Overview

- Formats, conventions, and models
- NetCDF-3 limitations
- NetCDF-4 features: examples and potential uses
- Compatibility issues
- Conventions issues
- Recommendations
Data Abstraction Levels: Formats, Conventions, and Models

Data Models
- netCDF classic
- netCDF/CF
- CDM (netCDF-4)
- HDF5

Data Conventions
- CF-1.0
- netCDF User Guide
- Unidata Obs
- ARGO

Data Formats
- HDF-EOS
- netCDF classic
- HDF5
- netCDF-4
- BUFR
- GRIB1
- GRIB2
- CDL

NetCDF Formats
- 1988: "classic" format
- 2002: NcML (XML-based)
- 2005: 64-bit offset variant
- 2007: netCDF-4 (HDF5-based)
Commitment to Backward Compatibility

Because preserving access to archived data for future generations is *sacrosanct*:

- **Data access**: New netCDF software will provide read and write access to all earlier forms of netCDF data.
- **APIs and programs**: Existing C, Fortran, and Java netCDF programs will be supported by new netCDF software (possibly after recompiling).
- **Commitment**: Future versions of netCDF software will continue to support data access, API, and conventions compatibility.

Purpose of Data Conventions

- To capture meaning in data
- To make files self-describing
- To faithfully represent intent of data provider
- To foster interoperability
- To add value to formats
  - Raise level of abstraction (e.g. adding coordinate systems)
  - Customize format for discipline or community (e.g. climate modeling)
NetCDF conventions

- Users Guide conventions:
  - Simple coordinate variables (same name for dimension and variable)
  - Common attributes: units, long_name, valid_range, scale_factor, add_offset, _FillValue, history, Conventions, ...
  - Not just for earth-science data
- Followed by lots of community conventions: COARDS, GDT, NCAR-RAF, ARGO, AMBER, PMEL-EPIC, NODC, ..., CF
- Unidata Obs Conventions for netCDF-3 (supported by Java interface)
- Climate and Forecast conventions (CF) endorsed by Unidata (2005)
- Unidata committed to development of libcf (2006)

CF Conventions (cfconventions.org)

- Clear, comprehensive, consistent (thanks to Eaton, Gregory, Drach, Taylor, Hankin)
- standard_name attribute for identifying quantities, comparison of variables from different sources
- Coordinate systems support
- Grid cell bounds and measures
- Acceptance by community: IPCC AR4 archive, ...
- Governance and stewardship: GO-ESSP, BADC, PCMDI, WCRP/WGCM (pending)
CF Conventions Issues

- cf-metadata mailing list
- cfconventions.org site: documents, forums, wiki, Trac system
- GO-ESSP annual meetings
- Recent CF issues and proposed CF extensions
  - Structured grids, staggered grids, subgrids, curvilinear coordinates (Balaji)
  - Unstructured grids (Gross)
  - Forecast time axis (Gregory, Caron)
  - Means and subgrid variation and anomaly modifier for standard names
  - Additions needed for observational data
  - NetCDF-4 issues
  - Needs for IPCC AR5 model output archives

Scientific Data Models

- Tabular data
  - Relational model
  - Tuples, types, queries, operations, normalization, integrity constraints
- Geographic data
  - GIS models
  - Features and coverages, observations and measurements
  - Adds spatial location to relational model
- Multidimensional array data
  - Basis of netCDF, HDF models
  - Dimensions, variables, attributes
- Scientific data types
  - Coordinate systems, groups, types: structures, varlens, enums
  - N-dimensional grids, in situ point observations, profiles, time series, trajectories, swaths, …
NetCDF Data Models

• “Classic” netCDF model (netCDF-3 and earlier)
  – Dimensions, Variables, and Attributes
  – Character arrays and a few numeric types
  – Simple, flat
• CDM (netCDF-4 and later)
  – Dimensions, Variables, Attributes, Groups, Types
  – Additional primitive types including strings
  – User-defined types support structures, variable-length values, enumerations
  – Power of recursive structures: hierarchical groups, nested types

Classic NetCDF Data Model

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.
Some Limitations of Classic NetCDF Data Model and Format

- Little support for data structures, just multidimensional arrays and lists
- No nested structures or "ragged arrays"
- 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
- Constraints on sizes of large variables
- No compression, just packing
- Schema additions may be very inefficient
- Big-endian bias may hamper performance on little-endian platforms

NetCDF-4 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.
NetCDF-4 Format and Data Model

Benefits

HDF5-based format provides:
• Per-variable compression
• Per-variable multidimensional tiling (chunking)
• Ample variable sizes
• Reader-makes-right conversion
• Efficient dynamic schema additions
• Parallel I/O

New 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

Chunking

• Allows efficient access of multidimensional data along multiple axes
• Compression applies separately to each chunk
• Can improve I/O performance for very large arrays and for compressed variables
• Default chunking parameters are based on a size of one in each unlimited dimension
NetCDF-4 Data Model Features

- Examples in “CDL-4”
  - Groups
  - Compound types
  - Enumerations
  - Variable-length types
- Not necessarily best practices
- Other potential known uses
- Advice on known limitations
- Potential conventions issues

