NetCDF Climate and Forecast (CF) Metadata Conventions

Brian Eaton, Jonathan Gregory, Bob Drach, Karl Taylor, Steve Hankin, John Caron, Rich Signell, Phil Bentley, Greg Rappa, Heinke Höck, Alison Pamment, Martin Juckes

Version 1.6, 5 December, 2011

Table of Contents

List of Tables

List of Examples

H.17. Time series of atmospheric sounding profiles from a set of locations stored in an orthogonal multidimensional array representation.
H.18. Time series of atmospheric sounding profiles from a single location stored in a multidimensional array representation.
H.19. Time series of atmospheric sounding profiles from a set of locations stored in a ragged array representation.
H.20. Time series of atmospheric sounding profiles along a set of trajectories stored in a multidimensional array representation.
H.21. Time series of atmospheric sounding profiles along a trajectory stored in a multidimensional array representation.
H.22. Time series of atmospheric sounding profiles along a set of trajectories stored in a ragged array representation.

About the authors

Original Authors

- Brian Eaton, NCAR
- Jonathan Gregory, Hadley Centre, UK Met Office
- Bob Drach, PCMDI, LLNL
- Karl Taylor, PCMDI, LLNL
- Steve Hankin, PMEL, NOAA

Additional Authors

- John Caron, UCAR
- Rich Signell, USGS
- Phil Bentley, Hadley Centre, UK Met Office
- Greg Rappa, MIT
- Heinke Höck, DKRZ
- Alison Pamment, BADC
- Martin Juckes, BADC

Many others have contributed to the development of CF through their participation in discussions about proposed changes.

Abstract

This document describes the CF conventions for climate and forecast metadata designed to promote the processing and sharing of files created with the netCDF Application Programmer Interface [NetCDF]. The conventions define metadata that provide a definitive description of what the data in each variable represents, and of 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.

The CF conventions generalize and extend the COARDS conventions [COARDS]. The extensions include metadata that provides a precise definition of each variable via specification of a standard name, describes the vertical locations corresponding to dimensionless vertical coordinate values, and provides the spatial coordinates of non-rectilinear gridded data. Since climate and forecast data are often not simply representative of points in space/time, other extensions provide for the description of coordinate intervals, multidimensional cells and climatological time coordinates, and indicate how a data value is representative of an interval or cell. This standard also relaxes the COARDS constraints on dimension order and specifies methods for reducing the size of datasets.

Preface

Home page:

Contains links to: previous draft and current working draft documents; applications for processing CF conforming files; email list for discussion about interpretation, clarification, and proposals for changes or extensions to the current conventions. http://cfconventions.org/

Revision history:

This document will be updated to reflect agreed changes to the standard and to correct mistakes according to the rules of CF governance. See Appendix G, Revision History for the full revision history.

Chapter 1. Introduction

1.1. Goals

The NetCDF library [NetCDF] is designed to read and write data that has been structured according to well-defined rules and is easily ported across various computer platforms. The netCDF interface enables but does not require the creation of *self-describing* datasets. The purpose of the CF conventions is to require conforming datasets to contain sufficient metadata that they are self-describing in the sense that each variable in the file has an associated description of what it represents, including physical units if appropriate, and that each value can be located in space (relative to earth-based coordinates) and time.

An important benefit of a convention is that it enables software tools to display data and perform operations on specified subsets of the data with minimal user intervention. It is possible to provide the metadata describing how a field is located in time and space in many different ways that a human would immediately recognize as equivalent. The purpose in restricting how the metadata is represented is to make it practical to write software that allows a machine to parse that metadata and to automatically associate each data value with its location in time and space. It is equally important that the metadata be easy for human users to write and to understand.

This standard is intended for use with climate and forecast data, for atmosphere, surface and ocean, and was designed with model-generated data particularly in mind. We recognise that there are limits to what a standard can practically cover; we restrict ourselves to issues that we believe to be of common and frequent concern in the design of climate and forecast metadata. Our main purpose therefore, is to propose a clear, adequate and flexible definition of the metadata needed for climate and forecast data. Although this is specifically a netCDF standard, we feel that most of the ideas are of wider application. The metadata objects could be contained in file formats other than netCDF. Conversion of the metadata between files of different formats will be facilitated if conventions for all formats are based on similar ideas.

This convention is designed to be backward compatible with the COARDS conventions [COARDS] , by which we mean that a conforming COARDS dataset also conforms to the CF standard. Thus new applications that implement the CF conventions will be able to process COARDS datasets.

We have also striven to maximize conformance to the COARDS standard, that is, wherever the COARDS metadata conventions provide an adequate description we require their use. Extensions to COARDS are implemented in a manner such that the content that doesn't depend on the extensions is still accessible to applications that adhere to the COARDS standard.

1.2. Terminology

The terms in this document that refer to components of a netCDF file are defined in the NetCDF User's Guide (NUG) [NUG] NUG. Some of those definitions are repeated below for convenience.

auxiliary coordinate variable

Any netCDF variable that contains coordinate data, but is not a coordinate variable (in the sense of that term defined by the NUG and used by this standard - see below). Unlike coordinate

variables, there is no relationship between the name of an auxiliary coordinate variable and the name(s) of its dimension(s).

boundary variable

A boundary variable is associated with a variable that contains coordinate data. When a data value provides information about conditions in a cell occupying a region of space/time or some other dimension, the boundary variable provides a description of cell extent.

CDL syntax

The ascii format used to describe the contents of a netCDF file is called CDL (network Common Data form Language). This format represents arrays using the indexing conventions of the C programming language, i.e., index values start at 0, and in multidimensional arrays, when indexing over the elements of the array, it is the last declared dimension that is the fastest varying in terms of file storage order. The netCDF utilities ncdump and ncgen use this format (see chapter 5 of the NUG). All examples in this document use CDL syntax.

cell

A region in one or more dimensions whose boundary can be described by a set of vertices. The term *interval* is sometimes used for one-dimensional cells.

coordinate variable

We use this term precisely as it is defined in section 2.3.1 of the NUG . It is a one-dimensional variable with the same name as its dimension [e.g., `time(time)`], and it is defined as a numeric data type with values that are ordered monotonically. Missing values are not allowed in coordinate variables.

grid mapping variable

A variable used as a container for attributes that define a specific grid mapping. The type of the variable is arbitrary since it contains no data.

latitude dimension

A dimension of a netCDF variable that has an associated latitude coordinate variable.

longitude dimension

A dimension of a netCDF variable that has an associated longitude coordinate variable.

multidimensional coordinate variable

An auxiliary coordinate variable that is multidimensional.

recommendation

Recommendations in this convention are meant to provide advice that may be helpful for reducing common mistakes. In some cases we have recommended rather than required particular attributes in order to maintain backwards compatibility with COARDS. An application must not depend on a dataset's adherence to recommendations.

scalar coordinate variable

A scalar variable that contains coordinate data. Functionally equivalent to either a size one coordinate variable or a size one auxiliary coordinate variable.

spatiotemporal dimension

A dimension of a netCDF variable that is used to identify a location in time and/or space.

time dimension

A dimension of a netCDF variable that has an associated time coordinate variable.

vertical dimension

A dimension of a netCDF variable that has an associated vertical coordinate variable.

1.3. Overview

No variable or dimension names are standardized by this convention. Instead we follow the lead of the NUG and standardize only the names of attributes and some of the values taken by those attributes. The overview provided in this section will be followed with more complete descriptions in following sections. Appendix A, Attributes contains a summary of all the attributes used in this convention.

We recommend that the NUG defined attribute `Conventions` be given the string value "`CF-1.6`" to identify datasets that conform to these conventions.

The general description of a file's contents should be contained in the following attributes: `title` , `history` , `institution` , `source` , `comment` and `references` (Section 2.6.2, "Description of file contents"). For backwards compatibility with COARDS none of these attributes is required, but their use is recommended to provide human readable documentation of the file contents.

Each variable in a netCDF file has an associated description which is provided by the attributes `units` , `long_name` , and `standard_name` . The `units` , and `long_name` attributes are defined in the NUG and the `standard_name` attribute is defined in this document.

The `units` attribute is required for all variables that represent dimensional quantities (except for boundary variables defined in Section 7.1, "Cell Boundaries" . The values of the `units` attributes are character strings that are recognized by UNIDATA's Udunits package [UDUNITS] , (with exceptions allowed as discussed in Section 3.1, "Units").

The `long_name` and `standard_name` attributes are used to describe the content of each variable. For backwards compatibility with COARDS neither is required, but use of at least one of them is strongly recommended. The use of standard names will facilitate the exchange of climate and forecast data by providing unambiguous identification of variables most commonly analyzed.

Four types of coordinates receive special treatment by these conventions: latitude, longitude, vertical, and time. Every variable must have associated metadata that allows identification of each such coordinate that is relevant. Two independent parts of the convention allow this to be done. There are conventions that identify the variables that contain the coordinate data, and there are conventions that identify the type of coordinate represented by that data.

There are two methods used to identify variables that contain coordinate data. The first is to use the NUG-defined "coordinate variables." *The use of coordinate variables is required for all dimensions that correspond to one dimensional space or time coordinates* . In cases where coordinate variables are not applicable, the variables containing coordinate data are identified by the `coordinates`

attribute.

Once the variables containing coordinate data are identified, further conventions are required to determine the type of coordinate represented by each of these variables. Latitude, longitude, and time coordinates are identified solely by the value of their `units` attribute. Vertical coordinates with units of pressure may also be identified by the `units` attribute. Other vertical coordinates must use the attribute `positive` which determines whether the direction of increasing coordinate value is up or down. Because identification of a coordinate type by its units involves the use of an external software package [UDUNITS] , we provide the optional attribute `axis` for a direct identification of coordinates that correspond to latitude, longitude, vertical, or time axes.

Latitude, longitude, and time are defined by internationally recognized standards, and hence, identifying the coordinates of these types is sufficient to locate data values uniquely with respect to time and a point on the earth's surface. On the other hand identifying the vertical coordinate is not necessarily sufficient to locate a data value vertically with respect to the earth's surface. In particular a model may output data on the dimensionless vertical coordinate used in its mathematical formulation. To achieve the goal of being able to spatially locate all data values, this convention includes the definitions of common dimensionless vertical coordinates in Appendix D, Dimensionless Vertical Coordinates . These definitions provide a mapping between the dimensionless coordinate values and dimensional values that can be uniquely located with respect to a point on the earth's surface. The definitions are associated with a coordinate variable via the `standard_name` and `formula_terms` attributes. For backwards compatibility with COARDS use of these attributes is not required, but is strongly recommended.

It is often the case that data values are not representative of single points in time and/or space, but rather of intervals or multidimensional cells. This convention defines a `bounds` attribute to specify the extent of intervals or cells. When data that is representative of cells can be described by simple statistical methods, those methods can be indicated using the `cell_methods` attribute. An important application of this attribute is to describe climatological and diurnal statistics.

Methods for reducing the total volume of data include both packing and compression. Packing reduces the data volume by reducing the precision of the stored numbers. It is implemented using the attributes `add_offset` and `scale_factor` which are defined in the NUG. Compression on the other hand loses no precision, but reduces the volume by not storing missing data. The attribute `compress` is defined for this purpose.

