Developing Conventions for NetCDF-4

Russ Rew and John Caron, UCAR Unidata Program
Last revised October 15, 2014

Background: NetCDF and Conventions

Since 1988, netCDF user documentation has recommended use of conventions for representing meaning in data and for encouraging interoperability between data providers, application developers, and data users. *User Guide Conventions* refer to recommendations that appear in the netCDF User Guide. User Guide Conventions are intended to be general enough to apply to any kind of data represented in netCDF form, including data that is not earth-referenced, for example.

User Guide Conventions include recommendations such as:

- Use of the same name for a variable as for a dimension to represent coordinates along that dimension
- A "units" attribute to store a string representation of units of measurement
- "scale_factor" and "add_offset" attributes for simple packing of floating-point values into smaller 8- or 16-bit integers
- A "_FillValue" attribute to represent data that has not been written or that is missing
- A "Conventions" attribute to declare which discipline-specific, project-specific, or community-specific conventions a data file complies with
- A "Coordinates" attribute (adopted from CF Conventions) to list names of auxiliary coordinate variables and, optionally, coordinate variables.

User Guide Conventions are intended to be independent of scientific discipline. Some are also independent of particular languages, such as the identity of variable and dimension names to identify coordinate variables. User Guide Conventions are intended to provide general solutions that anticipate needs of data providers, applications developers, and data services. They deal with issues that need a published standard to support interoperability.

User Guide Conventions are not intended to be comprehensive. More specific conventions are required for particular projects, disciplines, or communities. The climate and forecast community recognized that Users Guide conventions were not sufficient for writing generic analysis and visualization applications for climate and forecast model output datasets. This led to the development and publishing of the Climate and Forecast (CF) Conventions (www.cfconventions.org), designed to support self-description, to be easy to use for both data writers and readers, to be effectively understandable by both humans and programs, and to minimize redundancy in representation.

Development of the CF Conventions provides an example of how a community can agree on standard ways to represent quantities and coordinate systems within the simple framework provided by netCDF-3, using only dimensions, variables, attributes, and a limited set of six primitive types. In particular, the "standard_name" attribute is now used to represent thousands of observed and modeled quantities. CF also provides standard representations for grid cell bounds and grid cell measures.

The CF Conventions have achieved primacy among netCDF conventions more comprehensive than the Users Guide Conventions. Many earth-science datasets use CF-compliance as a brand of interoperability, and handling CF-compliant data has become an important requirement for servers, clients, and applications.

The CF Conventions originally focused on gridded model outputs. Since the addition of a chapter on Discrete Sampling Geometries and an appendix of associated examples, standard representations for many kinds of observational data sets are now also available.

With the development and release of netCDF-4, an enhanced data model is available. Added features in netCDF-4, such as groups, compound types, and variable-length types, provide new ways to represent metadata and opportunities to improve existing conventions. Useful conventions evolve over time from experience of data providers, application developers, and users of the data. Each new convention adopted potentially adds work for developers of compliant applications and for providers of compliant data.

Development of a new set of netCDF-4 conventions will require careful attention to compatibility concerns as well as realization that delay to achieve perfect consensus on a comprehensive and general convention is not always practical. Delay may result in the undesirable use of incompatible conventions and loss of interoperability by data providers who have a timely need to make data available.

