Open Geospatial Consortium Inc.

Date: 2009-10-13

Reference number of this OGC® project document: **OGC 09-122**

Version: 0.1.0

Category: OGC® Public Discussion Paper

Editor: Ben Domenico

Draft CF-netCDF Specification 0.1.0

Copyright

Warning

Document type: Draft OGC® Discussion Paper
Document subtype: Candidate standard
Document stage: Draft – for discussion only
Document language: English

Table of Contents

The following OGC clauses “i” through “v” shall be included in an OGC Implementation Specification, but are not included in most other OGC documents.

i. Preface

This is an OGC® Candidate Standard for encoding binary representations of georeferenced data.

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

ii. Submitting organizations

The following organizations submitted this Discussion Paper to the Open Geospatial Consortium Inc.

a) The University Corporation for Atmospheric Research

iii. Submission contact points

All questions regarding this submission should be directed to the editor or the submitters:

CONTACT	COMPANY
Ben Domenico, editor	Unidata Program Center, UCAR
Russ Rew	Unidata Program Center, UCAR
Glenn Davis, Ethan Davis, Dennis Heimbigner, Ed Hartnett John Caron	Unidata Program Center, UCAR

iv. Revision history

Date	Release	Author	Paragraph modified	Description
2009-09-09	0.1.0	Ben Domenico		First version in OGC document template
2009-10-10	0.1.0	Carl Reed	Various	Prepare for publications as DP

v. Changes to the OGC® Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate this OGC® standard.

Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all such patent rights. However, to date, no such rights have been claimed or identified.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

Introduction and Background

NetCDF (network Common Data Form) is a data model for array-oriented scientific data, a freely distributed collection of access libraries implementing support for that data model, and a machine-independent format. Together, the interfaces, libraries, and format support the creation, access, and sharing of scientific data.

NetCDF has been formally recognized by US Government standards bodies: the NASA Earth Standards Data Systems Working Groups (ESDSWG) Standards Process Group (SWG) and the Data Management and Communications Committee of the NOAA Integrated Ocean Observing System. The goal of this standards effort is to have the NetCDF with CF (Climate and Forecast) Conventions recognized as an international standard for encoding georeferenced data in binary form.

The OGC has developed a broad Standards Baseline. UCAR and the OGC believe that having netCDF included in that family will encourage broader use and greater interoperability among clients and servers interchanging data in binary form. Establishing CF-netCDF as an standard for binary encoding will make it possible to incorporate standard delivery of data in binary form via several OGC protocols, e.g., WCS, WFS, and SOS.

The OGC WCS standards working group is already developing an extension to the core WCS for delivery of data encoded in CF-netCDF. This standards effort is seen as complementary to that in WCS and hopefully will facilitate similar extensions for other standard protocols.

OGC Encoding specification: CF-netCDF

1 Scope

NetCDF data is intended to make possible the creation of collections of data that are:

- Self-Describing: NetCDF datasets include information about the data they contain.
- Portable: Computers with different ways of storing integers, characters, and floatingpoint numbers can access netCDF data.
- Direct-access: A small subset of a large data set may be accessed efficiently, without first reading through all the preceding data.
- Appendable: Data may be appended to a properly structured netCDF file without copying the data set or redefining its structure.
- Sharable: One writer and multiple readers may simultaneously access the same netCDF file. Using parallel netCDF interfaces, multiple writers may write a file concurrently.
- Archivable: Access to current and earlier forms of netCDF data will be supported by current and future versions of the software.

2 Conformance

A netCDF resource that conforms to this standard shall:

a) satisfy all requirements stipulated in this document;
b) pass all relevant test cases specified by the Abstract Test Suite (ATS) provided in OGC document 09-122.