1.4. Relationship to the COARDS Conventions

These conventions generalize and extend the COARDS conventions [COARDS] . A major design goal has been to maintain *backward compatibility* with COARDS. Hence applications written to process datasets that conform to these conventions will also be able to process COARDS conforming datasets. We have also striven to maximize *conformance* to the COARDS standard so that datasets that only require the metadata that was available under COARDS will still be able to be processed by COARDS conforming applications. But because of the extensions that provide new metadata content, and the relaxation of some COARDS requirements, datasets that conform to these conventions will not necessarily be recognized by applications that adhere to the COARDS conventions. The features of these conventions that allow writing netCDF files that are not COARDS conforming are summarized below.

COARDS standardizes the description of grids composed of independent latitude, longitude, vertical, and time axes. In addition to standardizing the metadata required to identify each of these axis types COARDS restricts the axis (equivalently dimension) ordering to be longitude, latitude, vertical, and time (with longitude being the most rapidly varying dimension). Because of I/O performance considerations it may not be possible for models to output their data in conformance with the COARDS requirement. The CF convention places no rigid restrictions on the order of dimensions, however we encourage data producers to make the extra effort to stay within the COARDS standard order. The use of non-COARDS axis ordering will render files inaccessible to some applications and limit interoperability. Often a buffering operation can be used to miminize performance penalties when axis ordering in model code does not match the axis ordering of a COARDS file.

COARDS addresses the issue of identifying dimensionless vertical coordinates, but does not provide any mechanism for mapping the dimensionless values to dimensional ones that can be located with respect to the earth's surface. For backwards compatibility we continue to allow (but do not require) the `units` attribute of dimensionless vertical coordinates to take the values "level", "layer", or "sigma_level." But we recommend that the `standard_name` and `formula_terms` attributes be used to identify the appropriate definition of the dimensionless vertical coordinate (see Section 4.3.2, "Dimensionless Vertical Coordinate").

The CF conventions define attributes which enable the description of data properties that are outside the scope of the COARDS conventions. These new attributes do not violate the COARDS conventions, but applications that only recognize COARDS conforming datasets will not have the capabilities that the new attributes are meant to enable. Briefly the new attributes allow:

- Identification of quantities using standard names.
- Description of dimensionless vertical coordinates.
- Associating dimensions with auxiliary coordinate variables.
- Linking data variables to scalar coordinate variables.
- Associating dimensions with labels.
- Description of intervals and cells.
- Description of properties of data defined on intervals and cells.
- Description of climatological statistics.
- Data compression for variables with missing values.

Chapter 2. NetCDF Files and Components

The components of a netCDF file are described in section 2 of the NUG [NUG] . In this section we describe conventions associated with filenames and the basic components of a netCDF file. We also introduce new attributes for describing the contents of a file.

2.1. Filename

NetCDF files should have the file name extension "`.nc`".

2.2. Data Types

The netCDF data types `char`, `byte`, `short`, `int`, `float` or `real`, and `double` are all acceptable. The `char` type is not intended for numeric data. One byte numeric data should be stored using the `byte` data type. All integer types are treated by the netCDF interface as signed. It is possible to treat the `byte` type as unsigned by using the NUG convention of indicating the unsigned range using the `valid_min`, `valid_max`, or `valid_range` attributes.

NetCDF does not support a character string type, so these must be represented as character arrays. In this document, a one dimensional array of character data is simply referred to as a "string". An n-dimensional array of strings must be implemented as a character array of dimension (n,max_string_length), with the last (most rapidly varying) dimension declared large enough to contain the longest string in the array. All the strings in a given array are therefore defined to be equal in length. For example, an array of strings containing the names of the months would be dimensioned (12,9) in order to accommodate "September", the month with the longest name.

2.3. Naming Conventions

Variable, dimension and attribute names should begin with a letter and be composed of letters, digits, and underscores. Note that this is in conformance with the COARDS conventions, but is more restrictive than the netCDF interface which allows use of the hyphen character. The netCDF interface also allows leading underscores in names, but the NUG states that this is reserved for system use.

Case is significant in netCDF names, but it is recommended that names should not be distinguished purely by case, i.e., if case is disregarded, no two names should be the same. It is also recommended that names should be obviously meaningful, if possible, as this renders the file more effectively self-describing.

This convention does not standardize any variable or dimension names. Attribute names and their contents, where standardized, are given in English in this document and should appear in English in conforming netCDF files for the sake of portability. Languages other than English are permitted for variables, dimensions, and non-standardized attributes. The content of some standardized attributes are string values that are not standardized, and thus are not required to be in English. For example, a description of what a variable represents may be given in a non-English language using the `long_name` attribute (see Section 3.2, "Long Name") whose contents are not standardized, but a description given by the `standard_name` attribute (see Section 3.3, "Standard Name") must be

taken from the standard name table which is in English.

2.4. Dimensions

A variable may have any number of dimensions, including zero, and the dimensions must all have different names. *COARDS strongly recommends limiting the number of dimensions to four, but we wish to allow greater flexibility* . The dimensions of the variable define the axes of the quantity it contains. Dimensions other than those of space and time may be included. Several examples can be found in this document. Under certain circumstances, one may need more than one dimension in a particular quantity. For instance, a variable containing a two-dimensional probability density function might correlate the temperature at two different vertical levels, and hence would have temperature on both axes.

If any or all of the dimensions of a variable have the interpretations of "date or time" (`T`), "height or depth" (`Z`), "latitude" (`Y`), or "longitude" (`X`) then we recommend, but do not require (see Section 1.4, "Relationship to the COARDS Conventions"), those dimensions to appear in the relative order `T`, then `Z`, then `Y`, then `X` in the CDL definition corresponding to the file. All other dimensions should, whenever possible, be placed to the left of the spatiotemporal dimensions.

Dimensions may be of any size, including unity. When a single value of some coordinate applies to all the values in a variable, the recommended means of attaching this information to the variable is by use of a dimension of size unity with a one-element coordinate variable. It is also acceptable to use a scalar coordinate variable which eliminates the need for an associated size one dimension in the data variable. The advantage of using either a coordinate variable or an auxiliary coordinate variable is that all its attributes can be used to describe the single-valued quantity, including boundaries. For example, a variable containing data for temperature at 1.5 m above the ground has a single-valued coordinate supplying a height of 1.5 m, and a time-mean quantity has a single-valued time coordinate with an associated boundary variable to record the start and end of the averaging period.

2.5. Variables

This convention does not standardize variable names.

NetCDF variables that contain coordinate data are referred to as *coordinate variables*, *auxiliary coordinate variables*, *scalar coordinate variables*, or *multidimensional coordinate variables*.

2.5.1. Missing Data

The NUG conventions (NUG appendix B) provide the `_FillValue`, `missing_value`, `valid_min`, `valid_max`, and `valid_range` attributes to indicate missing data.

The NUG conventions for missing data changed significantly between version 2.3 and version 2.4. Since version 2.4 the NUG defines missing data as all values outside of the `valid_range`, and specifies how the `valid_range` should be defined from the `_FillValue` (which has library specified default values) if it hasn't been explicitly specified. If only one missing value is needed for a variable then we recommend that this value be specified using the `_FillValue` attribute. Doing this guarantees that the missing value will be recognized by generic applications that follow either the before or

after version 2.4 conventions.

The scalar attribute with the name `_FillValue` and of the same type as its variable is recognized by the netCDF library as the value used to pre-fill disk space allocated to the variable. This value is considered to be a special value that indicates undefined or missing data, and is returned when reading values that were not written. The `_FillValue` should be outside the range specified by `valid_range` (if used) for a variable. The netCDF library defines a default fill value for each data type (NUG appendix C).

The missing values of a variable with `scale_factor` and/or `add_offset` attributes (see section Section 8.1, "Packed Data") are interpreted relative to the variable's external values (a.k.a. the packed values, the raw values, the values stored in the netCDF file), not the values that result after the scale and offset are applied. Applications that process variables that have attributes to indicate both a transformation (via a scale and/or offset) and missing values should first check that a data value is valid, and then apply the transformation. Note that values that are identified as missing should not be transformed. Since the missing value is outside the valid range it is possible that applying a transformation to it could result in an invalid operation. For example, the default `_FillValue` is very close to the maximum representable value of IEEE single precision floats, and multiplying it by 100 produces an "Infinity" (using single precision arithmetic).

2.6. Attributes

This standard describes many attributes (some mandatory, others optional), but a file may also contain non-standard attributes. Such attributes do not represent a violation of this standard. Application programs should ignore attributes that they do not recognise or which are irrelevant for their purposes. Conventional attribute names should be used wherever applicable. Non-standard names should be as meaningful as possible. Before introducing an attribute, consideration should be given to whether the information would be better represented as a variable. In general, if a proposed attribute requires ancillary data to describe it, is multidimensional, requires any of the defined netCDF dimensions to index its values, or requires a significant amount of storage, a variable should be used instead. When this standard defines string attributes that may take various prescribed values, the possible values are generally given in lower case. However, applications programs should not be sensitive to case in these attributes. Several string attributes are defined by this standard to contain "blank-separated lists". Consecutive words in such a list are separated by one or more adjacent spaces. The list may begin and end with any number of spaces. See Appendix A, Attributes for a list of attributes described by this standard.

2.6.1. Identification of Conventions

We recommend that netCDF files that follow these conventions indicate this by setting the NUG defined global attribute `Conventions` to the string value "`CF-1.6`". The string is interpreted as a directory name relative to a directory that is a repository of documents describing sets of discipline-specific conventions. The conventions directory name is currently interpreted relative to the directory `pub/netcdf/Conventions/` on the host machine `ftp.unidata.ucar.edu`. The web based versions of this document are linked from the netCDF Conventions web page.

2.6.2. Description of file contents

The following attributes are intended to provide information about where the data came from and what has been done to it. This information is mainly for the benefit of human readers. The attribute values are all character strings. For readability in ncdump outputs it is recommended to embed newline characters into long strings to break them into lines. For backwards compatibility with COARDS none of these global attributes is required.

The NUG defines `title` and `history` to be global attributes. We wish to allow the newly defined attributes, i.e., `institution`, `source`, `references`, and `comment`, to be either global or assigned to individual variables. When an attribute appears both globally and as a variable attribute, the variable's version has precedence.

`title`
: A succinct description of what is in the dataset.

`institution`
: Specifies where the original data was produced.

`source`
: The method of production of the original data. If it was model-generated, `source` should name the model and its version, as specifically as could be useful. If it is observational, `source` should characterize it (e.g., "`surface observation`" or "`radiosonde`").

`history`
: Provides an audit trail for modifications to the original data. Well-behaved generic netCDF filters will automatically append their name and the parameters with which they were invoked to the global history attribute of an input netCDF file. We recommend that each line begin with a timestamp indicating the date and time of day that the program was executed.

`references`
: Published or web-based references that describe the data or methods used to produce it.

`comment`
: Miscellaneous information about the data or methods used to produce it.

Chapter 3. Description of the Data

The attributes described in this section are used to provide a description of the content and the units of measurement for each variable. We continue to support the use of the `units` and `long_name` attributes as defined in COARDS. We extend COARDS by adding the optional `standard_name` attribute which is used to provide unique identifiers for variables. This is important for data exchange since one cannot necessarily identify a particular variable based on the name assigned to it by the institution that provided the data.

The `standard_name` attribute can be used to identify variables that contain coordinate data. But since it is an optional attribute, applications that implement these standards must continue to be able to identify coordinate types based on the COARDS conventions.

