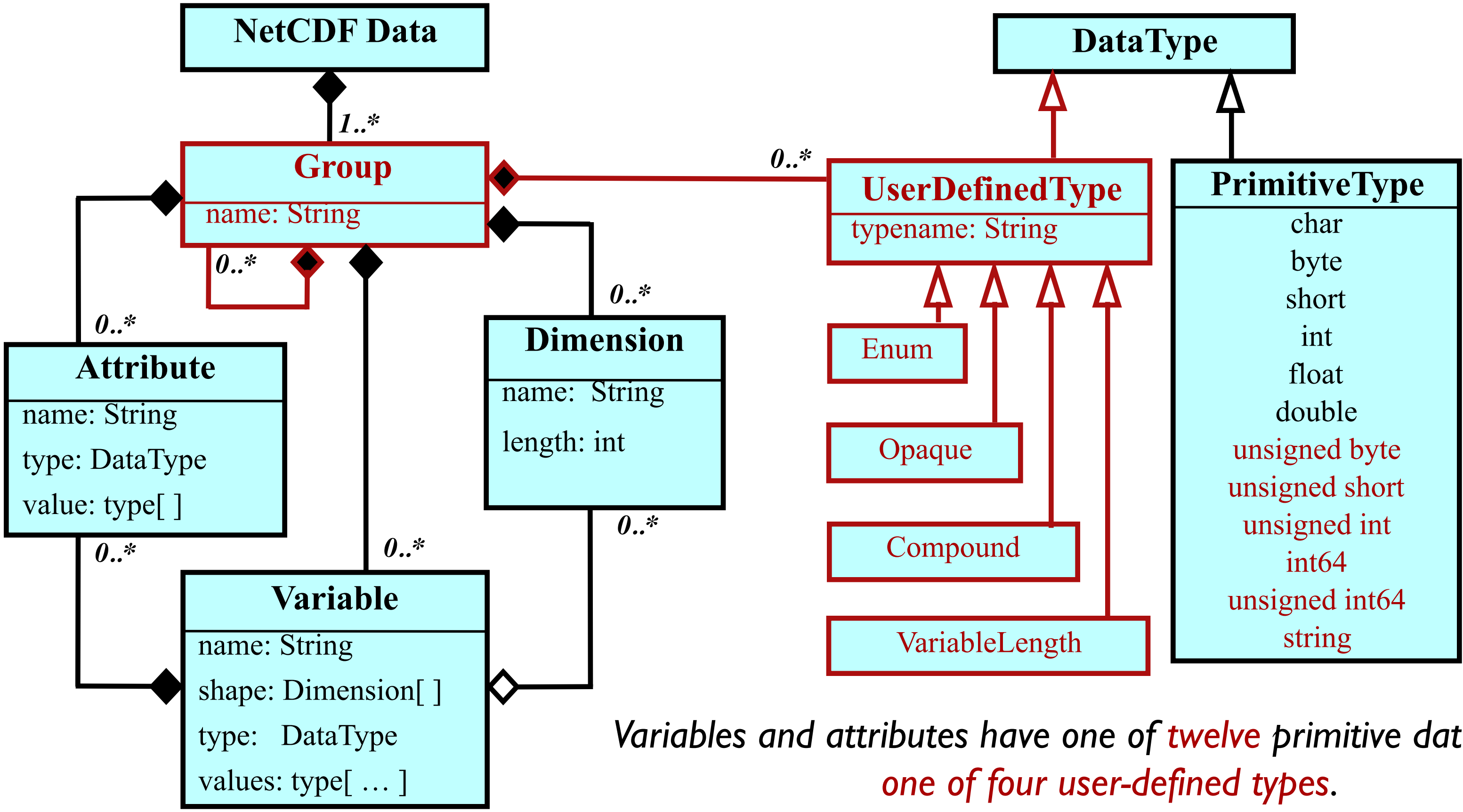
Variable and attribute values are of type
- Numeric: 8-bit **byte**, 16-bit **short**, 32-bit **int**, 32-bit **float**, 64-bit **double**
- Character: arrays of **char** for text

UML = Unified Modeling Language



The netCDF-*4 enhanced* data model

A file has a top-level unnamed group. Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One or more dimensions may be of unlimited length.