<<<Need to address the issue of an Abstract Test Suite??? >>>

3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this part of OGC 09-122. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply; however, parties to agreements based on this part of 09-122 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

NASA ESDS-RFC-011v1.00 R. Rew, E. Hartnett, D. Heimbigner, E. Davis, J. Caron: *NetCDF Classic and 64-bit Offset File Formats*

http://www.esdswg.org/spg/rfc/esds-rfc-011/ESDS-RFC-011v1.00.pdf

Unidata UCAR, NetCDF Reference Document, 2009
http://www.unidata.ucar.edu/software/netcdf/docs/

Unidata UCAR, NetCDF User Guide
http://www.unidata.ucar.edu/software/netcdf/docs/netcdf.html

http://www.unidata.ucar.edu/software/netcdf/docs/netcdf.html
Unidata UCAR, NetCDF Reference Implementations
ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf.tar.gz

NetCDF Climate and Forecast (CF) Metadata Convention
http://cf-pcmdi.llnl.gov/

4 Terms and symbols

4.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply /the terms and definitions given in … and the following apply.

4.2 Acronyms (and abbreviated terms)

Some frequently used abbreviated terms:

API	Application Program Interface
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf

DCE	Distributed Computing Environment
DCP	Distributed Computing Platform
DCOM	Distributed Component Object Model
IDL	Interface Definition Language
ISO	International Organization for Standardization
OGC	Open Geospatial Consortium
UML	Unified Modeling Language
XML	eXtended Markup Language
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional

5 Document Conventions

5.1 UML Notation

The diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram.

6 netCDF Core Standard

In different contexts, "netCDF" may refer to an abstract data model, a software implementation with associated application program interfaces (APIs), or a data format. Confusion may easily arise in discussions of different versions of the data models, software, and formats, because the relationship among versions of these entities is more complex than a simple one-to-one correspondence by version. For example, compatibility commitments require that new versions of the software support all previous variants of the format and data model.

To avoid this potential confusion, we assign distinct names to versions of the formats, data models, and software releases that will be used consistently in the remainder of this document.

This document formally specifies two format variants, the classic format and the 64-bit offset format for netCDF data. It also informally describes two additional format variants, the netCDF-4 format and the netCDF-4 classic model format.

6.1 The Classic Data Model

The classic model represents information in a netCDF data set using dimensions, variables, and attributes, to capture the meaning of array-oriented scientific data. Figure 1 presents a simplified UML diagram of the classic data model. Variables hold data values. In the classic model, a variable can hold a multidimensional array of values of the same type. A variable has a name, type, shape, attributes, and values. The shape of a variable is specified with a list of zero or more dimensions:

- 0 dimensions: a scalar variable with only one value
- 1 dimension: a 1-D (vector) variable
- 2 dimensions: a 2-D (matrix or grid) variable
- ...

Dimensions are used to specify variable shapes, common grids, and coordinate systems. A dimension has a name and a length. Dimensions may be shared among variables, indicating a common grid. Dimensions may be associated with coordinate variables to identify coordinate axes. In the classic model, at most one dimension can have the unlimited length, which means variables can grow along that dimension. Record dimension is another term for an unlimited dimension. (In the enhanced model, multiple dimensions can have the unlimited length.) Attributes hold metadata (data about data). An attribute contains information about properties of a variable or an entire data set. Variable attributes may be used to specify properties such as units. Attributes that apply to a whole data set, also called global attributes, may be used to record properties of all the data in a file, such as processing history or conventions used. An attribute may have zero, one, or multiple values (1-D), but attributes cannot be multidimensional.

NetCDF conventions are defined primarily in terms of attributes. Thus the names of attributes are typically standardized in conventions rather than the names of variables.