3.1. Units

The `units` attribute is required for all variables that represent dimensional quantities (except for boundary variables defined in Section 7.1, "Cell Boundaries" and climatology variables defined in Section 7.4, "Climatological Statistics"). The value of the `units` attribute is a string that can be recognized by UNIDATA's Udunits package [UDUNITS], with a few exceptions that are given below. The Udunits package includes a file `udunits.dat`, which lists its supported unit names. Note that case is significant in the `units` strings.

The COARDS convention prohibits the unit `degrees` altogether, but this unit is not forbidden by the CF convention because it may in fact be appropriate for a variable containing, say, solar zenith angle. The unit `degrees` is also allowed on coordinate variables such as the latitude and longitude coordinates of a transformed grid. In this case the coordinate values are not true latitudes and longitudes which must always be identified using the more specific forms of `degrees` as described in Section 4.1, "Latitude Coordinate" and Section 4.2, "Longitude Coordinate".

Units are not required for dimensionless quantities. A variable with no units attribute is assumed to be dimensionless. However, a units attribute specifying a dimensionless unit may optionally be included. The Udunits package defines a few dimensionless units, such as `percent`, but is lacking commonly used units such as ppm (parts per million). This convention does not support the addition of new dimensionless units that are not udunits compatible. The conforming unit for quantities that represent fractions, or parts of a whole, is "1". The conforming unit for parts per million is "1e-6". Descriptive information about dimensionless quantities, such as sea-ice concentration, cloud fraction, probability, etc., should be given in the `long_name` or `standard_name` attributes (see below) rather than the `units` .

The units `level`, `layer`, and `sigma_level` are allowed for dimensionless vertical coordinates to maintain backwards compatibility with COARDS. These units are not compatible with Udunits and are deprecated by this standard because conventions for more precisely identifying dimensionless vertical coordinates are introduced (see Section 4.3.2, "Dimensionless Vertical Coordinate").

The Udunits syntax that allows scale factors and offsets to be applied to a unit is not supported by this standard. The application of any scale factors or offsets to data should be indicated by the `scale_factor` and `add_offset` attributes. Use of these attributes for data packing, which is their most important application, is discussed in detail in Section 8.1, "Packed Data".

Udunits recognizes the following prefixes and their abbreviations.

Table 3.1. Supported Units

Factor	Prefix	Abbreviation		Factor	Prefix	Abbreviation
1e1	deca,deka	da		1e-1	deci	d
1e2	hecto	h		1e-2	centi	c
1e3	kilo	k		1e-3	milli	m
1e6	mega	M		1e-6	micro	u
1e9	giga	G		1e-9	nano	n
1e12	tera	T		1e-12	pico	p
1e15	peta	P		1e-15	femto	f
1e18	exa	E		1e-18	atto	a
1e21	zetta	Z		1e-21	zepto	z
1e24	yotta	Y		1e-24	yocto	y

3.2. Long Name

The `long_name` attribute is defined by the NUG to contain a long descriptive name which may, for example, be used for labeling plots. For backwards compatibility with COARDS this attribute is optional. But it is highly recommended that either this or the `standard_name` attribute defined in the next section be provided to make the file self-describing. If a variable has no `long_name` attribute then an application may use, as a default, the `standard_name` if it exists, or the variable name itself.

3.3. Standard Name

A fundamental requirement for exchange of scientific data is the ability to describe precisely the physical quantities being represented. To some extent this is the role of the `long_name` attribute as defined in the NUG. However, usage of `long_name` is completely ad-hoc. For some applications it would be desirable to have a more definitive description of the quantity, which would allow users of data from different sources to determine whether quantities were in fact comparable. For this reason an optional mechanism for uniquely associating each variable with a standard name is provided.

A standard name is associated with a variable via the attribute `standard_name` which takes a string value comprised of a standard name optionally followed by one or more blanks and a standard name modifier (a string value from Appendix C, Standard Name Modifiers).

The set of permissible standard names is contained in the standard name table. The table entry for each standard name contains the following:

standard name

The name used to identify the physical quantity. A standard name contains no whitespace and is case sensitive.

canonical units

Representative units of the physical quantity. Unless it is dimensionless, a variable with a `standard_name` attribute must have units which are physically equivalent (not necessarily identical) to the canonical units, possibly modified by an operation specified by either the standard name modifier (see below and Appendix C, Standard Name Modifiers) or by the `cell_methods` attribute (see Section 7.3, "Cell Methods" and Appendix E, Cell Methods).

description

The description is meant to clarify the qualifiers of the fundamental quantities such as which surface a quantity is defined on or what the flux sign conventions are. We don"t attempt to provide precise definitions of fundumental physical quantities (e.g., temperature) which may be found in the literature.

When appropriate, the table entry also contains the corresponding GRIB parameter code(s) (from ECMWF and NCEP) and AMIP identifiers.

The standard name table is located at http://cfconventions.org/Data/cf-standard-names/current/src/cf-standard-name-table.xml, written in compliance with the XML format, as described in Appendix B, Standard Name Table Format. Knowledge of the XML format is only necessary for application writers who plan to directly access the table. A formatted text version of the table is provided at http://cfconventions.org/Data/cf-standard-names/current/build/cf-standard-name-table.html, and this table may be consulted in order to find the standard name that should be assigned to a variable. Some standard names (e.g. `region` and `area_type`) are used to indicate quantities which are permitted to take only certain standard values. This is indicated in the definition of the quantity in the standard name table, accompanied by a list or a link to a list of the permitted values.

Standard names by themselves are not always sufficient to describe a quantity. For example, a variable may contain data to which spatial or temporal operations have been applied. Or the data may represent an uncertainty in the measurement of a quantity. These quantity attributes are expressed as modifiers of the standard name. Modifications due to common statistical operations are expressed via the `cell_methods` attribute (see Section 7.3, "Cell Methods" and Appendix E, Cell Methods). Other types of quantity modifiers are expressed using the optional modifier part of the `standard_name` attribute. The permissible values of these modifiers are given in Appendix C, Standard Name Modifiers.

Example 3.1. Use of `standard_name`

```
float psl(lat,lon) ;
  psl:long_name = "mean sea level pressure" ;
  psl:units = "hPa" ;
  psl:standard_name = "air_pressure_at_sea_level" ;
```

The description in the standard name table entry for `air_pressure_at_sea_level` clarifies that "sea level" refers to the mean sea level, which is close to the geoid in sea areas.

Here are lists of equivalences between the CF standard names and the standard names from the ECMWF GRIB tables, the NCEP GRIB tables, and the PCMDI tables.

3.4. Ancillary Data

When one data variable provides metadata about the individual values of another data variable it may be desirable to express this association by providing a link between the variables. For example, instrument data may have associated measures of uncertainty. The attribute `ancillary_variables` is used to express these types of relationships. It is a string attribute whose value is a blank separated list of variable names. The nature of the relationship between variables associated via `ancillary_variables` must be determined by other attributes. The variables listed by the `ancillary_variables` attribute will often have the standard name of the variable which points to them including a modifier (Appendix C, Standard Name Modifiers) to indicate the relationship.

Example 3.2. Instrument data

```
float q(time) ;
  q:standard_name = "specific_humidity" ;
  q:units = "g/g" ;
  q:ancillary_variables = "q_error_limit q_detection_limit" ;
float q_error_limit(time)
  q_error_limit:standard_name = "specific_humidity standard_error" ;
  q_error_limit:units = "g/g" ;
float q_detection_limit(time)
  q_detection_limit:standard_name = "specific_humidity detection_minimum" ;
  q_detection_limit:units = "g/g" ;
```

3.5. Flags

The attributes `flag_values`, `flag_masks` and `flag_meanings` are intended to make variables that contain flag values self describing. Status codes and Boolean (binary) condition flags may be expressed with different combinations of `flag_values` and `flag_masks` attribute definitions.

The `flag_values` and `flag_meanings` attributes describe a status flag consisting of mutually exclusive coded values. The `flag_values` attribute is the same type as the variable to which it is attached, and

contains a list of the possible flag values. The `flag_meanings` attribute is a string whose value is a blank separated list of descriptive words or phrases, one for each flag value. Each word or phrase should consist of characters from the alphanumeric set and the following five: '_', '-', '.', '+', '@'. If multi-word phrases are used to describe the flag values, then the words within a phrase should be connected with underscores. The following example illustrates the use of flag values to express a speed quality with an enumerated status code.

Example 3.3. A flag variable, using `flag_values`

```
byte current_speed_qc(time, depth, lat, lon) ;
  current_speed_qc:long_name = "Current Speed Quality" ;
  current_speed_qc:standard_name = "sea_water_speed status_flag" ;
  current_speed_qc:_FillValue = -128b ;
  current_speed_qc:valid_range = 0b, 2b ;
  current_speed_qc:flag_values = 0b, 1b, 2b ;
  current_speed_qc:flag_meanings = "quality_good sensor_nonfunctional
                                    outside_valid_range" ;
```

The `flag_masks` and `flag_meanings` attributes describe a number of independent Boolean conditions using bit field notation by setting unique bits in each `flag_masks` value. The `flag_masks` attribute is the same type as the variable to which it is attached, and contains a list of values matching unique bit fields. The `flag_meanings` attribute is defined as above, one for each `flag_masks` value. A flagged condition is identified by performing a bitwise AND of the variable value and each `flag_masks` value; a non-zero result indicates a `true` condition. Thus, any or all of the flagged conditions may be `true`, depending on the variable bit settings. The following example illustrates the use of `flag_masks` to express six sensor status conditions.

Example 3.4. A flag variable, using `flag_masks`

```
byte sensor_status_qc(time, depth, lat, lon) ;
  sensor_status_qc:long_name = "Sensor Status" ;
  sensor_status_qc:_FillValue = 0b ;
  sensor_status_qc:valid_range = 1b, 63b ;
  sensor_status_qc:flag_masks = 1b, 2b, 4b, 8b, 16b, 32b ;
  sensor_status_qc:flag_meanings = "low_battery processor_fault
                                    memory_fault disk_fault
                                    software_fault
                                    maintenance_required" ;
```

The `flag_masks`, `flag_values` and `flag_meanings` attributes, used together, describe a blend of independent Boolean conditions and enumerated status codes. The `flag_masks` and `flag_values` attributes are both the same type as the variable to which they are attached. A flagged condition is identified by a bitwise AND of the variable value and each `flag_masks` value; a result that matches the `flag_values` value indicates a `true` condition. Repeated `flag_masks` define a bit field mask that identifies a number of status conditions with different `flag_values`. The `flag_meanings` attribute is

defined as above, one for each `flag_masks` bit field and `flag_values` definition. Each `flag_values` and `flag_masks` value must coincide with a `flag_meanings` value. The following example illustrates the use of `flag_masks` and `flag_values` to express two sensor status conditions and one enumerated status code.

Example 3.5. A flag variable, using `flag_masks` and `flag_values`

```
byte sensor_status_qc(time, depth, lat, lon) ;
  sensor_status_qc:long_name = "Sensor Status" ;
  sensor_status_qc:_FillValue = 0b ;
  sensor_status_qc:valid_range = 1b, 15b ;
  sensor_status_qc:flag_masks = 1b, 2b, 12b, 12b, 12b ;
  sensor_status_qc:flag_values = 1b, 2b, 4b, 8b, 12b ;
  sensor_status_qc:flag_meanings =
       "low_battery
        hardware_fault
        offline_mode calibration_mode maintenance_mode" ;
```

In this case, mutually exclusive values are blended with Boolean values to maximize use of the available bits in a flag value. The table below represents the four binary digits (bits) expressed by the `sensor_status_qc` variable in the previous example.

