Developing Conventions for netCDF-4

Russ Rew, UCAR Unidata

June 11, 2007

GO-ESSP

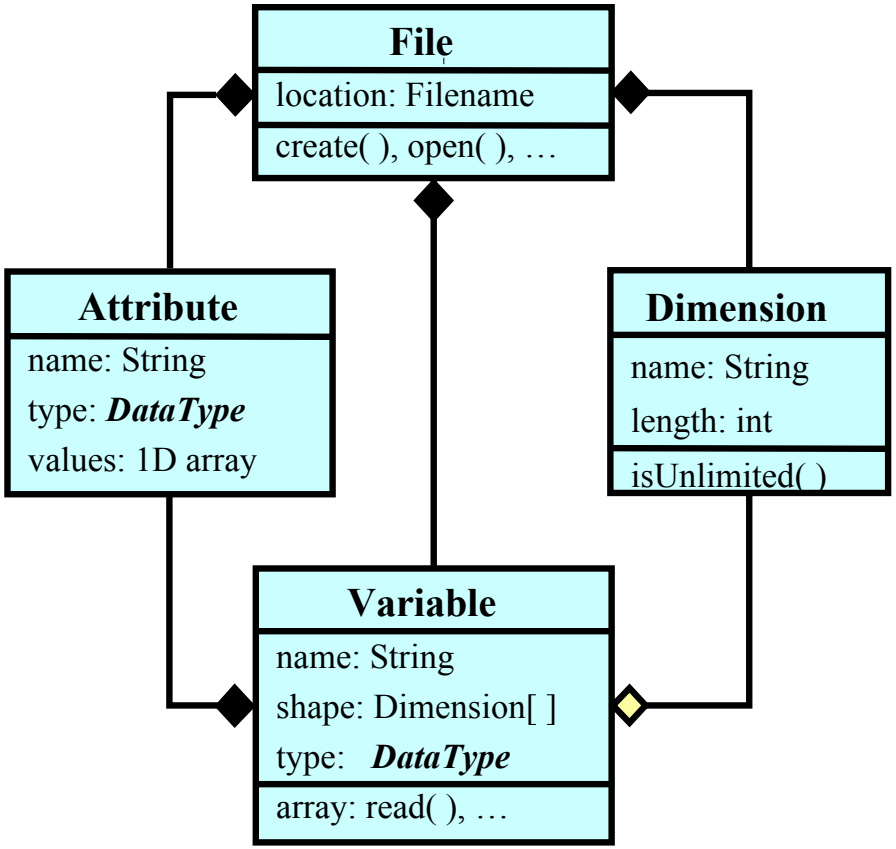
Overview

- Two levels of conventions: NUG and CF
- “Classic” and extended netCDF-4 data models
- Data models and data formats
- Potential uses and examples of netCDF-4 data model features
- CF conventions issues
- Benefits of using netCDF-4 format but classic data model
- Recommendations and conclusions

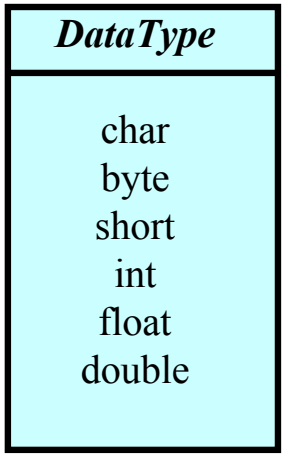
Background: netCDF and Conventions

- Purpose of conventions
  - To capture meaning in data, intent of data provider
  - To foster interoperability
- NetCDF User Guide conventions
  - Concepts: simple coordinate variables, …
  - Attribute based: `units`, `Conventions`, …
- Climate and Forecast (CF) conventions
  - Concepts: generalized coordinates, …
  - Models relationships among variables
  - Standard names
  - Attribute based