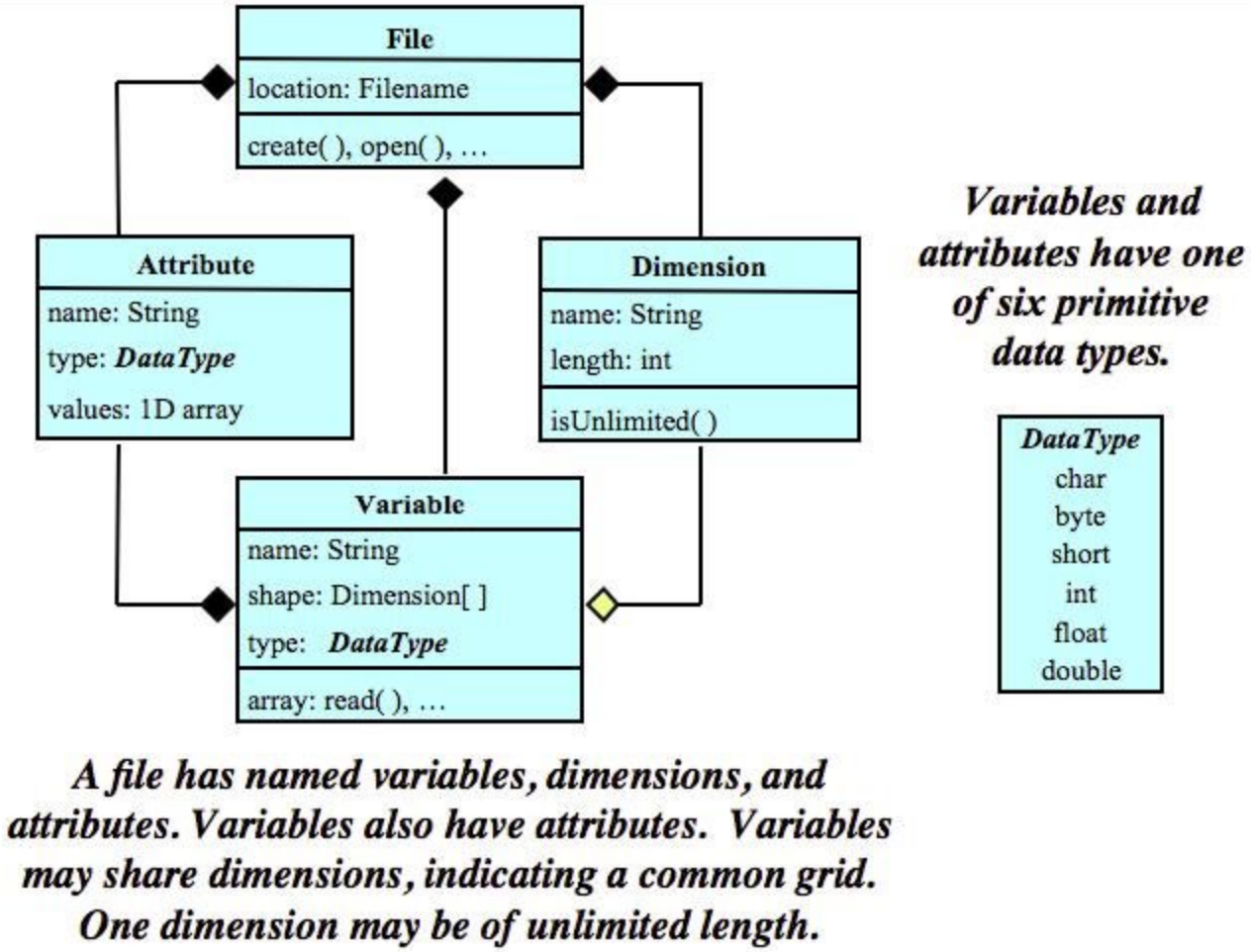

Figure 1 The netCDF "classic" data model

For a more comprehensive explanation of the netCDF data model, see the NetCDF User's Guide [1] or the online NetCDF Workshop for Developers and Data Providers [3].