Bit 0 and Bit 1 are Boolean values indicating a low battery condition and a hardware fault, respectively. The next two bits (Bit 2 and Bit 3) express an enumeration indicating abnormal sensor operating modes. Thus, if Bit 0 is set, the battery is low and if Bit 1 is set, there is a hardware fault - independent of the current sensor operating mode.

Table 3.2. Flag Variable Bits (from Example)

Bit 3 (MSB)	**Bit 2**	**Bit 1**	**Bit 0 (LSB)**
		H/W Fault	Low Batt

The remaining bits (Bit 2 and Bit 3) are decoded as follows:

Table 3.3. Flag Variable Bit 2 and Bit 3 (from Example)

Bit 3	**Bit 2**	**Mode**
0	1	offline_mode
1	0	calibration_mode
1	1	maintenance_mode

The "12b" flag mask is repeated in the `sensor_status_qc flag_masks` definition to explicitly declare the recommended bit field masks to repeatedly AND with the variable value while searching for matching enumerated values. An application determines if any of the conditions declared in the `flag_meanings` list are `true` by simply iterating through each of the `flag_masks` and AND'ing them with the variable. When a result is equal to the corresponding `flag_values` element, that condition

is `true`. The repeated `flag_masks` enable a simple mechanism for clients to detect all possible conditions.

Chapter 4. Coordinate Types

Four types of coordinates receive special treatment by these conventions: latitude, longitude, vertical, and time. We continue to support the special role that the `units` and `positive` attributes play in the COARDS convention to identify coordinate type. We extend COARDS by providing explicit definitions of dimensionless vertical coordinates. The definitions are associated with a coordinate variable via the `standard_name` and `formula_terms` attributes. For backwards compatibility with COARDS use of these attributes is not required, but is strongly recommended.

Because identification of a coordinate type by its units is complicated by requiring the use of an external software package [UDUNITS] , we provide two optional methods that yield a direct identification. The attribute `axis` may be attached to a coordinate variable and given one of the values `X`, `Y`, `Z` or `T` which stand for a longitude, latitude, vertical, or time axis respectively. Alternatively the `standard_name` attribute may be used for direct identification. But note that these optional attributes are in addition to the required COARDS metadata.

Coordinate types other than latitude, longitude, vertical, and time are allowed. To identify generic spatial coordinates we recommend that the `axis` attribute be attached to these coordinates and given one of the values `X`, `Y` or `Z`. The values `X` and `Y` for the axis attribute should be used to identify horizontal coordinate variables. If both X- and Y-axis are identified, `X-Y-up` should define a right-handed coordinate system, i.e. rotation from the positive X direction to the positive Y direction is anticlockwise if viewed from above. We strongly recommend that coordinate variables be used for all coordinate types whenever they are applicable.

The methods of identifying coordinate types described in this section apply both to coordinate variables and to auxiliary coordinate variables named by the `coordinates` attribute (see Chapter 5, Coordinate Systems).

The values of a coordinate variable or auxiliary coordinate variable indicate the locations of the gridpoints. The locations of the boundaries between cells are indicated by bounds variables (see Section 7.1, "Cell Boundaries"). If bounds are not provided, an application might reasonably assume the gridpoints to be at the centers of the cells, but we do not require that in this standard.

4.1. Latitude Coordinate

Variables representing latitude must always explicitly include the `units` attribute; there is no default value. The `units` attribute will be a string formatted as per the `udunits.dat` file. The recommended unit of latitude is `degrees_north`. Also acceptable are `degree_north`, `degree_N`, `degrees_N`, `degreeN`, and `degreesN`.

Example 4.1. Latitude axis

```
float lat(lat) ;
  lat:long_name = "latitude" ;
  lat:units = "degrees_north" ;
  lat:standard_name = "latitude" ;
```

Application writers should note that the Udunits package does not recognize the directionality implied by the "north" part of the unit specification. It only recognizes its size, i.e., 1 degree is defined to be pi/180 radians. Hence, determination that a coordinate is a latitude type should be done via a string match between the given unit and one of the acceptable forms of `degrees_north`.

Optionally, the latitude type may be indicated additionally by providing the `standard_name` attribute with the value `latitude`, and/or the `axis` attribute with the value `Y`.

Coordinates of latitude with respect to a rotated pole should be given units of `degrees`, not `degrees_north` or equivalents, because applications which use the units to identify axes would have no means of distinguishing such an axis from real latitude, and might draw incorrect coastlines, for instance.

4.2. Longitude Coordinate

Variables representing longitude must always explicitly include the `units` attribute; there is no default value. The units `attribute` will be a string formatted as per the `udunits.dat` file. The recommended unit of longitude is `degrees_east`. Also acceptable are `degree_east`, `degree_E`, `degrees_E`, `degreeE`, and `degreesE`.

Example 4.2. Longitude axis

```
float lon(lon) ;
  lon:long_name = "longitude" ;
  lon:units = "degrees_east" ;
  lon:standard_name = "longitude" ;
```

Application writers should note that the Udunits package has limited recognition of the directionality implied by the "east" part of the unit specification. It defines `degrees_east` to be pi/180 radians, and hence equivalent to `degrees_north`. We recommend the determination that a coordinate is a longitude type should be done via a string match between the given unit and one of the acceptable forms of `degrees_east`.

Optionally, the longitude type may be indicated additionally by providing the `standard_name` attribute with the value `longitude`, and/or the `axis` attribute with the value `X`.

Coordinates of longitude with respect to a rotated pole should be given units of `degrees`, not `degrees_east` or equivalents, because applications which use the units to identify axes would have no means of distinguishing such an axis from real longitude, and might draw incorrect coastlines, for instance.

4.3. Vertical (Height or Depth) Coordinate

Variables representing dimensional height or depth axes must always explicitly include the `units` attribute; there is no default value.

The direction of positive (i.e., the direction in which the coordinate values are increasing), whether

up or down, cannot in all cases be inferred from the units. The direction of positive is useful for applications displaying the data. For this reason the attribute `positive` as defined in the COARDS standard is required if the vertical axis units are not a valid unit of pressure (a determination which can be made using the udunits routine, utScan) — otherwise its inclusion is optional. The `positive` attribute may have the value `up` or `down` (case insensitive). This attribute may be applied to either coordinate variables or auxillary coordinate variables that contain vertical coordinate data.

For example, if an oceanographic netCDF file encodes the depth of the surface as 0 and the depth of 1000 meters as 1000 then the axis would use attributes as follows:

```
axis_name:units = "meters" ;
axis_name:positive = "down" ;
```

If, on the other hand, the depth of 1000 meters were represented as -1000 then the value of the `positive` attribute would have been `up`. If the `units` attribute value is a valid pressure unit the default value of the `positive` attribute is `down`.

A vertical coordinate will be identifiable by:

- units of pressure; or
- the presence of the positive attribute with a value of `up` or `down` (case insensitive).

Optionally, the vertical type may be indicated additionally by providing the `standard_name` attribute with an appropriate value, and/or the `axis` attribute with the value `Z`.

4.3.1. Dimensional Vertical Coordinate

The `units` attribute for dimensional coordinates will be a string formatted as per the `udunits.dat` file. The acceptable units for vertical (depth or height) coordinate variables are:

- units of pressure as listed in the file `udunits.dat`. For vertical axes the most commonly used of these include include `bar`, `millibar`, `decibar`, `atmosphere (atm)`, `pascal (Pa)`, and `hPa`.
- units of length as listed in the file udunits.dat. For vertical axes the most commonly used of these include `meter (metre, m)`, and `kilometer (km)`.
- other units listed in the file udunits.dat that may under certain circumstances reference vertical position such as units of density or temperature.

Plural forms are also acceptable.

4.3.2. Dimensionless Vertical Coordinate

The `units` attribute is not required for dimensionless coordinates. For backwards compatibility with COARDS we continue to allow the `units` attribute to take one of the values: `level`, `layer`, or `sigma_level`. These values are not recognized by the Udunits package, and are considered a deprecated feature in the CF standard.

For dimensionless vertical coordinates we extend the COARDS standard by making use of the `standard_name` attribute to associate a coordinate with its definition from Appendix D,

Dimensionless Vertical Coordinates . The definition provides a mapping between the dimensionless coordinate values and dimensional values that can positively and uniquely indicate the location of the data. A new attribute, `formula_terms`, is used to associate terms in the definitions with variables in a netCDF file. To maintain backwards compatibility with COARDS the use of these attributes is not required, but is strongly recommended.

Example 4.3. Atmosphere sigma coordinate

```
float lev(lev) ;
  lev:long_name = "sigma at layer midpoints" ;
  lev:positive = "down" ;
  lev:standard_name = "atmosphere_sigma_coordinate" ;
  lev:formula_terms = "sigma: lev ps: PS ptop: PTOP" ;
```

In this example the `standard_name` value `atmosphere_sigma_coordinate` identifies the following definition from Appendix D, Dimensionless Vertical Coordinates which specifies how to compute pressure at gridpoint `(n,k,j,i)` where `j` and `i` are horizontal indices, `k` is a vertical index, and `n` is a time index:

```
p(n,k,j,i) = ptop + sigma(k)*(ps(n,j,i)-ptop)
```

The `formula_terms` attribute associates the variable `lev` with the term `sigma`, the variable `PS` with the term `ps`, and the variable `PTOP` with the term `ptop`. Thus the pressure at gridpoint `(n,k,j,i)` would be calculated by

```
p(n,k,j,i) = PTOP + lev(k)*(PS(n,j,i)-PTOP)
```

4.4. Time Coordinate

Variables representing time must always explicitly include the `units` attribute; there is no default value. The `units` attribute takes a string value formatted as per the recommendations in the Udunits package [UDUNITS] . The following excerpt from the Udunits documentation explains the time unit encoding by example:

```
The specification:

    seconds since 1992-10-8 15:15:42.5 -6:00

indicates seconds since October 8th, 1992  at  3  hours,  15
minutes  and  42.5 seconds in the afternoon in the time zone
which is six hours to the west of Coordinated Universal Time
(i.e.  Mountain Daylight Time).  The time zone specification
can also be written without a colon using one or  two-digits
(indicating hours) or three or four digits (indicating hours
and minutes).
```

The acceptable units for time are listed in the `udunits.dat` file. The most commonly used of these strings (and their abbreviations) includes `day (d)`, `hour (hr, h)`, `minute (min)` and `second (sec, s)`. Plural forms are also acceptable. The reference time string (appearing after the identifier `since`) may include date alone; date and time; or date, time, and time zone. The reference time is required. A reference time in year 0 has a special meaning (see Section 7.4, "Climatological Statistics").

Note: if the time zone is omitted the default is UTC, and if both time and time zone are omitted the default is 00:00:00 UTC.

We recommend that the unit `year` be used with caution. The Udunits package defines a `year` to be exactly 365.242198781 days (the interval between 2 successive passages of the sun through vernal equinox). *It is not a calendar year*. Udunits includes the following definitions for years: a `common_year` is 365 days, a `leap_year` is 366 days, a `Julian_year` is 365.25 days, and a `Gregorian_year` is 365.2425 days.

For similar reasons the unit `month`, which is defined in `udunits.dat` to be exactly `year/12`, should also be used with caution.

Example 4.4. Time axis

```
double time(time) ;
  time:long_name = "time" ;
  time:units = "days since 1990-1-1 0:0:0" ;
```

A time coordinate is identifiable from its units string alone. The Udunits routines `utScan()` and `utIsTime()` can be used to make this determination.