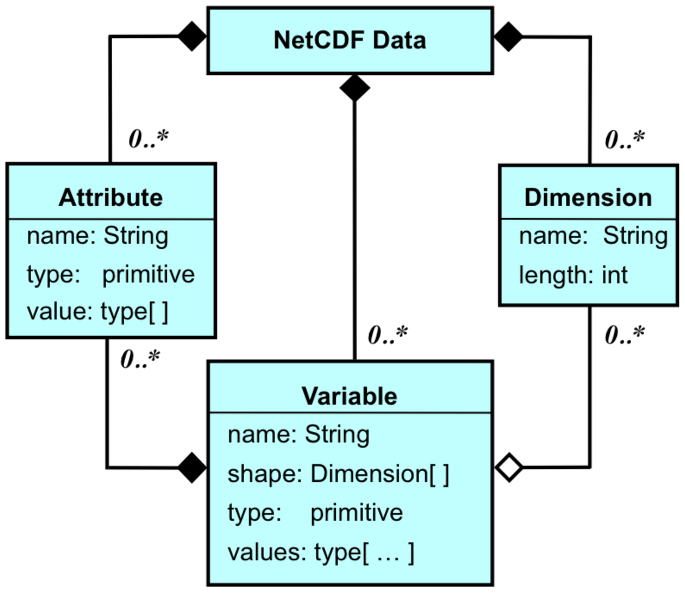
NetCDF Data Models

Two important data models for netCDF are

- the "classic" netCDF model, used for netCDF-3 and earlier versions
- the "enhanced" data model, used for netCDF-4, with features made possible by its use of HDF5 as a storage layer

The classic netCDF model represents data sets using named *variables*, *dimensions*, and *attributes*. A variable is a multidimensional array whose elements are all of the same type. A variable may also have attributes, which are associated named values. Each variable has a shape, specified by its dimensions, named axes that have a length. Variables may share dimensions, indicating a common grid. One dimension may be of unlimited length, so data may be efficiently appended to variables along that dimension. Variables and attributes have one of six primitive data types: char, byte, short, int, float, or double.

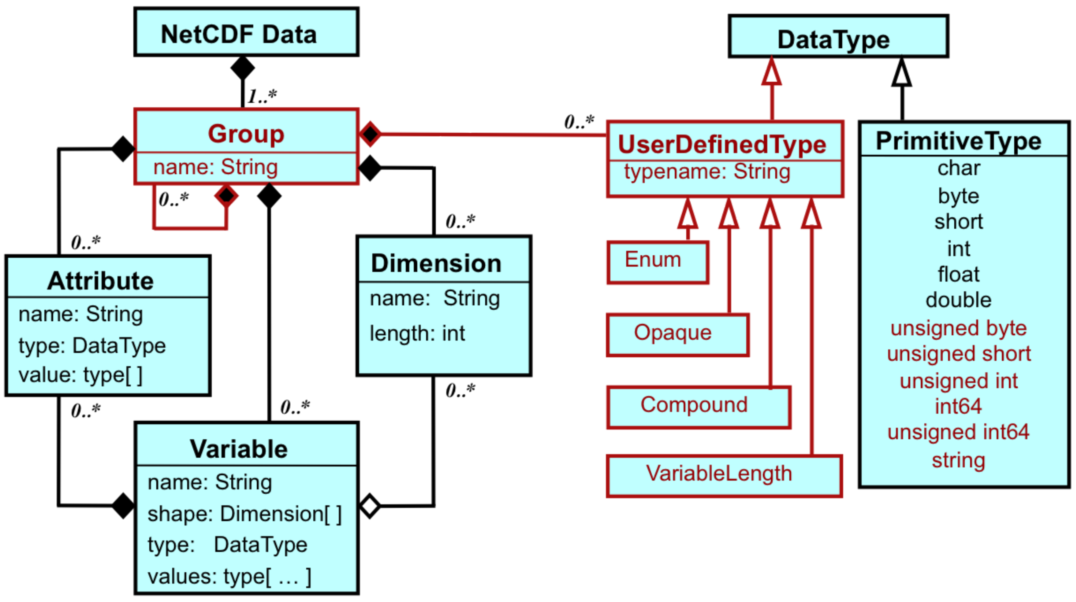
A Unified Modeling Language (UML) diagram of the classic netCDF Data Model exhibits its simplicity:



Limitations of the classic data model include lack of support for data structures other than multidimensional arrays and lists. In particular, nested structures and ragged arrays are not easily represented. Only one shared unlimited dimension per file means some datasets must use multiple files. A flat name space for dimensions and variables limits scalability. Character arrays can represent strings, but require the user to explicitly deal with their length. Lack of unsigned types and 64-bit integer types precludes some applications.

The netCDF-4 data model, implemented using an HDF5-based storage layer, deals with these limitations. In the enhanced data model, a file has a top-level, unnamed *group*. Each group may contain one or more named variables, dimensions, attributes, groups, and types. A variable is still a multidimensional array whose elements are all of the same type. Each variable may have attributes, and each variable's shape is specified by its dimensions, which may be shared. However, in the enhanced data model, one *or more* dimensions may be of unlimited length, so data may be efficiently appended to variables along any of those dimensions. Variables and attributes have one of twelve primitive data types or one of four kinds of user-defined types.