6.2 The NetCDF-4 Format

The netCDF-4 format implements and expands the classic model by using an enhanced version of HDF5 [7] as the storage layer. Use is made of features that are only available in HDF5 version 1.8 and later. Using HDF5 as the underlying storage layer, netCDF-4 files remove many of the restrictions for classic and 64-bit offset files. The richer enhanced model supports user-defined types and data structures, hierarchical scoping of names using groups, more primitive types including strings, larger variable sizes, and multiple unlimited dimensions.

The underlying HDF5 storage layer also supports per-variable compression, multidimensional tiling, and efficient dynamic schema changes, so that data need not be copied when adding new variables to a file schema. Although every file in netCDF-4 format is an HDF5 file, there are HDF5 files that are not netCDF-4 format files, because the netCDF-4 format intentionally uses a limited subset of the HDF5 data model and file

format features. Some HDF5 features not supported in the netCDF enhanced model and netCDF-4 format include non-hierarchical group structures, HDF5 reference types, multiple links to a data object, user-defined atomic data types, stored property lists, more permissive rules for data object names, the HDF5 date/time type, and attributes associated with user-defined types.

6.3 The NetCDF-4 Classic Model Format

Every classic and 64-bit offset file can be represented as a netCDF-4 file, with no loss of information. There are some significant benefits to using the simpler netCDF classic model with a netCDF-4 file format. For example, software that writes or reads classic model data can write or read netCDF-4 classic model format data by recompiling/relinking to a netCDF-4 API library, with no or only trivial changes needed to the program source code. The netCDF-4 classic model format supports this usage by enforcing rules on what functions may be called to store data in the file, to make sure its data can be read by older netCDF applications (when relinked to a netCDF-4 library).

Writing data in this format prevents use of enhanced model features such as groups, added primitive types not available in the classic model, and user-defined types. However performance features of the netCDF-4 formats that do not require additional features of the enhanced model, such as per-variable compression and chunking, efficient dynamic schema changes, and larger variable size limits, offer potentially significant performance improvements to readers of data stored in this format, without requiring program changes.

7 Classic Format Specification

7.1 Informal Description

To make the formal description more easily understood, we begin with an informal description of the classic format, which also applies to the 64-bit offset variant. Understanding the format at this level can make clear which netCDF operations are expensive, for example adding a new variable to an existing file.

A classic or 64-bit offset file is stored in three parts:

1. The header, containing information about dimensions, attributes, and variables
2. The fixed-size data, containing data values for variables that don't have an unlimited dimension
3. The record data, containing data values for variables that have an unlimited dimension

The header has information about the dimensions, variables, and attributes, including all the attribute values. There is typically little extra space in the header, unless such space is reserved when the file is created. This is why the dimensions, variables, and attributes in

a netCDF file are typically defined when the file is created, before any data is written. Operations that require the header to grow force moving all the data by copying it.

Only one unlimited dimension, the record dimension, is permitted in classic model files. The current size of the record dimension is stored in the header, which specifies how many records the file contains. New data may be efficiently added to record variables (variables that use the record dimension in specifying their shape) along the record dimension.

The fixed-size data part has all the data for each non-record variable. The data for each variable is stored contiguously, in row-major order for multi-dimensional variables.

Each record in the record data part is similar to the fixed-size data part, containing all the data for that record for each record variable. Each record's worth of data for each record variable is stored contiguously, in row major order for multidimensional variables. All records are the same size, because they each contain all the data for a particular record for each record variable.

7.2 Formal Specification of the Classic Format

To present the format more formally, we use a BNF grammar notation. In this notation:

- Non-terminals (entities defined by grammar rules) are in lower case.
- Terminals (atomic entities in terms of which the format specification is written) are in upper case, and are specified literally as US-ASCII characters within single-quote characters or are described with text between angle brackets (‘<’ and ‘>’).
- Optional entities are enclosed between braces (‘[’ and ‘]’).
- A sequence of zero or more occurrences of an entity is denoted by ‘[entity ...]’.
- A vertical line character (‘|’) separates alternatives. Alternation has lower precedence than concatenation.
- Comments follow ‘//’ characters.
- A single byte that is not a printable character is denoted using a hexadecimal number with the notation ‘\xDD’, where each D is a hexadecimal digit.
- A literal single-quote character is denoted by ‘\'', and a literal back-slash character is denoted by ‘\\’.