Optionally, the time coordinate may be indicated additionally by providing the `standard_name` attribute with an appropriate value, and/or the `axis` attribute with the value `T`.

4.4.1. Calendar

In order to calculate a new date and time given a base date, base time and a time increment one must know what calendar to use. For this purpose we recommend that the calendar be specified by the attribute `calendar` which is assigned to the time coordinate variable. The values currently

defined for `calendar` are:

`gregorian` *or* `standard`
: Mixed Gregorian/Julian calendar as defined by Udunits. *This is the default.*

`proleptic_gregorian`
: A Gregorian calendar extended to dates before 1582-10-15. That is, a year is a leap year if either (i) it is divisible by 4 but not by 100 or (ii) it is divisible by 400.

`noleap` *or* `365_day`
: Gregorian calendar without leap years, i.e., all years are 365 days long.

`all_leap` *or* `366_day`
: Gregorian calendar with every year being a leap year, i.e., all years are 366 days long.

`360_day`
: All years are 360 days divided into 30 day months.

`julian`
: Julian calendar.

`none`
: No calendar.

The `calendar` attribute may be set to `none` in climate experiments that simulate a fixed time of year. The time of year is indicated by the date in the reference time of the `units` attribute. The time coordinate that might apply in a perpetual July experiment are given in the following example.

Example 4.5. Perpetual time axis

```
variables:
  double time(time) ;
    time:long_name = "time" ;
    time:units = "days since 1-7-15 0:0:0" ;
    time:calendar = "none" ;
data:
  time = 0., 1., 2., ...;
```

Here, all days simulate the conditions of 15th July, so it does not make sense to give them different dates. The time coordinates are interpreted as 0, 1, 2, etc. days since the start of the experiment.

If none of the calendars defined above applies (e.g., calendars appropriate to a different paleoclimate era), a non-standard calendar can be defined. The lengths of each month are explicitly defined with the `month_lengths` attribute of the time axis:

`month_lengths`
: A vector of size 12, specifying the number of days in the months from January to December (in a non-leap year).

If leap years are included, then two other attributes of the time axis should also be defined:

`leap_year`
: An example of a leap year. It is assumed that all years that differ from this year by a multiple of four are also leap years. If this attribute is absent, it is assumed there are no leap years.

`leap_month`
: A value in the range 1-12, specifying which month is lengthened by a day in leap years (1=January). If this attribute is not present, February (2) is assumed. This attribute is ignored if `leap_year` is not specified.

The `calendar` attribute is not required when a non-standard calendar is being used. It is sufficient to define the calendar using the `month_lengths` attribute, along with `leap_year`, and `leap_month` as appropriate. However, the `calendar` attribute is allowed to take non-standard values and in that case defining the non-standard calendar using the appropriate attributes is required.

Example 4.6. Paleoclimate time axis

```
double time(time) ;
  time:long_name = "time" ;
  time:units = "days since 1-1-1 0:0:0" ;
  time:calendar = "126 kyr B.P." ;
  time:month_lengths = 34, 31, 32, 30, 29, 27, 28, 28, 28, 32, 32, 34 ;
```

The mixed Gregorian/Julian calendar used by Udunits is explained in the following excerpt from the udunits(3) man page:

```
The udunits(3) package uses a mixed Gregorian/Julian  calen-
dar  system.   Dates  prior to 1582-10-15 are assumed to use
the Julian calendar, which was introduced by  Julius  Caesar
in 46 BCE and is based on a year that is exactly 365.25 days
long.  Dates on and after 1582-10-15 are assumed to use  the
Gregorian calendar, which was introduced on that date and is
based on a year that is exactly 365.2425 days long.  (A year
is  actually  approximately 365.242198781 days long.)  Seem-
ingly strange behavior of the udunits(3) package can  result
if  a user-given time interval includes the changeover date.
For example, utCalendar() and utInvCalendar() can be used to
show that 1582-10-15 *preceded* 1582-10-14 by 9 days.
```

Due to problems caused by the discontinuity in the default mixed Gregorian/Julian calendar, we strongly recommend that this calendar should only be used when the time coordinate does not cross the discontinuity. For time coordinates that do cross the discontinuity the `proleptic_gregorian` calendar should be used instead.

4.5. Discrete Axis

The spatiotemporal coordinates described in sections 4.1-4.4 are continuous variables, and other geophysical quantities may likewise serve as continuous coordinate variables, for instance density, temperature or radiation wavelength. By contrast, for some purposes there is a need for an axis of a data variable which indicates either an ordered list or an unordered collection, and does not correspond to any continuous coordinate variable. Consequently such an axis may be called “discrete”. A discrete axis has a dimension but might not have a coordinate variable. Instead, there might be one or more auxiliary coordinate variables with this dimension (see preamble to section 5). Following sections define various applications of discrete axes, for instance section 6.1.1 “Geographical regions”, section 7.3.3 “Statistics applying to portions of cells”, section 9.3 “Representation of collections of features in data variables”.

Chapter 5. Coordinate Systems

A variable's spatiotemporal dimensions are used to locate data values in time and space. This is accomplished by associating these dimensions with the relevant set of latitude, longitude, vertical, and time coordinates. This section presents two methods for making that association: the use of *coordinate variables*, and the use of *auxiliary coordinate variables*.

All of a variable's dimensions that are latitude, longitude, vertical, or time dimensions (see Section 1.2, "Terminology") must have corresponding coordinate variables, i.e., one-dimensional variables with the same name as the dimension (see examples in Chapter 4, Coordinate Types). This is the only method of associating dimensions with coordinates that is supported by [COARDS].

All of a variable's spatiotemporal dimensions that are not latitude, longitude, vertical, or time dimensions are required to be associated with the relevant latitude, longitude, vertical, or time coordinates via the new `coordinates` attribute of the variable. The value of the `coordinates` attribute is *a blank separated list of the names of auxiliary coordinate variables*. There is no restriction on the order in which the auxiliary coordinate variables appear in the `coordinates` attribute string. The dimensions of an auxiliary coordinate variable must be a subset of the dimensions of the variable with which the coordinate is associated, with two exceptions. First, string-valued coordinates (Section 6.1, "Labels") have a dimension for maximum string length. Second, in the ragged array representations of data (Chapter 9, Discrete Sampling Geometries), special methods are needed to connect the data and coordinates

We recommend that the name of a multidimensional coordinate variable should not match the name of any of its dimensions because that precludes supplying a coordinate variable for the dimension. This practice also avoids potential bugs in applications that determine coordinate variables by only checking for a name match between a dimension and a variable and not checking that the variable is one dimensional.

The use of coordinate variables is required whenever they are applicable. That is, auxiliary coordinate variables may not be used as the only way to identify latitude and longitude coordinates that could be identified using coordinate variables. This is both to enhance conformance to COARDS and to facilitate the use of generic applications that recognize the NUG convention for coordinate variables. An application that is trying to find the latitude coordinate of a variable should always look first to see if any of the variable's dimensions correspond to a latitude coordinate variable. If the latitude coordinate is not found this way, then the auxiliary coordinate variables listed by the `coordinates` attribute should be checked. Note that it is permissible, but optional, to list coordinate variables as well as auxiliary coordinate variables in the `coordinates` attribute.

If an `axis` attribute is attached to an auxiliary coordinate variable, it can be used by applications in the same way the `axis` attribute attached to a coordinate variable is used. However, it is not permissible for a data variable to have both a coordinate variable and an auxiliary coordinate variable, or more than one of either type of variable, having an `axis` attribute with any given value e.g. there must be no more than one `axis` attribute for `X` for any data variable. Note that if the `axis` attribute is not specified for an auxiliary coordinate variable, it may still be possible to determine if it is a spatiotemporal dimension from its own units or standard_name, or from the units and standard_name of the coordinate variable corresponding to its dimensions (see Chapter 4, Coordinate Types). For instance, auxiliary coordinate variables which lie on the horizontal surface

can be identified as such by their dimensions being horizontal. Horizontal dimensions are those whose coordinate variables have an `axis` attribute of `X` or `Y`, or a `units` attribute indicating latitude and longitude.

If the coordinate variables for a horizontal grid are not longitude and latitude, it is recommended that they be supplied *in addition* to the required coordinates. For example, the Cartesian coordinates of a map projection should be supplied as coordinate variables in addition to the required two-dimensional latitude and longitude variables that are identified via the `coordinates` attribute. The use of the `axis` attribute with values `X` and `Y` is recommended for the coordinate variables (see Chapter 4, Coordinate Types).

It is sometimes not practical to specify the latitude-longitude location of data which is representative of geographic regions with complex boundaries. For this purpose, provision is made in Section 6.1.1, "Geographic Regions" for indicating the region by a standardized name.

5.1. Independent Latitude, Longitude, Vertical, and Time Axes

When each of a variable's spatiotemporal dimensions is a latitude, longitude, vertical, or time dimension, then each axis is identified by a coordinate variable.

Example 5.1. Independent coordinate variables

```
dimensions:
  lat = 18 ;
  lon = 36 ;
  pres = 15 ;
  time = 4 ;
variables:
  float xwind(time,pres,lat,lon) ;
    xwind:long_name = "zonal wind" ;
    xwind:units = "m/s" ;
  float lon(lon) ;
    lon:long_name = "longitude" ;
    lon:units = "degrees_east" ;
  float lat(lat) ;
    lat:long_name = "latitude" ;
    lat:units = "degrees_north" ;
  float pres(pres) ;
    pres:long_name = "pressure" ;
    pres:units = "hPa" ;
  double time(time) ;
    time:long_name = "time" ;
    time:units = "days since 1990-1-1 0:0:0" ;
```

`xwind(n,k,j,i)` is associated with the coordinate values `lon(i)`, `lat(j)`, `pres(k)`, and `time(n)`.

5.2. Two-Dimensional Latitude, Longitude, Coordinate Variables

The latitude and longitude coordinates of a horizontal grid that was not defined as a Cartesian product of latitude and longitude axes, can sometimes be represented using two-dimensional coordinate variables. These variables are identified as coordinates by use of the `coordinates` attribute.

Example 5.2. Two-dimensional coordinate variables

```
dimensions:
  xc = 128 ;
  yc = 64 ;
  lev = 18 ;
variables:
  float T(lev,yc,xc) ;
    T:long_name = "temperature" ;
    T:units = "K" ;
    T:coordinates = "lon lat" ;
  float xc(xc) ;
    xc:axis = "X" ;
    xc:long_name = "x-coordinate in Cartesian system" ;
    xc:units = "m" ;
  float yc(yc) ;
    yc:axis = "Y" ;
    yc:long_name = "y-coordinate in Cartesian system" ;
    yc:units = "m" ;
  float lev(lev) ;
    lev:long_name = "pressure level" ;
    lev:units = "hPa" ;
  float lon(yc,xc) ;
    lon:long_name = "longitude" ;
    lon:units = "degrees_east" ;
  float lat(yc,xc) ;
    lat:long_name = "latitude" ;
    lat:units = "degrees_north" ;
```

`T(k,j,i)` is associated with the coordinate values `lon(j,i)`, `lat(j,i)`, and `lev(k)`. The vertical coordinate is represented by the coordinate variable `lev(lev)` and the latitude and longitude coordinates are represented by the auxiliary coordinate variables `lat(yc,xc)` and `lon(yc,xc)` which are identified by the `coordinates` attribute.