Variables and attributes have one of twelve primitive data types or one of four user-defined types.

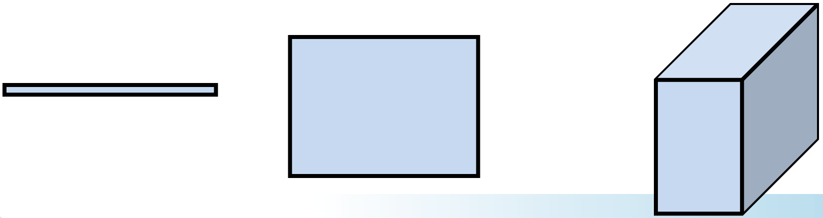
Variables or attributes?

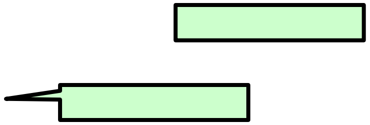
Variables

- intended for data
- can hold arrays too large for memory
- may be multidimensional
- support partial access (only a subset of values)
- values may be changed, more data may be appended
- may have attributes
- shape specified with netCDF dimensions
- not read until accessed

Attributes

- intended for metadata
- for small units of information that fit in memory
- for single values, strings, or small 1-D arrays
- atomic access, must be written or read all at once
- values typically don't change after creation
- an attribute may not have attributes
- read when file opened





NetCDF classic data model

Strengths

- ✓ Data model simple to understand and explain
- ✓ Can be efficiently implemented
- ✓ Representation good for gridded multidimensional data
- ✓ Shared dimensions useful for coordinate systems
- ✓ Generic applications easy to develop

Limitations

- Small set of primitive types
- Data model limited to multidimensional arrays, (name, value) pairs
- Flat name space that hinders organizing many data objects
- Lack of nested structures, variable-length types, enumerations

NetCDF-4 enhanced data model

Strengths

- ✓ Increased representational power for more complex data
- ✓ Adds shared, named dimensions to HDF5 data model
- ✓ Compatible with netCDF-3 classic data model
- ✓ Adds useful primitive types
- ✓ Provides nesting: hierarchical groups, recursive data types

Limitations

- ◆ More complex than classic data model
- ◆ Effort required to develop general tools and applications
- ◆ Adoption proceeding slowly
- ◆ Best practices and conventions still maturing

NetCDF utilities
(or netCDF without programming)

Common Data Language (CDL)

Text notation for netCDF metadata and data

```
  netcdf example {   // example of CDL notation
  dimensions:
         x = 2 ;
         y = 8 ;
  variables:
         float rh(x, y) ;
                rh:units = "percent" ;
                rh:long_name = "relative humidity" ;
  // global attributes
         :title = "simple example, lacks conventions" ;
  data:
   rh =
    2, 3, 5, 7, 11, 13, 17, 19,
    23, 29, 31, 37, 41, 43, 47, 53 ;
}
```

Example with 2 dimensions (x and y), 1 variable (rh), 2 variable attributes (units and long_name), 1 global attribute (title), and 16 data values.

Utility programs for netCDF to/from CDL

```
$ ncdump -h co2.nc  # converts netCDF to CDL
```