Following the grammar, a few additional notes are included to specify format characteristics that are impractical to capture in a BNF grammar, and to note some

special cases for implementers. Comments in the grammar point to the notes and special cases, and help to clarify the intent of elements of the format.

```
netcdf_file    = header data
header         = magic numrecs dim_list gatt_list var_list
magic          = 'C' 'D' 'F' VERSION
VERSION        = \x01 |                  // classic format
                 \x02                    // 64-bit offset format
numrecs        = NON_NEG | STREAMING     // length of record dimension
dim_list       = ABSENT | NC_DIMENSION nelems [dim ...]
gatt_list      = att_list                // global attributes
att_list       = ABSENT | NC_ATTRIBUTE nelems [attr ...]
var_list       = ABSENT | NC_VARIABLE nelems [var ...]
ABSENT         = ZERO ZERO               // Means list is not present
ZERO           = \x00 \x00 \x00 \x00     // 32-bit zero
NC_DIMENSION   = \x00 \x00 \x00 \x0A     // tag for list of dimensions
NC_VARIABLE    = \x00 \x00 \x00 \x0B     // tag for list of variables
NC_ATTRIBUTE   = \x00 \x00 \x00 \x0C     // tag for list of attributes
nelems         = NON_NEG                 // number of elements in following
sequence
dim            = name dim_length
name           = nelems namestring
                      // Names a dimension, variable, or attribute.
                     // Names should match the regular expression
                     //([a-zA-Z0-9_]|{MUTF8})([^\x00-\x1F/\x7F-\xFF]|{MUTF8})*
                     // For other constraints, see “Note on names”, below.
namestring     = ID1 [IDN ...]
ID1            = alphanumeric | '_'
IDN            = alphanumeric | special1 | special2
alphanumeric   = lowercase | uppercase | numeric | MUTF8
lowercase      = 'a'|'b'|'c'|'d'|'e'|'f'|'g'|'h'|'i'|'j'|'k'|'l'|'m'|
                 'n'|'o'|'p'|'q'|'r'|'s'|'t'|'u'|'v'|'w'|'x'|'y'|'z'
uppercase      = 'A'|'B'|'C'|'D'|'E'|'F'|'G'|'H'|'I'|'J'|'K'|'L'|'M'|
                 'N'|'O'|'P'|'Q'|'R'|'S'|'T'|'U'|'V'|'W'|'X'|'Y'|'Z'
numeric        = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
                      // special1 chars have traditionally been
                      // permitted in netCDF names.
special1       = '_'|'.'|'@'|'+'|'-'


                      // special2 chars are recently permitted in
                      // names (and require escaping in CDL).
                      // Note: '/' is not permitted.
special2       = ' ' | '!' | '"' | '#' | '$' | '%' | '&' | '\'' |
                 '(' | ')' | '*' | ',' | ':' | ';' | '<' | '=' |
                 '>' | '?' | '[' | '\\' | ']' | '^' | '`' | '{' |
                 '|' | '}' | '~'
MUTF8          =
dim_length     = NON_NEG       // If zero, this is the record dimension.
                               // There can be at most one record dimension.
attr           = name nc_type nelems [values ...]
nc_type        = NC_BYTE | NC_CHAR | NC_SHORT | NC_INT | NC_FLOAT | NC_DOUBLE
var            = name nelems [dimid ...] vatt_list nc_type vsize begin
                               // nelems is the dimensionality (rank) of the
                               // variable: 0 for scalar, 1 for vector, 2
                               // for matrix, ...
dimid          = NON_NEG       // Dimension ID (index into dim_list) for
                               // variable shape. We say this is a "record
                               // variable" if and only if the first
                               // dimension is the record dimension.
vatt_list      = att_list      // Variable-specific attributes
```

```
vsize          = NON_NEG       // Variable size. If not a record variable,
                               // the amount of space in bytes allocated to
                               // the variable's data. If a record variable,
                               // the amount of space per record. See “Note on
                               // vsize” below.
begin          = OFFSET        // Variable start location. The offset in
                               // bytes (seek index) in the file of the
                               // beginning of data for this variable.