Note that coordinate variables are also defined for the `xc` and `yc` dimensions. This faciliates processing of this data by generic applications that don't recognize the multidimensional latitude and longitude coordinates.

5.3. Reduced Horizontal Grid

A "reduced" longitude-latitude grid is one in which the points are arranged along constant latitude lines with the number of points on a latitude line decreasing toward the poles. Storing this type of gridded data in two-dimensional arrays wastes space, and results in the presence of missing values in the 2D coordinate variables. We recommend that this type of gridded data be stored using the compression scheme described in Section 8.2, "Compression by Gathering". Compression by gathering preserves structure by storing a set of indices that allows an application to easily scatter the compressed data back to two-dimensional arrays. The compressed latitude and longitude auxiliary coordinate variables are identified by the `coordinates` attribute.

Example 5.3. Reduced horizontal grid

```
dimensions:
  londim = 128 ;
  latdim = 64 ;
  rgrid = 6144 ;
variables:
  float PS(rgrid) ;
    PS:long_name = "surface pressure" ;
    PS:units = "Pa" ;
    PS:coordinates = "lon lat" ;
  float lon(rgrid) ;
    lon:long_name = "longitude" ;
    lon:units = "degrees_east" ;
  float lat(rgrid) ;
    lat:long_name = "latitude" ;
    lat:units = "degrees_north" ;
  int rgrid(rgrid);
    rgrid:compress = "latdim londim";
```

`PS(n)` is associated with the coordinate values `lon(n)`, `lat(n)`. Compressed grid index `(n)` would be assigned to 2D index `(j,i)` (C index conventions) where

```
j = rgrid(n) / 128
i = rgrid(n) - 128*j
```

Notice that even if an application does not recognize the `compress` attribute, the grids stored in this format can still be handled, by an application that recognizes the `coordinates` attribute.

5.4. Timeseries of Station Data

This section has been superseded by the treatment of time series as a type of discrete sampling geometry in Chapter 9.

5.5. Trajectories

This section has been superseded by the treatment of time series as a type of discrete sampling geometry in Chapter 9.

5.6. Horizontal Coordinate Reference Systems, Grid Mappings, and Projections

When the coordinate variables for a horizontal grid are not longitude and latitude, it is required that the true latitude and longitude coordinates be supplied via the `coordinates` attribute. If in addition it is desired to describe the mapping between the given coordinate variables and the true latitude and longitude coordinates, the attribute `grid_mapping` may be used to supply this description. This attribute is attached to data variables so that variables with different mappings may be present in a single file. The attribute takes a string value which is the name of another variable in the file that provides the description of the mapping via a collection of attached attributes. This variable is called *a grid mapping variable* and is of arbitrary type since it contains no data. Its purpose is to act as a container for the attributes that define the mapping. The one attribute that all grid mapping variables must have is `grid_mapping_name` which takes a string value that contains the mapping's name. The other attributes that define a specific mapping depend on the value of `grid_mapping_name`. The valid values of `grid_mapping_name` along with the attributes that provide specific map parameter values are described in Appendix F, Grid Mappings.

When the coordinate variables for a horizontal grid are longitude and latitude, a grid mapping variable with `grid_mapping_name` of `latitude_longitude` may be used to specify the ellipsoid and prime meridian.

In order to make use of a grid mapping to directly calculate latitude and longitude values it is necessary to associate the coordinate variables with the independent variables of the mapping. This is done by assigning a `standard_name` to the coordinate variable. The appropriate values of the `standard_name` depend on the grid mapping and are given in Appendix F, Grid Mappings.

Example 5.6. Rotated pole grid

```
dimensions:
  rlon = 128 ;
  rlat = 64 ;
  lev = 18 ;
variables:
  float T(lev,rlat,rlon) ;
    T:long_name = "temperature" ;
    T:units = "K" ;
    T:coordinates = "lon lat" ;
    T:grid_mapping = "rotated_pole" ;
  char rotated_pole
    rotated_pole:grid_mapping_name = "rotated_latitude_longitude" ;
    rotated_pole:grid_north_pole_latitude = 32.5 ;
    rotated_pole:grid_north_pole_longitude = 170. ;
  float rlon(rlon) ;
    rlon:long_name = "longitude in rotated pole grid" ;
    rlon:units = "degrees" ;
    rlon:standard_name = "grid_longitude";
  float rlat(rlat) ;
    rlat:long_name = "latitude in rotated pole grid" ;
    rlat:units = "degrees" ;
    rlon:standard_name = "grid_latitude";
  float lev(lev) ;
    lev:long_name = "pressure level" ;
    lev:units = "hPa" ;
  float lon(rlat,rlon) ;
    lon:long_name = "longitude" ;
    lon:units = "degrees_east" ;
  float lat(rlat,rlon) ;
    lat:long_name = "latitude" ;
    lat:units = "degrees_north" ;
```

A CF compliant application can determine that rlon and rlat are longitude and latitude values in the rotated grid by recognizing the standard names `grid_longitude` and `grid_latitude`. Note that the units of the rotated longitude and latitude axes are given as `degrees`. This should prevent a COARDS compliant application from mistaking the variables `rlon` and `rlat` to be actual longitude and latitude coordinates. The entries for these names in the standard name table indicate the appropriate sign conventions for the units of `degrees`.

Example 5.7. Lambert conformal projection

```
dimensions:
  y = 228;
  x = 306;
  time = 41;

variables:
  int Lambert_Conformal;
    Lambert_Conformal:grid_mapping_name = "lambert_conformal_conic";
    Lambert_Conformal:standard_parallel = 25.0;
    Lambert_Conformal:longitude_of_central_meridian = 265.0;
    Lambert_Conformal:latitude_of_projection_origin = 25.0;
  double y(y);
    y:units = "km";
    y:long_name = "y coordinate of projection";
    y:standard_name = "projection_y_coordinate";
  double x(x);
    x:units = "km";
    x:long_name = "x coordinate of projection";
    x:standard_name = "projection_x_coordinate";
  double lat(y, x);
    lat:units = "degrees_north";
    lat:long_name = "latitude coordinate";
    lat:standard_name = "latitude";
  double lon(y, x);
    lon:units = "degrees_east";
    lon:long_name = "longitude coordinate";
    lon:standard_name = "longitude";
  int time(time);
    time:long_name = "forecast time";
    time:units = "hours since 2004-06-23T22:00:00Z";
  float Temperature(time, y, x);
    Temperature:units = "K";
    Temperature:long_name = "Temperature @ surface";
    Temperature:missing_value = 9999.0;
    Temperature:coordinates = "lat lon";
    Temperature:grid_mapping = "Lambert_Conformal";
```

An application can determine that `x` and `y` are the projection coordinates by recognizing the standard names `projection_x_coordinate` and `projection_y_coordinate`. The grid mapping variable `Lambert_Conformal` contains the mapping parameters as attributes, and is associated with the `Temperature` variable via its `grid_mapping` attribute.

Example 5.8. Latitude and longitude on a spherical Earth

```
dimensions:
  lat = 18 ;
  lon = 36 ;
variables:
  double lat(lat) ;
  double lon(lon) ;
  float temp(lat, lon) ;
    temp:long_name = "temperature" ;
    temp:units = "K" ;
    temp:grid_mapping = "crs" ;
  int crs ;
    crs:grid_mapping_name = "latitude_longitude"
    crs:semi_major_axis = 6371000.0 ;
    crs:inverse_flattening = 0 ;
```

Example 5.9. Latitude and longitude on the WGS 1984 datum

```
dimensions:
  lat = 18 ;
  lon = 36 ;
variables:
  double lat(lat) ;
  double lon(lon) ;
  float temp(lat, lon) ;
    temp:long_name = "temperature" ;
    temp:units = "K" ;
    temp:grid_mapping = "crs" ;
  int crs ;
    crs:grid_mapping_name = "latitude_longitude";
    crs:longitude_of_prime_meridian = 0.0 ;
    crs:semi_major_axis = 6378137.0 ;
    crs:inverse_flattening = 298.257223563 ;
```

Example 5.10. British National Grid

```
dimensions:
  lat = 648 ;
  lon = 648 ;
  y = 18 ;
  x = 36 ;
variables:
  double x(x) ;
    x:standard_name = "projection_x_coordinate" ;
    x:units = "m" ;
  double y(y) ;
    y:standard_name = "projection_y_coordinate" ;
    y:units = "m" ;
  double lat(y, x) ;
  double lon(y, x) ;
  float temp(y, x) ;
    temp:long_name = "temperature" ;
    temp:units = "K" ;
    temp:coordinates = "lat lon" ;
    temp:grid_mapping = "crs" ;
  int crs ;
    crs:grid_mapping_name = "transverse_mercator";
    crs:semi_major_axis = 6377563.396 ;
    crs:semi_minor_axis = 6356256.910 ;
    crs:inverse_flattening = 299.3249646 ;
    crs:latitude_of_projection_origin = 49.0 ;
    crs:longitude_of_projection_origin = -2.0 ;
    crs:false_easting = 400000.0 ;
    crs:false_northing = -100000.0 ;
    crs:scale_factor_at_projection_origin = 0.9996012717 ;
```

5.7. Scalar Coordinate Variables

When a variable has an associated coordinate which is single-valued, that coordinate may be represented as a scalar variable. Since there is no associated dimension these scalar coordinate variables should be attached to a data variable via the `coordinates` attribute.

Under COARDS the method of providing a single valued coordinate was to add a dimension of size one to the variable, and supply the corresponding coordinate variable. The new scalar coordinate variable is a convenience feature which avoids adding size one dimensions to variables. Scalar coordinate variables have the same information content and can be used in the same contexts as a size one coordinate variable. Note however that use of this feature with a latitude, longitude, vertical, or time coordinate will inhibit COARDS conforming applications from recognizing them.

Once a name is used for a scalar coordinate variable it can not be used for a 1D coordinate variable. For this reason we strongly recommend against using a name for a scalar coordinate variable that

matches the name of any dimension in the file.

Example 5.11. Multiple forecasts from a single analysis

```
dimensions:
  lat = 180 ;
  lon = 360 ;
  time = UNLIMITED ;
variables:
  double atime
    atime:standard_name = "forecast_reference_time" ;
    atime:units = "hours since 1999-01-01 00:00" ;
  double time(time);
    time:standard_name = "time" ;
    time:units = "hours since 1999-01-01 00:00" ;
  double lon(lon) ;
    lon:long_name = "station longitude";
    lon:units = "degrees_east";
  double lat(lat) ;
    lat:long_name = "station latitude" ;
    lat:units = "degrees_north" ;
  double p500
    p500:long_name = "pressure" ;
    p500:units = "hPa" ;
    p500:positive = "down" ;
  float height(time,lat,lon);
    height:long_name = "geopotential height" ;
    height:standard_name = "geopotential_height" ;
    height:units = "m" ;
    height:coordinates = "atime p500" ;
data:
  time = 6., 12., 18., 24. ;
  atime = 0. ;
  p500 = 500. ;
```

In this example both the analysis time and the single pressure level are represented using scalar coordinate variables. The analysis time is identified by the standard name "forecast_reference_time" while the valid time of the forecast is identified by the standard name "time".

Chapter 6. Labels and Alternative Coordinates

6.1. Labels

Character strings can be used to provide a name or label for each element of an axis. This is particularly useful for discrete axes (section 4.5). For instance, if a data variable contains time series of observational data from a number of observing stations, it may be convenient to provide the names of the stations as labels for the elements of the station dimension (Section H.2, "Time Series Data"). Example H.1, "Point data" illustrates another application for labels.