A UML diagram of the enhanced netCDF data model used for netCDF-4 shows (in red) what it adds to the classic netCDF data model:



Because preserving future access to archived data is important, the netCDF-4 data model, data format, and software are designed to provide compatibility with and continued support for netCDF-3 data and applications.

Use of NetCDF-4 Model Features

New projects that lack legacy issues or constraints from need for interoperability with existing applications, are already experimenting with the enhanced data model. Features attracting the most interest include

- additional primitive types, including 64-bit integers, unsigned integers, and strings
- compound types, to provide arrays of portable structures
- variable-length types, for ragged arrays
- multiple unlimited dimensions, for flexibility in appending data
- groups, for nested name spaces
- enumerated types, for compact self-description
- nested combinations of user-defined types, for complex data structures

Some features of the new data model may be adopted and supported earlier in applications than other features. It is possible that some features may not be widely used or supported by third-party software. For example, support for groups and strings is easier than supporting arbitrarily nested user-defined types.

Below, we discuss benefits that may be obtained for new projects that lack legacy issues or constraints of interoperability with current systems, by making use of new netCDF-4 features. Interspersed with discussion of features, we identify some new conventions issues not encountered when using the classic data model.

For the examples presented, we use the netCDF-4 Common Data Language (CDL) notation to show the structure of the data and metadata, as produced by the netCDF-4 ncdump utility and interpreted by the netCDF-4 ncgen utility.

Uses for Groups

Groups provide nested scopes for names, similar to directories in a file system. Just as files in different directories may have the same names, variables in different groups may also have the same names. A netCDF group is analogous to a netCDF file, with its own set of named dimensions, variables, attributes, types, and subgroups. Names for objects in groups may be specified using a "/" separator to identify their location in the group hierarchy, just as with file systems.

Here is an example use of groups to organize data by a named property, in this case geographical regions:

```
group: UnitedStates {
  dimensions: time = unlimited;
  variables: float average_temperature(time);
  group: Washington {
    dimensions: time = unlimited, stations = 47;
    variables: float temperature(time, stations);
  }
  group: Oregon {
    dimensions: time = unlimited, stations = 61;
    variables: float temperature(time, stations);
  }
  group: California {
    dimensions: time = unlimited, stations = 53;
    variables: float temperature(time, stations);
  }
}
```

In the above example, each inner group has its own variable named "temperature", its own dimension named "stations", and its own unlimited dimension named "time". (The enhanced data model also permits multiple unlimited dimensions within a single group or without using groups.)

Potential uses for groups include applications that require:

- containers to "factor out" common information, such as for regions, grids, or model ensembles
- hierarchies to organize a large number of variables
- separate name scopes, so that multiple sets of data may use the same names for dimensions, variables, and group-level attributes
- sequences of analysis steps, from raw data to derived products
- containers for storing closely coupled data, such as instrument calibration parameters and instrument sensor descriptions

User-Defined Types

The netCDF-4 data model makes available several kinds of user-defined types: compound types, enumerations, variable-length types, and opaque types. Each type has a name and a definition. Named types are contained in groups, but may be referenced in type definitions in other groups. Both variables and attributes may be declared to be of user-defined types, which allows a natural extension of conventions that require some variable attributes to be of the same type as the variable, for example _FillValue.

Types exist independently of variables or attributes that use them, so it is possible for a type to be contained in a netCDF group even though no variables or attributes are declared to be of that type. This may be useful for declaring types to be used for data to be added later or as templates for derived data objects.

Since each type requires a name, proliferation of names suggests it may be useful to have a convention for type names to easily distinguish them from variable names. In the examples below, we add the "_t" suffix to type names to make them easier to identify.