data           = non_recs recs
non_recs       = [vardata ...] // The data for all non-record variables,
                               // stored contiguously for each variable, in
                               // the same order the variables occur in the
                               // header.
vardata        = [values ...]  // All data for a non-record variable, as a
                               // block of values of the same type as the


                               // variable, in row-major order (last
                               // dimension varying fastest).
recs           = [record ...]  // The data for all record variables are
                               // stored interleaved at the end of the
                               // file.
record         = [varslab ...] // Each record consists of the n-th slab
                               // from each record variable, for example
                               // x[n,...], y[n,...], z[n,...] where the
                               // first index is the record number, which
                               // is the unlimited dimension index.
varslab        = [values ...]  // One record of data for a variable, a
                               // block of values all of the same type as
                               // the variable in row-major order (last
                               // index varying fastest).
values         = bytes | chars | shorts | ints | floats | doubles
string         = nelems [chars]
bytes          = [BYTE ...] padding
chars          = [CHAR ...] padding
shorts         = [SHORT ...] padding
ints           = [INT ...]
floats         = [FLOAT ...]
doubles        = [DOUBLE ...]
padding        = <0, 1, 2, or 3 bytes to next 4-byte boundary>
                               // Header padding uses null (\x00) bytes. In
                               // data, padding uses variable's fill value.
                               // See “Note on padding” below for a special
                               // case.
NON_NEG        =
STREAMING      = \xFF \xFF \xFF \xFF         // Indicates indeterminate record
                                             // count, allows streaming data
OFFSET         =  |        // For classic format or
                       // for 64-bit offset format
BYTE           = <8-bit byte>               // See “Note on byte data”, below.
CHAR           = <8-bit byte>               // See “Note on char data”, below.
SHORT          = <16-bit signed integer, Bigendian, two's complement>
INT            = <32-bit signed integer, Bigendian, two's complement>


INT64          = <64-bit signed integer, Bigendian, two's complement>
FLOAT          = <32-bit IEEE single-precision float, Bigendian>
DOUBLE         = <64-bit IEEE double-precision float, Bigendian>
                               // following type tags are 32-bit integers
NC_BYTE        = \x00 \x00 \x00 \x01        // 8-bit signed integers
NC_CHAR        = \x00 \x00 \x00 \x02        // text characters
NC_SHORT       = \x00 \x00 \x00 \x03        // 16-bit signed integers
NC_INT         = \x00 \x00 \x00 \x04        // 32-bit signed integers
```

```
NC_FLOAT       = \x00 \x00 \x00 \x05        // IEEE single precision floats
NC_DOUBLE      = \x00 \x00 \x00 \x06        // IEEE double precision floats
                           // Default fill values for each type, may be
                           // overridden by variable attribute named
                           // '_FillValue', see "Note on fill values", below
FILL_BYTE      = \x81 // (signed char) -127
FILL_CHAR      = \x00 // null byte
FILL_SHORT     = \x80 \x01 // (short) -32767
FILL_INT       = \x80 \x00 \x00 \x01 // (int) -2147483647
FILL_FLOAT     = \x7C \xF0 \x00 \x00 // (float) 9.9692099683868690e+36
FILL_DOUBLE    = \x47 \x9E \x00 \x00 \x00 \x00 //(double)9.9692099683868690e+36
```

Note on vsize: This number is the product of the dimension lengths (omitting the record dimension) and the number of bytes per value (determined from the type), increased to the next multiple of 4, for each variable. If a record variable, this is the amount of space per record. The netCDF "record size" is calculated as the sum of the vsize's of all the record variables.