Character strings labelling the elements of an axis are regarded as string-valued auxiliary coordinate variables. The `coordinates` attribute of the data variable names the variable that contains the string array. An application processing the variables listed in the `coordinates` attribute can recognize a string-valued auxiliary coordinate variable because it contains an array of character data. The inner dimension (last dimension in CDL terms) is the maximum length of each string, and the other dimensions are axis dimensions. If a character variable has only one dimension (the maximum length of the string), it is regarded as a string-valued scalar coordinate variable, analogous to a numeric scalar coordinate variable (see Section 5.7, "Scalar Coordinate Variables")

6.1.1. Geographic Regions

When data is representative of geographic regions which can be identified by names but which have complex boundaries that cannot practically be specified using longitude and latitude boundary coordinates, a labeled axis should be used to identify the regions. We recommend that the names be chosen from the list of standardized region names whenever possible. To indicate that the label values are standardized the variable that contains the labels must be given the `standard_name` attribute with the value `region`.

Example 6.2. Northward heat transport in Atlantic Ocean

Suppose we have data representing northward heat transport across a set of zonal slices in the Atlantic Ocean. Note that the standard names to describe this quantity do not include location information. That is provided by the latitude coordinate and the labeled axis:

```
dimensions:
  times = 20 ;
  lat = 5
  lbl = 1 ;
  strlen = 64 ;
variables:
  float n_heat_transport(time,lat,lbl);
    n_heat_transport:units="W";
    n_heat_transport:coordinates="geo_region";
    n_heat_transport:standard_name="northward_ocean_heat_transport";
  double time(time) ;
    time:long_name = "time" ;
    time:units = "days since 1990-1-1 0:0:0" ;
  float lat(lat) ;
    lat:long_name = "latitude" ;
    lat:units = "degrees_north" ;
  char geo_region(lbl,strlen) ;
    geo_region:standard_name="region"
data:
  geo_region = "atlantic_ocean" ;
  lat = 10., 20., 30., 40., 50. ;
```

6.2. Alternative Coordinates

In some situations a dimension may have alternative sets of coordinates values. Since there can only be one coordinate variable for the dimension (the variable with the same name as the dimension), any alternative sets of values have to be stored in auxiliary coordinate variables. For such alternative coordinate variables, there are no mandatory attributes, but they may have any of the attributes allowed for coordinate variables.

Example 6.3. Model level numbers

Levels on a vertical axis may be described by both the physical coordinate and the ordinal model level number.

```
float xwind(sigma,lat);
  xwind:coordinates="model_level";
float sigma(sigma); // physical height coordinate
  sigma:long_name="sigma";
  sigma:positive="down";
int model_level(sigma); // model level number at each height
  model_level:long_name="model level number";
  model_level:positive="up";
```

Chapter 7. Data Representative of Cells

When gridded data does not represent the point values of a field but instead represents some characteristic of the field within cells of finite "volume," a complete description of the variable should include metadata that describes the domain or extent of each cell, and the characteristic of the field that the cell values represent. It is possible for a single data value to be the result of an operation whose domain is a disjoint set of cells. This is true for many types of climatological averages, for example, the mean January temperature for the years 1970-2000. The methods that we present below for describing cells only provides an association of a grid point with a single cell, not with a collection of cells. However, climatological statistics are of such importance that we provide special methods for describing their associated computational domains in Section 7.4, "Climatological Statistics".

7.1. Cell Boundaries

To represent cells we add the attribute `bounds` to the appropriate coordinate variable(s). The value of `bounds` is the name of the variable that contains the vertices of the cell boundaries. We refer to this type of variable as a "boundary variable." *A boundary variable will have one more dimension than its associated coordinate or auxiliary coordinate variable.* The additional dimension should be the most rapidly varying one, and its size is the maximum number of cell vertices. Since a boundary variable is considered to be part of a coordinate variable's metadata, it is not necessary to provide it with attributes such as `long_name` and `units`.

Note that the boundary variable for a set of N contiguous intervals is an array of shape (N,2). Although in this case there will be a duplication of the boundary coordinates between adjacent intervals, this representation has the advantage that it is general enough to handle, without modification, non-contiguous intervals, as well as intervals on an axis using the unlimited dimension.

Applications that process cell boundary data often times need to determine whether or not adjacent cells share an edge. In order to facilitate this type of processing the following restrictions are placed on the data in boundary variables.

Bounds for 1-D coordinate variables

For a coordinate variable such as `lat(lat)` with associated boundary variable `latbnd(x,2)`, the interval endpoints must be ordered consistently with the associated coordinate, e.g., for an increasing coordinate, `lat(1)` > `lat(0)` implies `latbnd(i,1)` >= `latbnd(i,0)` for all `i`

If adjacent intervals are contiguous, the shared endpoint must be represented indentically in each instance where it occurs in the boundary variable. For example, if the intervals that contain grid points `lat(i)` and `lat(i+1)` are contiguous, then `latbnd(i+1,0)` = `latbnd(i,1)`.

Bounds for 2-D coordinate variables with 4-sided cells

In the case where the horizontal grid is described by two-dimensional auxiliary coordinate variables in latitude `lat(n,m)` and longitude `lon(n,m)`, and the associated cells are four-sided, then the boundary variables are given in the form `latbnd(n,m,4)` and `lonbnd(n,m,4)`, where the trailing index runs over the four vertices of the cells. Let us call the side of cell `(j,i)` facing cell

`(j,i-1)` the "`i-1`" side, the side facing cell `(j,i+1)` the "`i+1`" side, and similarly for "`j-1`" and "`j+1`". Then we can refer to the vertex formed by sides `i-1` and `j-1` as `(j-1,i-1)`. With this notation, the four vertices are indexed as follows: `0=(j-1,i-1)`, `1=(j-1,i+1)`, `2=(j+1,i+1)`, `3=(j+1,i-1)`.

If i-j-upward is a right-handed coordinate system (like lon-lat-upward), this ordering means the vertices will be traversed anticlockwise on the lon-lat surface seen from above. If i-j-upward is left-handed, they will be traversed clockwise on the lon-lat surface.

The bounds can be used to decide whether cells are contiguous via the following relationships. In these equations the variable `bnd` is used generically to represent either the latitude or longitude boundary variable.

```
For 0 < j < n and 0 < i < m,
    If cells (j,i) and (j,i+1) are contiguous, then
        bnd(j,i,1)=bnd(j,i+1,0)
        bnd(j,i,2)=bnd(j,i+1,3)
    If cells (j,i) and (j+1,i) are contiguous, then
        bnd(j,i,3)=bnd(j+1,i,0) and bnd(j,i,2)=bnd(j+1,i,1)
```

Bounds for multi-dimensional coordinate variables with p-sided cells

In all other cases, the bounds should be dimensioned `(⋯,n,p)`, where `(⋯,n)` are the dimensions of the auxiliary coordinate variables, and `p` the number of vertices of the cells. The vertices must be traversed anticlockwise in the lon-lat plane as viewed from above. The starting vertex is not specified.

Example 7.1. Cells on a latitude axis

```
dimensions:
  lat = 64;
  nv = 2;    // number of vertices
variables:
  float lat(lat);
    lat:long_name = "latitude";
    lat:units = "degrees_north";
    lat:bounds = "lat_bnds";
  float lat_bnds(lat,nv);
```

The boundary variable `lat_bnds` associates a latitude gridpoint `i` with the interval whose boundaries are `lat_bnds(i,0)` and `lat_bnds(i,1)`. The gridpoint location, `lat(i)`, should be contained within this interval.

For rectangular grids, two-dimensional cells can be expressed as Cartesian products of one-dimensional cells of the type in the preceding example. However for non-rectangular grids a "rectangular" cell will in general require specifying all four vertices for each cell.

Example 7.2. Cells in a non-rectangular grid

```
dimensions:
  imax = 128;
  jmax = 64;
  nv = 4;
variables:
  float lat(jmax,imax);
    lat:long_name = "latitude";
    lat:units = "degrees_north";
    lat:bounds = "lat_bnds";
  float lon(jmax,imax);
    lon:long_name = "longitude";
    lon:units = "degrees_east";
    lon:bounds = "lon_bnds";
  float lat_bnds(jmax,imax,nv);
  float lon_bnds(jmax,imax,nv);
```

The boundary variables `lat_bnds` and `lon_bnds` associate a gridpoint `(j,i)` with the cell determined by the vertices `(lat_bnds(j,i,n),lon_bnds(j,i,n))`, `n=0,..,3`. The gridpoint location, `(lat(j,i),lon(j,i))`, should be contained within this region.

7.2. Cell Measures

For some calculations, information is needed about the size, shape or location of the cells that cannot be deduced from the coordinates and bounds without special knowledge that a generic application cannot be expected to have. For instance, in computing the mean of several cell values, it is often appropriate to "weight" the values by area. When computing an area-mean each grid cell value is multiplied by the grid-cell area before summing, and then the sum is divided by the sum of the grid-cell areas. Area weights may also be needed to map data from one grid to another in such a way as to preserve the area mean of the field. The preservation of area-mean values while regridding may be essential, for example, when calculating surface heat fluxes in an atmospheric model with a grid that differs from the ocean model grid to which it is coupled.

In many cases the areas can be calculated from the cell bounds, but there are exceptions. Consider, for example, a spherical geodesic grid composed of contiguous, roughly hexagonal cells. The vertices of the cells can be stored in the variable identified by the `bounds` attribute, but the cell perimeter is not uniquely defined by its vertices (because the vertices could, for example, be connected by straight lines, or, on a sphere, by lines following a great circle, or, in general, in some other way). Thus, given the cell vertices alone, it is generally impossible to calculate the area of a grid cell. This is why it may be necessary to store the grid-cell areas in addition to the cell vertices.

In other cases, the grid cell-volume might be needed and might not be easily calculated from the coordinate information. In ocean models, for example, it is not uncommon to find "partial" grid cells at the bottom of the ocean. In this case, rather than (or in addition to) indicating grid cell area, it may be necessary to indicate volume.

To indicate extra information about the spatial properties of a variable's grid cells, a `cell_measures` attribute may be defined for a variable. This is a string attribute comprising a list of blank-separated pairs of words of the form "`measure: name`". For the moment, "`area`" and "`volume`" are the only defined measures, but others may be supported in future. The "name" is the name of the variable containing the measure values, which we refer to as a "measure variable". The dimensions of the measure variable should be the same as or a subset of the dimensions of the variable to which they are related, but their order is not restricted. In the case of area, for example, the field itself might be a function of longitude, latitude, and time, but the variable containing the area values would only include longitude and latitude dimensions (and the dimension order could be reversed, although this is not recommended). The variable must have a `units` attribute and may have other attributes such as a `standard_name`.

For rectangular longitude-latitude grids, the area of grid cells can be calculated from the bounds: the area of a cell is proportional to the product of the difference in the longitude bounds of the cell and the difference between the sine of each latitude bound of the cell. In this case supplying grid-cell areas via the `cell_measures` attribute is unnecessary because it may be assumed that applications can perform this calculation, using their own value for the radius of the Earth.