Example Use of Groups

Organize data by named property, e.g. region:

```plaintext
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, ensembles
  – Model metadata
• Organizing a large number of variables
• Providing name spaces for multiple uses of same names for dimensions, variables, attributes
• Modeling large hierarchies

Example Use of Compound Type

Vector quantity, such as wind:

```plaintext
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}, ...;
```
Another Compound Type Example

Point observations :

types:
  compound ob_t {
    int station_id;
    double time;
    float temperature;
    float pressure;
  }
dimensions:
  nstations = unlimited;
variables:
  ob_t obs(nstations);
data:
  obs = {42, 0.0, 20.5, 950.0}, ... ;

Potential Uses for Compound Types

• Representing vector quantities like wind
• Modeling relational database tuples
• Representing objects with components
• Bundling multiple \textit{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

```plaintext
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 Types

In situ observations:

```c
// type for a single observation
typedef struct {
    float pressure;
    float temperature;
    float salinity;
} obs_t;

// type for some observations
obs_t some_obs_t[4];

// type for a single profile
typedef struct {
    float latitude;
    float longitude;
    int time;
    obs_t obs;
} profile_t;

// type for some profiles
profile_t some_profiles_t[4];

// type for a single track
typedef struct {
    string id;
    string description;
    profile_t profiles;
} track_t;

// this cruise has 42 tracks
track_t cruise(42);
```

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 other variable-length types)
• No associated shared dimension, unlike multiple unlimited dimensions
• Due to atomic access, using large base types may not be practical

Recommendations and Best Practices …
NetCDF Data Models and File Formats

Data providers writing new netCDF data have two obvious alternatives:

1. Use netCDF-3: classic data model and classic 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

Third Choice: “Classic model” netCDF-4

- Pseudo format 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 performance benefits of new format
- Readers can
  - access compressed or chunked variables transparently
  - get performance benefits of reader-makes-right
  - use HDF5 tools on files
<table>
<thead>
<tr>
<th>NetCDF-4 Format and Data Model Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDF5-based format provides:</td>
</tr>
<tr>
<td>• Per-variable compression</td>
</tr>
<tr>
<td>• Per-variable multidimensional tiling</td>
</tr>
<tr>
<td>(chunking)</td>
</tr>
<tr>
<td>• Ample variable sizes</td>
</tr>
<tr>
<td>• Reader-makes-right conversion</td>
</tr>
<tr>
<td>• Efficient dynamic schema additions</td>
</tr>
<tr>
<td>• Parallel I/O</td>
</tr>
<tr>
<td>New data model provides:</td>
</tr>
<tr>
<td>• Groups for nested scopes</td>
</tr>
<tr>
<td>• User-defined enumeration types</td>
</tr>
<tr>
<td>• User-defined compound types</td>
</tr>
<tr>
<td>• User-defined variable-length types</td>
</tr>
<tr>
<td>• Multiple unlimited dimensions</td>
</tr>
<tr>
<td>• String type</td>
</tr>
<tr>
<td>• Additional numeric types</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Why Not Make Use of NetCDF-4 Data Model Now?</th>
</tr>
</thead>
<tbody>
<tr>
<td>• C-based netCDF-4 software still only in</td>
</tr>
<tr>
<td>beta release (depending on HDF5 1.8</td>
</tr>
<tr>
<td>release)</td>
</tr>
<tr>
<td>• Few netCDF utilities or applications</td>
</tr>
<tr>
<td>adapted to full netCDF-4 model yet</td>
</tr>
<tr>
<td>• Development of useful conventions will</td>
</tr>
<tr>
<td>take experience, time</td>
</tr>
<tr>
<td>• Significant performance improvements</td>
</tr>
<tr>
<td>available now, <em>without</em> netCDF-4 data</td>
</tr>
<tr>
<td>model</td>
</tr>
<tr>
<td>– using classic model with netCDF-4</td>
</tr>
<tr>
<td>format</td>
</tr>
</tbody>
</table>
When to Use NetCDF-4 Data Model

- On “greenfield projects” (lacking legacy issues or constraints of prior work)
- If non-classic primitive types needed
  - 64-bit integers for statistical applications
  - unsigned bytes, shorts, or ints for wider range
  - real strings instead of fixed-length char arrays
- If making data self-descriptive requires new user-defined types
  - compound
  - variable-length
  - enumerations
  - nested combinations of types
- If multiple unlimited dimensions needed
- If groups needed for organizing data in hierarchical name scopes

Recommendations for Data Providers

- Continue using classic data model and format, if suitable
- Evaluate practicality and benefits of classic model with netCDF-4 format
- Test and explore uses of extended netCDF-4 data model features
- Help evolve netCDF-4 conventions and Best Practices based on experience with what works
Best Practices: Where to Go From Here

- We’re updating current netCDF-3 Best Practices document before Workshop in July
- New “Developing Conventions for NetCDF-4” document is under development
- Benchmarks may help with guidance on compression, chunking parameters, use of compound types
- We depend on community experience for distillation into new Best Practices

Adoption of NetCDF-4: A 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 application developers will confront to support new conventions
Thanks!

Questions?