The vsize field is actually redundant, because its value may be computed from other information in the header. The 32-bit vsize field is not large enough to contain the size of variables that require more than 232 - 4 bytes, so 232 - 1 is used in the vsize field for such variables.

Note on names: Earlier versions of the netCDF C-library reference implementation enforced a more restricted set of characters in creating new names, but permitted reading names containing arbitrary bytes. This RFC extends the permitted characters in names to include multi-byte UTF-8 encoded[7] Unicode[4] and additional printing characters from the US-ASCII alphabet. The first character of a name must be alphanumeric, a multi-byte UTF-8 character, or '_' (traditionally reserved for names with meaning to implementations, such as the "_FillValue" attribute). Subsequent characters may also include printing special characters, except for '/' which is not allowed in names. Names that have trailing space characters are also not permitted.

Implementations of the netCDF classic and 64-bit offset format must ensure that names are normalized according to Unicode NFC normalization rules [5] during encoding as UTF-8 for storing in the file header. This is necessary to ensure that gratuitous differences in the representation of Unicode names do not cause anomalies in comparing files and querying data objects by name.

Note on streaming data: The largest possible record count, 232-1, is reserved to indicate an indeterminate number of records. This means that the number of records in the file must be determined by other means, such as reading them or computing the current number of records from the file length and other information in the header. It also means that the numrecs field in the header will not be updated as records are added to the file.

Note on padding: In the special case of only a single record variable of character, byte, or short type, no padding is used between data values.

Note on byte data: It is possible to interpret byte data as either signed (-128 to 127) or unsigned (0 to 255). When reading byte data through an interface that converts it into

another numeric type, the default interpretation is signed. There are various attribute conventions for specifying whether bytes represent signed or unsigned data, but no standard convention has been established. The variable attribute "_Unsigned" is reserved for this purpose in future implementations.

Note on char data: Although the characters used in netCDF names must be encoded as UTF-8, character data may use other encodings. The variable attribute "_Encoding" is reserved for this purpose in future implementations.

Note on fill values: Because data variables may be created before their values are written, and because values need not be written sequentially in a netCDF file, default "fill values" are defined for each type, for initializing data values before they are explicitly written. This makes it possible to detect reading values that were never written. The variable attribute "_FillValue", if present, overrides the default fill value for a variable. If _FillValue is defined then it should be scalar and of the same type as the variable.

Fill values are not required, however, because netCDF libraries have traditionally supported a "no fill" mode when writing, omitting the initialization of variable values with fill values. This makes the creation of large files faster, but also eliminates the possibility of detecting the inadvertent reading of values that were not written. 6.3 The 64-bit Offset Format Variant The netCDF 64-bit offset format differs from the classic format only in the VERSION byte, \x02 instead of \x01, and the OFFSET entity, a 64-bit instead of a 32-bit offset from the beginning of the file.

This small format change permits much larger files, but there are still some practical size restrictions. Each fixed-size variable and the data for one record's worth of each record variable are still limited in size to a little less that 4 GiB. The rationale for this limitation is to permit aggregate access to all the data in a netCDF variable (or a record's worth of data) on 32-bit platforms.

8 Extensions Standards for CF Conventions

The conventions for climate and forecast (CF) metadata are designed to promote the processing and sharing of files created with the **NetCDF API**. The CF conventions are increasingly gaining acceptance and have been adopted by a number of **projects and groups** as a primary standard. The conventions define metadata that provide a definitive description of what the data in each variable represents, and the spatial and temporal properties of the data. This enables users of data from different sources to decide which quantities are comparable, and facilitates building applications with powerful extraction, regridding, and display capabilities.

http://cf-pcmdi.llnl.gov/

These CF conventions will be candidates for future extensions to the core CF-netCDF standard.

