# Introduction to NetCDF-4

Ed Hartnett, Unidata/UCAR

NetCDF Workshop July 25 – 26, 2011

## What is NetCDF-4?

- A project to allow the netCDF library to use HDF5 as a data store.
- Joint project between netCDF and HDF teams.
- Allows netCDF users to take advantage of HDF5 features without have to know (too much) about HDF5.

## New NetCDF Binary Format

- Before the netCDF-4 project, there were two binary formats: classic and 64-bit offset.
- NetCDF-4.0 introduced a new binary format: netCDF-4/HDF5.
- It is an HDF5 file, with some additional meta-data.
- It is read by netCDF code just like any other netCDF file.

## Timeline

- Initial netCDF code development 2003 – 2005 at Unidata.
- Co-released netCDF-4.0 with HDF5 1.8.0 in 2008.
- Is maintained and developed at the Unidata Program Center.
- Most recent release: 4.1.3, June, 2011.

## Some Nice HDF5 Features

- Performance: chunking, endianness control, parallel I/O, deflation.
- New data types: unsigned ints, user-defined types.
- Hierarchical organization within file: groups.
- Less restrictions on sizes, number of unlimited dimensions.

## Interoperability Advantage

- With netCDF-4/HDF5 files, the data can be easily understood by both netCDF and HDF5 applications.
- With (sometimes) extra effort, HDF5 applications can write netCDF readable files.
- No more converting between HDF5 and netCDF!
- How many people in the room have written such a program?
- We have slain a horrible monster.

## Backward Compatibility

- Backward compatibility was the first requirement of the netCDF-4 project.
- Classic binary format is still the default output format. NetCDF-4/HDF5 files must be specified at create-time.
- But how to handle code compatibility?
- If netCDF-4 allows multiple unlimited dimensions (for example), how to ensure code compatibility in all cases?

## Backward Code Compatibility Mode

- In order to provide the exact same behavior in existing code, netCDF-4/HDF5 files can be created with a special flag: NC_CLASSIC_MODEL
- This is a backward compatibility flag. It restricts what is allowed in the file.
- If the user attempts to (for example) create two unlimited dimensions in this file, an error will result, just as with classic format.
- NetCDF-4/HDF5 files produced without this flag can have any number of unlimited dimensions.

## Don't **Need** CLASSIC_MODEL for Code Compatibility

- The CLASSIC_MODEL is not needed for backward code compatibility if you don't care about erroring out calls to the enhanced model features.
- CLASSIC_MODEL just makes netCDF-4 more strict.
- The data format is still HDF5, and no different from the same file created without CLASSIC_MODEL, except the CLASSIC_MODEL file will always reject any enhanced model feature.

## CLASSIC_MODEL and NetCDF Testing

- The CLASSIC_MODEL flag is needed for existing netCDF tests to run on netCDF-4/HDF5 files.
- These tests (see directories nctest, nc_test) check that the correct errors are returned for conditions like attempting to define two unlimited dimensions.
- With the CLASSIC_MODEL flag, netCDF-4/HDF5 files are restricted to the classic model, so that such tests will work.

## Creating Different Formats: Classic

- Classic file (the default):

```
nc_create(“file.nc”, 0, &ncid);
nc_def_dim(...);
nc_def_var(...);
nc_put_var(...)
nc_close(...)
```

## Creating Different Formats: 64-bit Offset

- 64-bit offset file:

```
nc_create(“file.nc”, NC_64BIT_OFFSET, &ncid);
nc_def_dim(...);
nc_def_var(...);
nc_put_var(...)
nc_close(...)
```

# Creating Different Formats: NetCDF-4/HDF5

- NetCDF-4/HDF5 file:

```
nc_create(“file.nc”, NC_NETCDF4, &ncid);
nc_def_dim(...);
nc_def_var(...);
nc_put_var(...)
nc_close(...)
```

# Creating Different Formats: NetCDF-4/HDF5 with CLASSIC_MODEL

- NetCDF-4/HDF5 file:

```
nc_create(“file.nc”, NC_NETCDF4|NC_CLASSIC_MODEL, &ncid);
nc_def_dim(...);
nc_def_var(...);
nc_put_var(...)
nc_close(...)
```

## The Difference Between Last Two Examples

- Both will work the same.
- If opened later, the file **without** CLASSIC_MODEL can have additional unlimited dimensions, groups, user-defined types, and other enhanced model features added to the file.
- The file created with CLASSIC_MODEL can never enjoy these nifty new features.

## Build NetCDF without NetCDF-4 Features

- NetCDF-4 requires that the HDF5 and zlib libraries be installed.
- It also adds a lot of code to the library.
- The configure option –disable-netcdf-4 will turn off all netCDF-4 features, and HDF5 will not be required (zlib will be required unless the remote data client is also turned off).
- Such a build will not read or write netCDF-4/HDF5 files.
- Why? Embedded platforms, limited platforms, or quick build.

## How to Upgrade Code to Write NetCDF-4/HDF5

- Install latest netCDF release.
- Change nc_create calls to include the NC_NETCDF4 flag.
- Optionally control performance features like chunking.
- Optionally add deflation.
- Recompile, relink, and run.

## Summary: How to Upgrade Existing Applications to Read NetCDF-4/HDF5 Files

- Install latest version of netCDF.
- Link to it.
- Only classic model is handled without extra code.
- Deflation is handled transparently. Data are uncompressed in chunks as they are read.

## Define vs. Data Mode

- For classic and 64-bit offset files, define and data mode are explicitly managed by the user.
- For netCDF-4/HDF5 files, data and define mode can be left to netCDF-4. The mode will be switched as needed.
- So calls to nc_enddef/nc_redef can be removed.
- This automatic mode switching is turned off for files created with CLASSIC_MODEL flag.

## Conclusion

- NetCDF-4/HDF5 is an addition to the toolbox, not a replacement for existing formats.
- Full backward compatibility will be maintained in the C and Fortran APIs.
- Always upgrade to the latest version of netCDF!
- Changing output data to netCDF-4/HDF5 is optional, but easy.