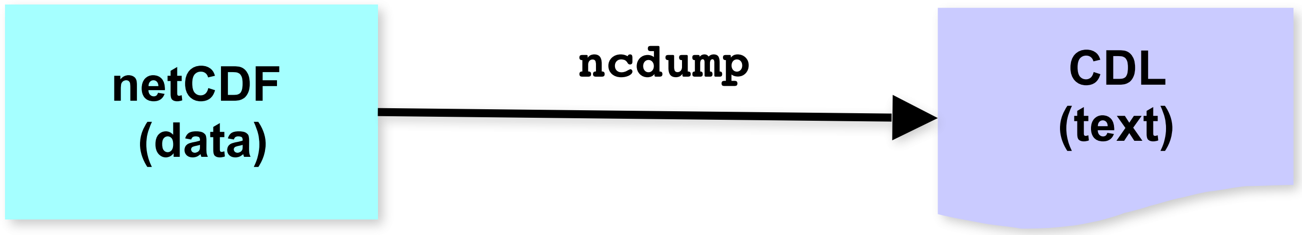
```
netcdf co2 {
dimensions:
       T = 456 ;
variables:
       float T(T) ;
              T:units = "months since 1960-01-01" ;
       float co2(T) ;
              co2:long_name = "CO2 concentration by volume" ;
              co2:units = "1.0e-6" ;
              co2:_FillValue = -99.99f ;

// global attributes:
              :references = "Keeling_etal1996, Keeling_etal1995" ;
}
```

- "-h" is for "header only", just outputs metadata, no data
- "-c" outputs header and coordinate variable data
- **ncgen** does the opposite of **ncdump**, converts CDL to netCDF

The ncdump utility

ncdump converts netCDF data to human-readable text form.

Useful for browsing data, has lots of options.

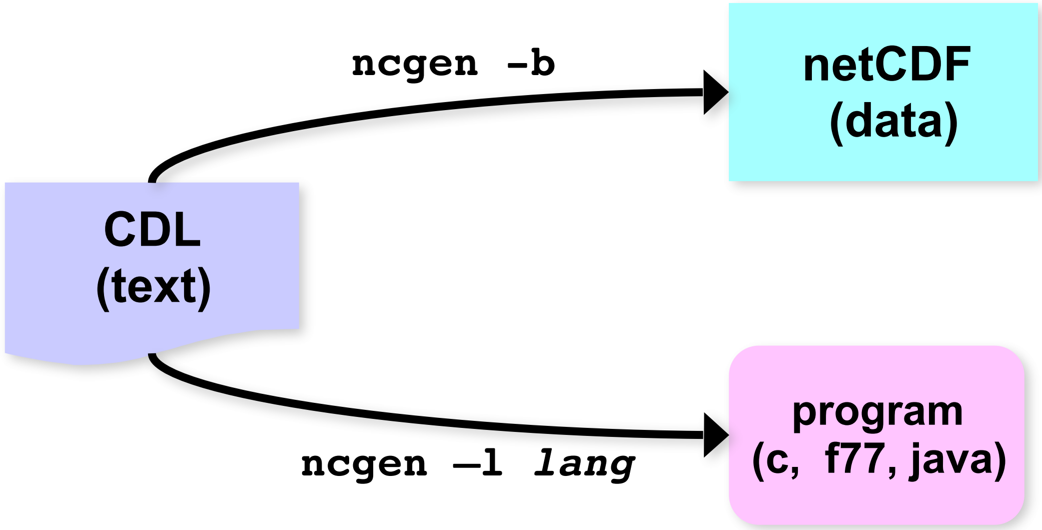


ncdump [-c|-h] [-v ...] [-k] *file.nc*

[-h | -c]	header only, or coordinates and header
[-v *var1* [,...]]	data for variable(s) *var1*,... only
[-k]	output kind of netCDF file instead of CDL
[-t]	show time data as date-time
file.nc	name of netCDF input file or OPeNDAP URL

The ncgen utility

ncgen generates a netCDF file, or a program to generate the netCDF file.