8.1 What: standard names and units

CF standard names and units

8.2 Where: coordinate systems

CF coordinate systems

8.3 When: time coordinates

CF time coordinates

9 Extension Standards for Applications Programming Interfaces (API)

The netCDF software libraries supplied by UCAR provide read-write access to netCDF files, encoding and decoding the necessary arrays and metadata. The core library is written in C, and provides an API for C, C++ and Fortran applications. An independent implementation, also developed and maintained by Unidata, is written in 100%Java, which extends the core data model and adds additional functionality. Interfaces to netCDF based on the C library are also available in other languages includingR (*ncdf* and *ncvar* packages), Perl, Python, Ruby, Matlab, IDL, and Octave. The specification of the API calls is very similar across the different languages, apart from inevitable differences of syntax. The API calls for version 2 were rather different from those in version 3, but are also supported by version 3 for backward compatibility. Application programmers using supported languages need not normally be concerned with the file structure itself, even though it is available as an open format.

These applications programming interfaces will be candidates for future extensions to the core CF-netCDF standard.

9.1 C language API

http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c/

9.2 C++ API

http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx/

9.3 FORTRAN API

http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77/

http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90/

9.4 Java API

http://www.unidata.ucar.edu/software/netcdf-java/

10 Extension Standard for NcML-GML

NcML-GML is a prime candidate to become an extension standard to the core CF-netCDF standard.

10.1 NcML

NcML is an XML representation of netCDF metadata, (approximately) the header information one gets from a netCDF file with the "ncdump -h" command. NcML is similar to the netCDF CDL (network Common data form Description Language), except, of course, it uses XML syntax.

10.2 NcML-GML

ncML-Gml is:

- an Abstract and Content Model reconciliation schema for ES and GIS info realms
- a Mediation Markup Language between ncML (netCDF Markup Language) and GML
- an extension of ncML core schema, based on GML grammar

At the moment, to support some legacy software packages, ncML-Gml is not a standard GML profile. This will be fixed in a future release.

11 Bibliography

NASA ESDS-RFC-011v1.00 R. Rew, E. Hartnett, D. Heimbigner, E. Davis, J. Caron: *NetCDF Classic and 64-bit Offset File Formats*
http://www.esdswg.org/spg/rfc/esds-rfc-011/ESDS-RFC-011v1.00.pdf

Unidata UCAR, NetCDF Reference Document, 2009
http://www.unidata.ucar.edu/software/netcdf/docs/

Unidata UCAR, NetCDF User Guide
http://www.unidata.ucar.edu/software/netcdf/docs/netcdf.html

http://www.unidata.ucar.edu/software/netcdf/docs/netcdf.html
Unidata UCAR, NetCDF Reference Implementations
ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf.tar.

NetCDF Climate and Forecast (CF) Metadata Convention
http://cf-pcmdi.llnl.gov/
NetCDF C Language Interface Guide
http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-c/

NetCDF C++ Language Interface Guide
http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-cxx/

NetCDF FORTRAN Language Interface Guides
http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f77/
http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-f90/

NetCDF Java Language Interface Guide
http://www.unidata.ucar.edu/software/netcdf-java/

IETF RFC 2616, Hypertext Transfer Protocol – HTTP/1.1. (June 1999)

ISO 8601:2004, Data elements and interchange formats — Information interchange — Representation of dates and times.

ISO 19101:2002. Geographic information -- Reference model

ISO 19107:2003, Geographic Information — Spatial schema.

ISO 19111:—1), Geographic Information — Spatial referencing by coordinates.

ISO 19123: . Abstract Coverage Specification

ISO 19136:2007, Geographic information — Geography Markup Language (GML)

OGC 00-014r1, Guidelines for Successful OGC Interface Specifications

Annex A (normative) Schemas

Annex B (normative) CF-netCDF Coordinate Reference System Definition

Annex A
(normative)

Annex title

A.1 General

A paragraph.