Classic NetCDF Data Model



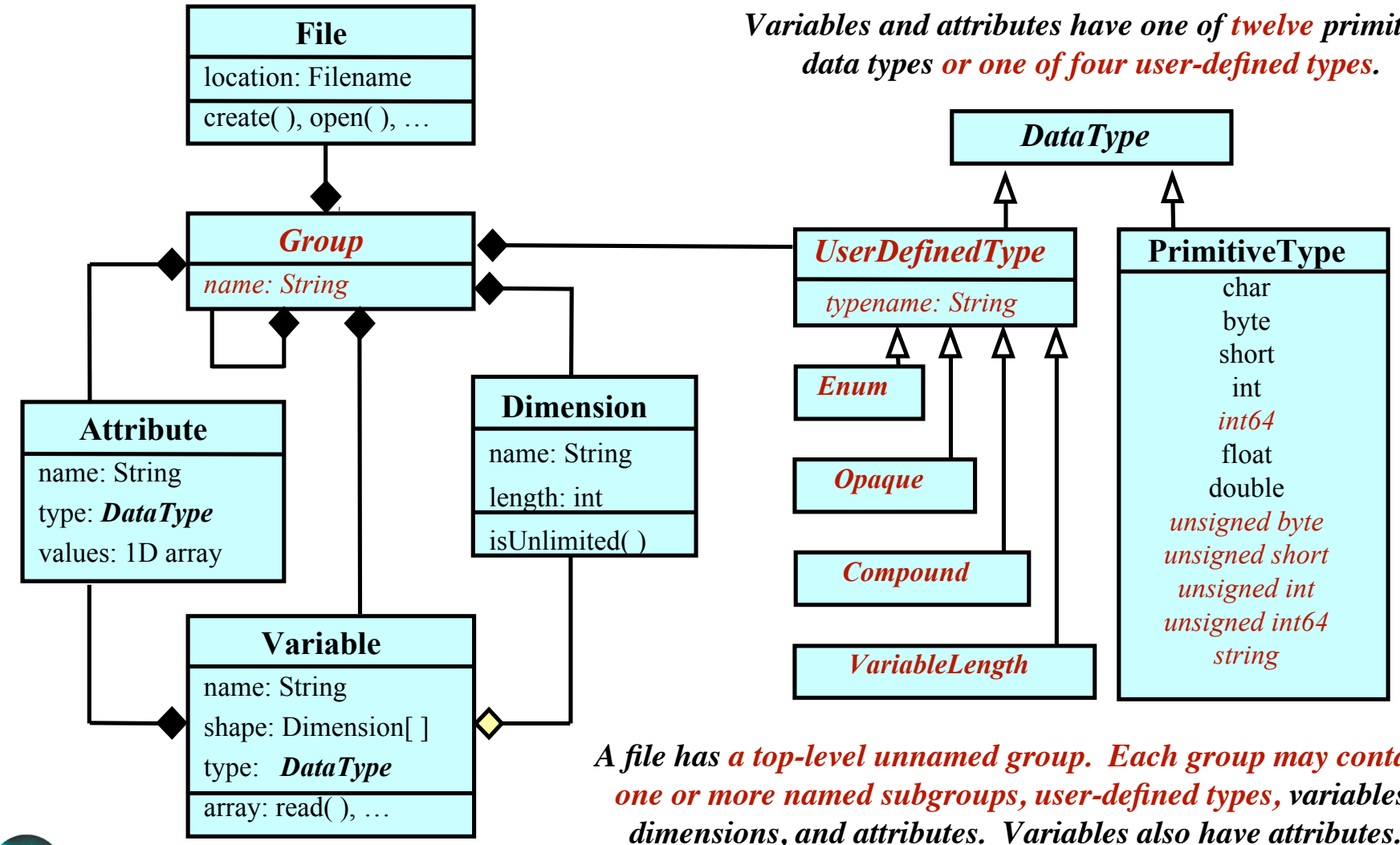
Variables and attributes have one of six primitive data types.