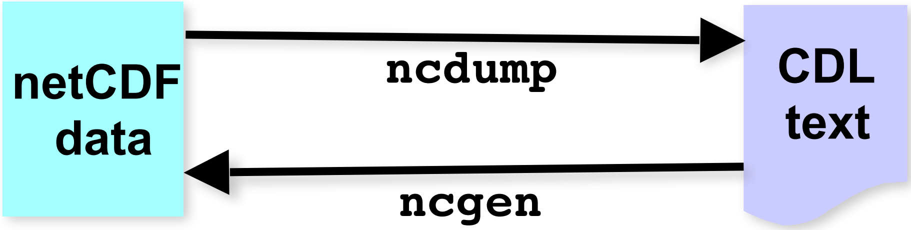


ncgen [-b] [-o *file.nc*] [-k *kind*] [-l *lang*] *file.cdl*

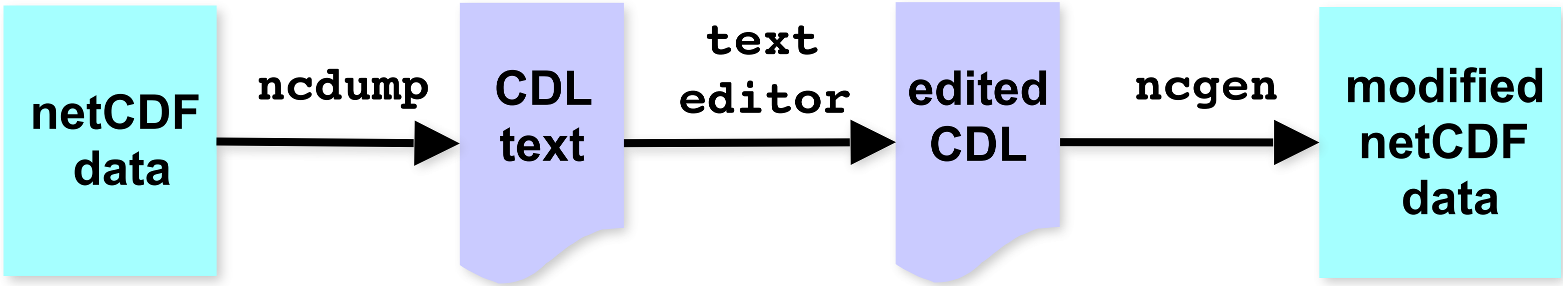
[-b]	binary output as a netCDF file, with ".nc" extension
[-o *file.nc*]	like -b except output netCDF to specified file
[-k *kind*]	kind of output netCDF file, simplest that works if omitted
[-l c | f77 | java]	language of program generated to standard output
file.cdl	name of input CDL file

Using ncgen and ncdump together

Together, **ncdump** and **ncgen** can accomplish simple manipulations with no programming. **ncdump** and **ncgen** are inverses:



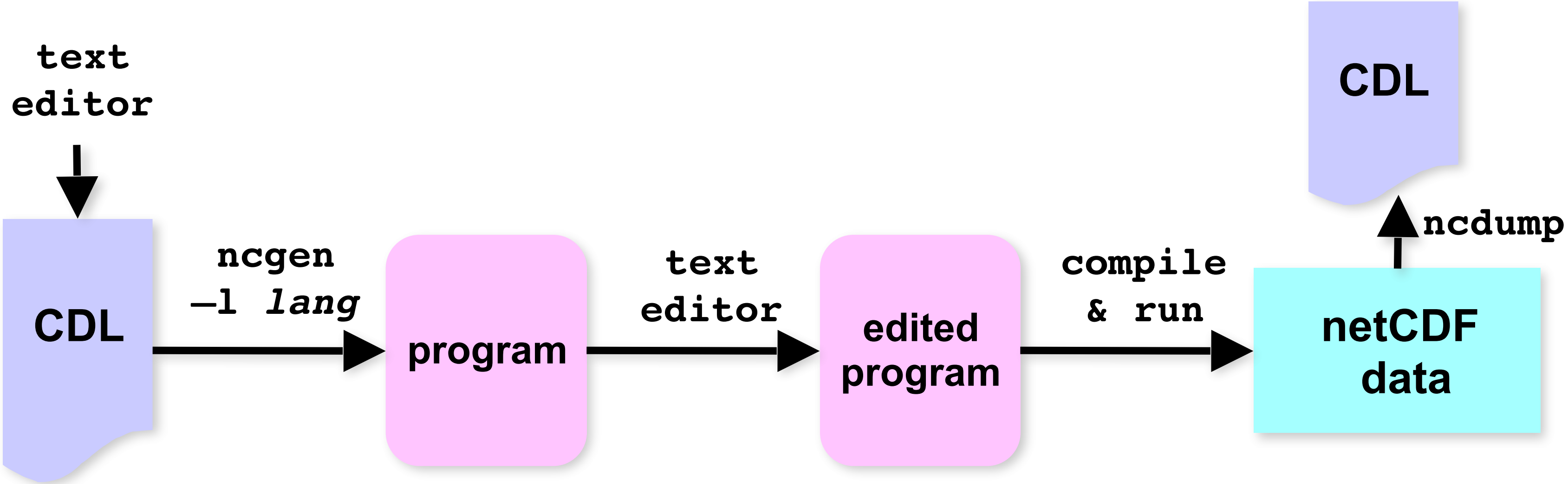
To edit metadata or data in a netCDF file:



Note: not practical for huge netCDF files or lots of files. For that, you need to write a program, using netCDF library.

Using ncgen and ncdump together

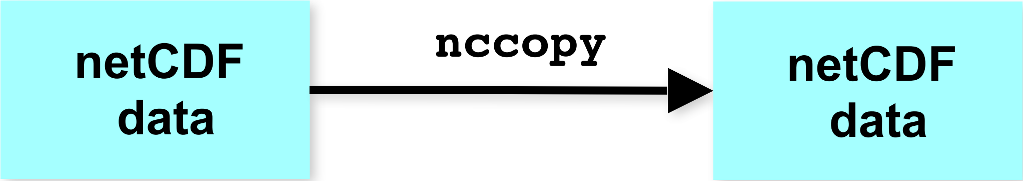
To create a new netCDF file with lots of metadata:



1. Use text editor to write CDL file with lots of metadata but no data.
2. Use **ncgen** to generate C or Fortran program for writing netCDF.
3. With text editor, insert netCDF "var_put" calls for writing data.
4. Compile and run the program to create desired netCDF file.
5. Optionally, use **ncdump** to verify result.

The nccopy utility

nccopy copies and optionally compresses and chunks netCDF data.



netCDF data

nccopy

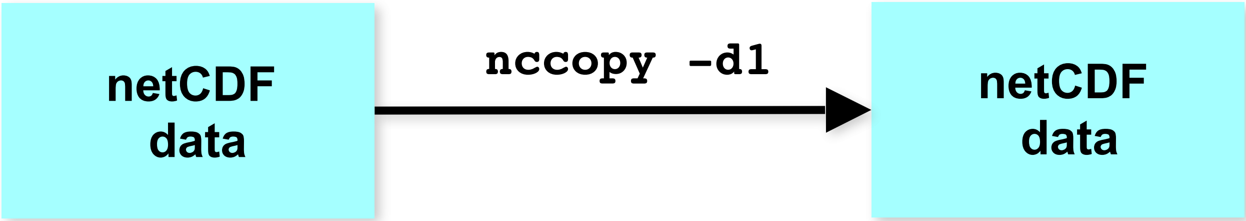
netCDF data

nccopy [-*kind_num*] [-u] [-d *level*] [-s] [-c *chunkspec*] *input* *output*

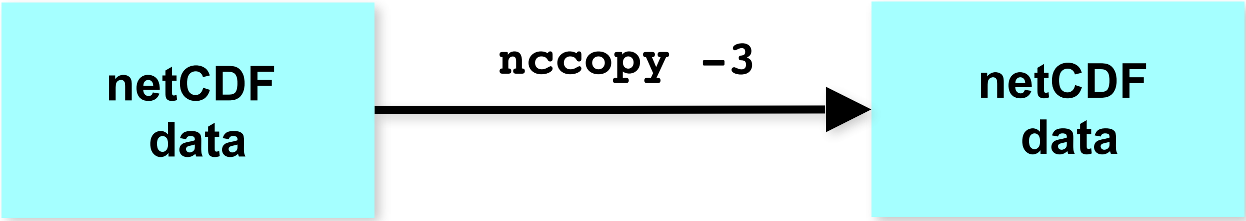
[-*kind_num*]	-3, -4, -6, -7 kind of netCDF output, default same as input
[-u]	convert unlimited dimensions to fixed-size in output
[-d *level*]	zlib "deflation" level, default same as input
[-s]	shuffling option, sometimes improves compression
[-c *chunkspec*]	specify chunking for dimensions
input	name of input file or OPeNDAP URL
output	name of output file

Using nccopy

Compress netCDF data to a specified level, compressing each variable separately.