Uses for Compound Types

Compound types are like C structures, grouping together named fields (also called "members"), that may be of different types, into a structure that may be accessed as a unit. For example:

```
types:
  compound wind_vector_t {
    float eastward ;
    float northward ;
    }
dimensions:
    lat = 18 ;
    lon = 36 ;
    pres = 15 ;
    time = 4 ;
variables:
    wind_vector_t  wind(time, pres, lat, lon) ;
```

```
             wind:standard_name = "geostrophic_wind_vector" ;
      data:
          wind = {0, 0}, {10, 20}, {20, 10}, {15, 15}, {20, -5.5}, ...;
```

defines a wind vector type with two members, `eastward` and `northward`. The standard_name "geostrophic_wind_vector" is not currently a valid CF standard name but a plausible future standard name for a structure of the scalar quantities currently identified by standard names "geostrophic_eastward_wind" and "geostrophic_northward_wind". If closely related data values will be accessed together, use of such a compound type should be recommended for clarity and efficiency.

This brings up new conventions issues in the use of compound types. Does use of a particular standard name also imply use of a standard type for the associated compound type? For example should a quantity whose standard name includes "_wind_vector" be of a compound type equivalent to the `wind_vector_t` type defined in the example? If so, are the member names of the compound type also part of the convention for this quantity?

As another compound type example, consider this representation of point observation data:

```
      types:
        compound wind_vector_t {
          float eastward ;
          float northward ;
          }
        compound ob_t {
            int station_id ;
            double time ;
            float temperature ;
            float pressure ;
            wind_vector_t wind ;
        }
      dimensions:
          stations = unlimited ;
      variables:
          ob_t obs(stations) ;
      data:
          obs = {42, 0.0, 20.5, 950.0, {2.5, 3.5}}, ;
```

Compound types may be nested, as the above example shows with the use of a `wind` member of type `wind_vector_t`.

Potential uses for compound types include

- Representing vector quantities like velocities
- Bundling other kinds of multiple *in situ* observations together (profiles, soundings)
- Modeling relational database tuples
- Representing other kinds of objects that have components
- Representing C structures in a portable form

Member fields of a type have a name and a type, but are *not* netCDF variables. In particular, there is no way to directly assign variable attributes to them, but we next propose a convention to handle this.

Convention for Assigning Attributes to Members of Compound Type

This issue is discussed in NetCDF-4 Compound Attributes, based on experiments and experience with using netCDF-4 compound types to represent observational data.

Use of Variable-Length Types

Named variable-length types may be created for any netCDF-4 base type, to represent one-dimensional arrays of variable length. In netCDF-4, these currently must be read atomically, that is the entire one-dimensional array must be accessed with one function call to access its length and the values of the data.

Here is an example of using nested variable-length types to represent marine data:

```
types:
  compound obs_t {                // type for a single observation
    float pressure ;
    float temperature ;
    float salinity ;
  }
  obs_t(*) some_obs_t ;           // type for some observations
  compound profile_t {            // type for a single profile
    float latitude ;
    float longitude ;
    int time ;
    some_obs_t obs ;
  }
  profile_t(*) some_profiles_t ;  // type for some profiles
  compound track_t {              // type for a single track
    string id ;
    string description ;
    some_profiles_t profiles ;
  }

dimensions:
  tracks = 42 ;

variables:
  track_t cruise(tracks) ;         // this cruise had 42 tracks
```

The above defines 42 tracks, each of which is of compound type containing an id, a description, and a variable number of profiles. Each profile comprises a location, time, and a variable number of observations. Each observation is a compound structure of pressure, temperature, and salinity.

Potential uses for variable-length types include ragged arrays and *in situ* observational data typical of soundings, profiles, and time series. For a variable-length type, any base type may be used, including a compound type or another variable-length type. There is no associated shared dimension, and the value of a variable-length type is currently accessed all at once, for example a whole row of a ragged array. Access to one base value at a time of variable length types may be supported soon in some language interfaces by iterators. That means accessing a single value of the `cruise` array in the example above would read all of the associated profiles, each containing its own set of observations that would also be read into memory.

It may be useful to distinguish variable length types with a prefix such as "some_" or "list_of_" as in the example above. A convention for names of variable-length types might enhance interoperability with other data models and make declarations of complex types more easily understood. However, such a convention may be too English-centric for international use.