A file has named variables, dimensions, and attributes. A variable may also have attributes. Variables may share dimensions, indicating a common grid. One dimension may be of unlimited length.

unidata

NetCDF-4 Data Model



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

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.

unidata

Some Limitations of Classic NetCDF Data Model

- Little support for data structures, just multidimensional arrays and lists
- No “ragged arrays” or nested structures
- Only one shared unlimited dimension for appending new data efficiently
- Flat name space for dimensions and variables
- Character arrays rather than strings
- Small set of numeric types
- Variable size constraints, packing instead of compression, inefficient schema additions, …

NetCDF-4 Features for Data Providers

HDF5-based format provides:

- Per-variable compression
- Per-variable multidimensional tiling (chunking)
- Liberal variable size constraints
- Reader-makes-right conversion
- Efficient dynamic schema additions
- Parallel I/O

Data model provides:

- Groups for nested scopes
- User-defined enumeration types
- User-defined compound types
- User-defined variable-length types
- Multiple unlimited dimensions
- String type
- Additional numeric types

NetCDF Data Models and File Formats

Data providers writing new netCDF data have two obvious alternatives:

1. Use simple classic data model and format
2. Use richer netCDF-4 data model and netCDF-4 format

and a third less obvious choice:

3. Use classic data model with the netCDF-4 format

“Classic model” netCDF-4 files

- Supported by netCDF-4 library with file creation flag
- Ensures data can be read by netCDF-3 software (relinked to netCDF-4 library)
- Compatible with current conventions
- Writers get benefits of new format, but not data model
- Readers can
  - access compressed or chunked variables transparently
  - get performance benefits of reader-makes-right
  - use of HDF5 tools

Is it Time to Adopt NetCDF-4 Data Model?

- C-based netCDF-4 software still only in beta release
- Few netCDF utilities or applications adapted to full netCDF-4 model yet
- Little experience with netCDF-4 means useful conventions still in early stages
- Significant performance improvements available *without* netCDF-4 data model

NetCDF-4 Data Model Features: Examples and Potential Uses

- Groups
- Compound types
- Enumerations
- Variable-length types

Example Use of Groups

Data for named geographical regions:

```
group Europe {
  group France {
    dimensions: time = unlimited, stations = 47;
    variables: float temperature(time, stations);
  }
  group England{
    dimensions: time = unlimited, stations = 61;
    variables: float temperature(time, stations);
  }
  group Germany {
    dimensions: time = unlimited, stations = 53;
    variables: float temperature(time, stations);
  }
  …
  dimensions: time = unlimited;
  variables: float average_temperature(time);
}
```

Potential Uses for Groups

- Factoring out common information
  - Containers for data within regions
  - Model metadata
- Organizing a large number of variables
- Providing name spaces for multiple uses of same names for dims, vars, atts
- Modeling large hierarchies
- CF conventions issues
  - Ensembles
  - Shared structured grids
  - Other uses?

Example Use of Compound Type

Vector quantity, such as wind:

```
types:
  compound wind_vector_t {
    float eastward ;
    float northward ;
    }
dimensions:
    lat = 18 ;
    lon = 36 ;
    pres = 15 ;
    time = 4 ;
variables:
    wind_vector_t gwind(time, pres, lat, lon) ;
       wind:long_name = "geostrophic wind vector" ;
       wind:standard_name = "geostrophic_wind_vector" ;
data:
    gwind = {1, -2.5}, {-1, 2}, {20, 10}, {1.5, 1.5}, ...;
```

Potential Uses for Compound Types

- Representing vector quantities like wind
- Modeling relational database tuples
- Representing objects with components
- Bundling multiple *in situ* observations together (profiles, soundings)
- Providing containers for related values of other user-defined types (strings, enums, …)
- Representing C structures portably
- CF Conventions issues:
  - should type definitions or names be in conventions?
  - should member names be part of convention?
  - should quantities associated with groups of compound standard names be represented by compound types?