Convert a netCDF-4 classic model file to a netCDF-3 classic file, uncompressing any compressed variables.



NetCDF exercises

Sample files for exercises

- Small netCDF file: data/mslp.nc
- Test file for compression: data/testrh.nc
- You can run netCDF utilities in iPython using “!” prefix, as in

```
>>> !ncdump file.nc
```

- Here’s an example of capturing the output of a command (as a list of strings) into a python variable using assignment, and writing the list of output strings, separated by newlines, to a file:

```
>>> f = open(‘file.cdl’, ‘w’)
>>> output_string = !ncdump file.nc
>>> f.write(output_string.n)
>>> f.close()
```

Try ncdump utility

- Look at just the header information (also called the schema or metadata):

  ```
  $ ncdump -h mslp.nc
  ```

- Store entire CDL output for use later in ncgen exercises

  ```
  $ ncdump mslp.nc > mslp.cdl
  ```

- Look at header and coordinate information, but not the data:

  ```
  $ ncdump -c mslp.nc
  ```

- Look at all the data in the file, in addition to the metadata:

  ```
  $ ncdump mslp.nc
  ```

- Look at a subset of the data by specifying one or more variables:

  ```
  $ ncdump -v lat,time mslp.nc
  ```

- Look at times in human-readable form:

  ```
  $ ncdump -t -v lat,time mslp.nc
  ```

- Look at what kind of netCDF data is in the file (classic, 64-bit offset, netCDF-4, or netCDF-4 classic model):

  ```
  $ ncdump -k mslp.nc
  ```

Try ncgen utility

- Check a CDL file for any syntax errors:

  ```
  $ ncgen mslp.cdl
  ```

- Edit mslp.cdl and change something (name of variable, data value, etc.).
- Use ncgen to generate new binary netCDF file (my.nc) with your changes:

  ```
  $ ncgen –o my.nc mslp.cdl
  $ ncdump my.nc
  ```

- Generate a C, Fortran, or Java program which, when compiled and run, will create the binary netCDF file corresponding to the CDL text file.

  ```
  $ ncgen –l c mslp.cdl > mslp.c
  $ ncgen –l f77 mslp.cdl > mslp.f77
  $ ncgen –l java mslp.cdl > mslp.java
  ```

- Try compiling and running one of those programs. You will need to know where the netCDF library is to link your program.

Try remote access

- Look at what's in some remote data from an OPeNDAP server:

```
$ SERVER=http://test.opendap.org
$ REMOTE=$SERVER/opendap/data/nc/3fnoc.nc
$ ncdump -c $REMOTE
```

- Copy 3 coordinate variables out of the file

```
$ nccopy $REMOTE’?’lat,lon,time coords.nc
```

- Copy subarray of variable u out of the file into a new netCDF file

```
$ nccopy $REMOTE’?’u[2:5][0:4][0:5] u.nc
$ ncdump u.nc
```

Try compression with nccopy utility

- Compress variables in a test file, testrh.nc, by using nccopy. Then check if adding the shuffling option improved compression:

```
$ nccopy -d1 testeh.nc testrhd1.nc        # compress data, level 1
$ nccopy -d1 -s testrh.nc testrhd1s.nc    # shuffle and compress data
$ ls -l testrh.nc testrhd1.nc testrhd1s.nc  # check results
```

Join the netCDF community

- Why participate?
  - To help extend netCDF to meet an important need.
  - To fix a bug that affects you or your users.
  - To help the geosciences community.
- How to collaborate?
  - Join netcdfgroup@unidata.ucar.edu mailing list
  - Use Unidata netCDF GitHub repository.
  - Build and test release candidates, provide feedback.
  - Contribute code, tests, and documentation improvements.
  - Suggest new features.
  - Also see netCDF Jira site for current open issues.