Use of Enumerations

Enumerated types may be used to represent a small number of named values more concisely than strings, because small numeric values are stored even though the corresponding text symbols are displayed when the data is dumped. For example, consider the example:

```
types:
  byte enum cloud_t {
    Clear = 0, Cumulonimbus = 1, Stratus = 2, Stratocumulus = 3,
    Cumulus = 4, Altostratus = 5, Nimbostratus = 6, Altocumulus = 7,
    Cirrostratus = 8, Cirrocumulus = 9, Cirrus = 10, Missing = 127
  } ;
dimensions:
    time = unlimited ;
variables:
    cloud_t primary_cloud(time) ;
         cloud_t  primary_cloud:_FillValue = Missing ;
data:
    primary_cloud = Clear, Stratus, Clear, Cumulonimbus, Missing ;
```

Each data value of the variable `primary_cloud` in the example above only requires one byte of storage. Using an array of fixed-length strings instead, as required in netCDF-3, would use at least 13 bytes of storage for each value, in order to reserve enough space to store the longest string "`Stratocumulus`".

Enumeration types can improve self-description while keeping data compact. They provide a better alternative to using strings for flags for such purposes as data quality indicators, soil type, cloud type, and similar situations where a small fixed set of non-numeric values are appropriate. In the CF Conventions, the attributes `flag_meaning` and `flag_values` are used for this purpose, but using an enumeration type may be somewhat simpler.

A potential conventions issue is whether the gain in simplicity is worth the cost of a new convention. If enumerations are used, would the enumeration symbols also be standardized, as the names for standard quantities are in the standard names table? Would a convention be needed to associate a more descriptive string for each enumeration symbol?

Cautions Regarding Use of Unsigned Integers

Unsigned integers are not a supported type in some programming languages, such as Fortran and Java. In these languages, n-bit unsigned integer values may have to be read into signed integers with more bits to ensure values are preserved. For example, unsigned 16-bit shorts might need to be read into 32-bit signed integers. This is especially problematical for the unsigned 64-bit type, for which no integer type may be available wide enough to hold large unsigned values. As a general recommendation, avoid using the unsigned 64-bit integer type for data that someone might want to read using Fortran, Java, or other languages not supporting this type.

Use of Strings

The primitive type "string" is available in the netCDF-4 data model for variable-length strings. Arrays of strings are useful for representing multiple lines of text, lists of variable-length text values, and similar applications. However string is a new primitive type, not available to netCDF-3 C and Fortran APIs. It is not compatible with netCDF-3 applications. Data providers must weigh the convenience of using the string primitive type against the adaptation that will be required for software to access string data.

Currently, long multi-line strings used for metadata such as "history", "source", or "references" global attributes, must use embedded newline characters "\n" to separate lines. With an attribute of string type, arrays of lines may be represented without "\n" separators, and the values of such attributes will be displayed with one string value per line.

Names for netCDF variables, dimensions, attributes, groups, user-defined types, and enumeration constants use UTF-8 for character encoding. Variables with the string data type are assumed to be UTF-8 encoded, although the netCDF C library does not yet enforce this encoding (as it does for names). It would be possible to use another encoding, for example by adding a standard attribute to the variable. This is not currently implemented, and we are waiting for use cases to clarify the need for it, if any.

Conclusion

A principle for the CF-1 Conventions that has been important to their success and widespread use is that

> *Conventions [should be] developed only for known issues. Instead of trying to foresee the future, features are added as required.*

Development of CF Conventions for netCDF-4 should honor this principal until there is enough experience with using features of the enhanced data model to guide the evolution of conventions.

In general, it seems wise to avoid replacing an existing adequate convention with a better alternative convention that uses netCDF-4 features unless there is some significant advantage to the new convention, because applications will have to accept both the old and new conventions.

There is still little experience with representing model outputs in the new data model. Most of the experience so far is with use of the netCDF-4 format and the classic data model. Gaining the experience needed to provide guidance for use of new features requires data providers, users, and tool developers. Application developers are likely to delay supporting netCDF-4 features until it's clear which features will prove useful for representing the next generation of model output archives and observational datasets.

Best practices that become crystallized into new conventions develop out of the experience of data providers, application developers, and users. These considerations mean a definitive set of user guide and CF conventions for netCDF-4 may take a while to develop and mature. More usage examples and draft proposals from users, developers, and data providers will advance the process.