Drawbacks with Compound Types

- Member fields have type and name, but are *not* netCDF variables
- Can't directly assign attributes to compound type members
  - New proposed convention solves this problem, but requires new user-defined type for each attribute
- Compound type not as useful for Fortran developers, member values must be accessed individually

Example Convention for Member Attributes

```
types:
  compound wind_vector_t {
    float eastward ;
    float northward ;
    }
  compound wv_units_t {
    string eastward ;
    string northward ;
    }
dimensions:
    station = 5;
variables:
    wind_vector_t wind(station) ;
       wv_units_t wind:units = {"m/s", "m/s"} ;
       wind_vector_t wind:_FillValue = {-9999, -9999} ;
data:
    wind = {1, -2.5}, {-1, 2}, {20, 10}, ... ;
```

Example Use of Enumerations

Named flag values for improving self-description:

```
types:
  byte enum cloud_t {
     Clear = 0, Cumulonimbus = 1, Stratus = 2,
     Stratocumulus = 3, Cumulus = 4, Altostratus = 5,
     Nimbostratus = 6, Altocumulus = 7, Missing = 127
     };
dimensions:
     time = unlimited;
variables:
     cloud_t primary_cloud(time);
        cloud_t primary_cloud:_FillValue = Missing;
data:
     primary_cloud = Clear, Stratus, Cumulus, Missing, …;
```

Potential Uses for Enumerations

- Alternative for using strings with `flag_values` and `flag_meanings` attributes for quantities such as `soil_type`, `cloud_type`, ...
- Improving self-description while keeping data compact
- CF Conventions issues:
  - standardize on enum type definitions and enumeration symbols?
  - include enum symbol in standard name table?
  - standardize way to store descriptive string for each enumeration symbol?

Example Use of Variable-Length Type

In situ observations:

```
types:
  compound obs_t {
    float pressure ;
    float temperature ;
    float salinity ;
  }
  obs_t observations_t(*) ;  // a variable number of observations
  compound sounding_t {
    float latitude ;
    float longitude ;
    int time;
    obs_t obs;
  }
  sounding_t soundings_t(*) ;  // a variable number of soundings
  compound track_t {
    string id ;
    string description ;
    soundings_t soundings;
  }
dimensions: tracks = 42;
variables:  track_t cruise(tracks);
```

Potential Uses for Variable-Length Type

- Ragged arrays
- In situ observational data (profiles, soundings, time series)

Notes on netCDF-4 Variable-Length Types

- Variable length value must be accessed all at once (e.g. whole row of a ragged array)
- Any base type may be used (including compound types and oter variable-length types)
- No associated shared dimension, unlike multiple unlimited dimensions
- Due to atomic access, using large base types may not be practical

Recommendations for Data Providers

- Continue using classic data model and format, if suitable

  CF Principle: Conventions should be developed only for known issues. Instead of trying to foresee the future, features are added as required
- Evaluate practicality and benefits of classic model with netCDF-4 format
- Test and explore uses of extended netCDF-4 data model features
- Help create new netCDF-4 conventions based on experience with what works

When is NetCDF-4 Data Model Needed?

- If non-classic primitive type is needed
  - 64-bit integers for statistical applications
  - unsigned bytes, shorts, or ints for wider range
  - real strings instead of char arrays
- If making data self-descriptive requires new user-defined types
  - groups
  - compound
  - variable-length
  - enumerations
  - nested combinations of types

Three-Stage Chicken and Egg Problem

- Data providers
  - Won't be first to use features not supported by applications or standardized by conventions
- Application developers
  - Won't expend effort needed to support features not used by data providers and not standardized as published conventions
- Convention creators
  - Likely to wait until data providers identify needs for new conventions
  - Must consider issues applications developers will confront to support new conventions

Importance of CF

Ray Pierrehumbert (University of Chicago) had this to say on realclimate.org:

... I think one mustn't discount a breakthrough of a technological sort in AR4 though: The number of model runs exploring more of scenario and parameter space is vastly increased, and more importantly, it is available in a coherent archive to the full research community for the first time. The amount of good science that will be done with this archive in the next several years is likely to have a significant impact on our understanding of climate.