Example 7.3. Cell areas for a spherical geodesic grid

```
dimensions:
  cell = 2562 ;  // number of grid cells
  time = 12 ;
  nv = 6 ;       // maximum number of cell vertices
variables:
  float PS(time,cell) ;
    PS:units = "Pa" ;
    PS:coordinates = "lon lat" ;
    PS:cell_measures = "area: cell_area" ;
  float lon(cell) ;
    lon:long_name = "longitude" ;
    lon:units = "degrees_east" ;
    lon:bounds="lon_vertices" ;
  float lat(cell) ;
    lat:long_name = "latitude" ;
    lat:units = "degrees_north" ;
    lat:bounds="lat_vertices" ;
  float time(time) ;
    time:long_name = "time" ;
    time:units = "days since 1979-01-01 0:0:0" ;
  float cell_area(cell) ;
    cell_area:long_name = "area of grid cell" ;
    cell_area:standard_name="area";
    cell_area:units = "m2"
  float lon_vertices(cell,nv) ;
  float lat_vertices(cell,nv) ;
```

7.3. Cell Methods

To describe the characteristic of a field that is represented by cell values, we define the `cell_methods` attribute of the variable. This is a string attribute comprising a list of blank-separated words of the form "*name: method*". Each "*name: method*" pair indicates that for an axis identified by *name*, the cell values representing the field have been determined or derived by the specified *method*. For example, if data values have been generated by computing time means, then this could be indicated with `cell_methods="t: mean"`, assuming here that the name of the time dimension variable is "t".

In the specification of this attribute, *name* can be a dimension of the variable, a scalar coordinate variable, a valid standard name, or the word "`area`". (See Section 7.3.4, "Cell methods when there are no coordinates" concerning the use of standard names in cell_methods.) The values of *method* should be selected from the list in Appendix E, Cell Methods, which includes `point`, `sum`, `mean`, `maximum`, `minimum`, `mid_range`, `standard_deviation`, `variance`, `mode`, and `median`. Case is not significant in the method name. Some methods (e.g., `variance`) imply a change of units of the variable, as is indicated in Appendix E, Cell Methods.

It must be remembered that the method applies only to the axis designated in `cell_methods` by *name*, and different methods may apply to other axes. If, for instance, a precipitation value in a longitude-latitude cell is given the method `maximum` for these axes, it means that it is the maximum within these spatial cells, and does not imply that it is also the maximum in time. Furthermore, it should be noted that if any *method* other than "`point`" is specified for a given axis, then `cell_bounds` should also be provided for that axis (except for the relatively rare exceptions described in Section 7.3.4, "Cell methods when there are no coordinates").

The default interpretation for variables that do not have the `cell_methods` attribute specified depends on whether the quantity is extensive (which depends on the size of the cell) or intensive (which does not). Suppose, for example, the quantities "accumulated precipitation" and "precipitation rate" each have a time axis. A variable representing accumulated precipitation is extensive in time because it depends on the length of the time interval over which it is accumulated. For correct interpretation, it therefore requires a time interval to be completely specified via a boundary variable (i.e., via a `cell_bounds` attribute for the time axis). In this case the default interpretation is that the cell method is a sum over the specified time interval. This can be (optionally) indicated explicitly by setting the cell method to `sum`. A precipitation rate on the other hand is intensive in time and could equally well represent either an instantaneous value or a mean value over the time interval specified by the cell. In this case the default interpretation for the quantity would be "instantaneous" (which, optionally, can be indicated explicitly by setting the cell method to `point`). More often, however, cell values for intensive quantities are means, and this should be indicated explicitly by setting the cell method to `mean` and specifying the cell bounds.

Because the default interpretation for an intensive quantity differs from that of an extensive quantity and because this distinction may not be understood by some users of the data, it is recommended that every data variable include for each of its dimensions and each of its scalar coordinate variables the `cell_methods` information of interest (unless this information would not be meaningful). It is especially recommended that `cell_methods` be explicitly specified for each spatio-temporal dimension and each spatio-temporal scalar coordinate variable.

Example 7.4. Methods applied to a timeseries

Consider 12-hourly timeseries of pressure, temperature and precipitation from a number of stations, where pressure is measured instantaneously, maximum temperature for the preceding 12 hours is recorded, and precipitation is accumulated in a rain gauge. For a period of 48 hours from 6 a.m. on 19 April 1998, the data is structured as follows:

```
dimensions:
  time = UNLIMITED; // (5 currently)
  station = 10;
  nv = 2;
variables:
  float pressure(time,station);
    pressure:long_name = "pressure";
    pressure:units = "kPa";
    pressure:cell_methods = "time: point";
  float maxtemp(time,station);
    maxtemp:long_name = "temperature";
    maxtemp:units = "K";
    maxtemp:cell_methods = "time: maximum";
  float ppn(time,station);
    ppn:long_name = "depth of water-equivalent precipitation";
    ppn:units = "mm";
    ppn:cell_methods = "time: sum";
  double time(time);
    time:long_name = "time";
    time:units = "h since 1998-4-19 6:0:0";
    time:bounds = "time_bnds";
  double time_bnds(time,nv);
data:
  time = 0., 12., 24., 36., 48.;
  time_bnds = -12.,0., 0.,12., 12.,24., 24.,36., 36.,48.;
```

Note that in this example the time axis values coincide with the end of each interval. It is sometimes desirable, however, to use the midpoint of intervals as coordinate values for variables that are representative of an interval. An application may simply obtain the midpoint values by making use of the boundary data in `time_bnds`.

7.3.1. Statistics for more than one axis

If more than one cell method is to be indicated, they should be arranged in the order they were applied. The left-most operation is assumed to have been applied first. Suppose, for example, that within each grid cell a quantity varies in both longitude and time and that these dimensions are named "lon" and "time", respectively. Then values representing the time-average of the zonal maximum are labeled `cell_methods="lon: maximum time: mean"` (i.e. find the largest value at each instant of time over all longitudes, then average these maxima over time); values of the zonal maximum of time-averages are labeled `cell_methods="time: mean lon: maximum"`. If the methods could have been applied in any order without affecting the outcome, they may be put in any order

in the `cell_methods` attribute.

If a data value is representative of variation over a combination of axes, a single method should be prefixed by the names of all the dimensions involved (listed in any order, since in this case the order must be immaterial). Dimensions should be grouped in this way only if there is an essential difference from treating the dimensions individually. For instance, the standard deviation of topographic height within a longitude-latitude gridbox could have `cell_methods="lat: lon: standard_deviation"`. (Note also, that in accordance with the recommendation of the following paragraph, this could be equivalently and preferably indicated by `cell_methods="area: standard_deviation"`.) This is not the same as `cell_methods="lon: standard_deviation lat: standard_deviation"`, which would mean finding the standard deviation along each parallel of latitude within the zonal extent of the gridbox, and then the standard deviation of these values over latitude.

To indicate variation over horizontal area, it is recommended that instead of specifying the combination of horizontal dimensions, the special string "`area`" be used. The common case of an area-mean can thus be indicated by `cell_methods="area: mean"` (rather than, for example, "`lon: lat: mean`"). The horizontal coordinate variables to which "`area`" refers are in this case not explicitly indicated in `cell_methods` but can be identified, if necessary, from attributes attached to the coordinate variables, scalar coordinate variables, or auxiliary coordinate variables, as described in Chapter 4, Coordinate Types.

7.3.2. Recording the spacing of the original data and other information

To indicate more precisely how the cell method was applied, extra information may be included in parentheses () after the identification of the method. This information includes standardized and non-standardized parts. Currently the only standardized information is to provide the typical interval between the original data values to which the method was applied, in the situation where the present data values are statistically representative of original data values which had a finer spacing. The syntax is (`interval`: *value unit*), where *value* is a numerical value and *unit* is a string that can be recognized by UNIDATA's Udunits package [UDUNITS]. The *unit* will usually be dimensionally equivalent to the unit of the corresponding dimension, but this is not required (which allows, for example, the interval for a standard deviation calculated from points evenly spaced in distance along a parallel to be reported in units of length even if the zonal coordinate of the cells is given in degrees). Recording the original interval is particularly important for standard deviations. For example, the standard deviation of daily values could be indicated by `cell_methods="time: standard_deviation (interval: 1 day)"` and of annual values by `cell_methods="time: standard_deviation (interval: 1 year)"`.

If the cell method applies to a combination of axes, they may have a common original interval e.g. `cell_methods="lat: lon: standard_deviation (interval: 10 km)"`. Alternatively, they may have separate intervals, which are matched to the names of axes by position e.g. `cell_methods="lat: lon: standard_deviation (interval: 0.1 degree_N interval: 0.2 degree_E)"`, in which 0.1 degree applies to latitude and 0.2 degree to longitude.

If there is both standardized and non-standardized information, the non-standardized follows the standardized information and the keyword `comment:`. If there is no standardized information, the keyword `comment:` should be omitted. For instance, an area-weighted mean over latitude could be indicated as `lat: mean (area-weighted)` or `lat: mean (interval: 1 degree_north comment: area-`

weighted).

A dimension of size one may be the result of "collapsing" an axis by some statistical operation, for instance by calculating a variance from time series data. We strongly recommend that dimensions of size one be retained (or scalar coordinate variables be defined) to enable documentation of the method (through the cell_methods attribute) and its domain (through the cell_bounds attribute).

Example 7.5. Surface air temperature variance

The variance of the diurnal cycle on 1 January 1990 has been calculated from hourly instantaneous surface air temperature measurements. The time dimension of size one has been retained.

```
dimensions:
  lat=90;
  lon=180;
  time=1;
  nv=2;
variables:
  float TS_var(time,lat,lon);
    TS_var:long_name="surface air temperature variance"
    TS_var:units="K2";
    TS_var:cell_methods="time: variance (interval: 1 hr comment: sampled
instantaneously)";
  float time(time);
    time:units="days since 1990-01-01 00:00:00";
    time:bounds="time_bnds";
  float time_bnds(time,nv);
data:
  time=.5;
  time_bnds=0.,1.;
```

Notice that a parenthesized comment in the cell_methods attribute provides the nature of the samples used to calculate the variance.

7.3.3. Statistics applying to portions of cells

By default, the statistical method indicated by cell_methods is assumed to have been evaluated over the entire horizontal area of the cell. Sometimes, however, it is useful to limit consideration to only a portion of a cell (e.g. a mean over the sea-ice area). To indicate this, one of two conventions may be used.

The first convention is a method that can be used for the common case of a single area-type. In this case, the cell_methods attribute may include a string of the form "*name: method* where *type*". Here *name* could, for example, be area and *type* may be any of the strings permitted for a variable with a standard_name of area_type. As an example, if the method were mean and the area_type were sea_ice, then the data would represent a mean over only the sea ice portion of the grid cell. If the data writer expects *type* to be interpreted as one of the standard area_type strings, then none of the

variables in the netCDF file should be given a name identical to that of the string (because the second convention, described in the next paragraph, takes precedence).

The second convention is the more general. In this case, the `cell_methods` entry is of the form "*name: method* `where` *typevar*". Here *typevar* is a string-valued auxiliary coordinate variable or string-valued scalar coordinate variable (see Section 6.1, "Labels") with a `standard_name` of `area_type`. The variable *typevar* contains the name(s) of the selected portion(s) of the grid cell to which the *method* is applied. This convention can accommodate cases in which a method is applied to more than one area type and the result is stored in a single data variable (with a dimension which ranges across the various area types). It provides a convenient way to store output from land surface models, for example, since they deal with many area types within each surface gridbox (e.g., `vegetation`, `bare_ground`, `snow`, etc.).

Example 7.6. Mean surface temperature over land and sensible heat flux averaged separately over land and sea.

```
dimensions:
  lat=73;
  lon=96;
  maxlen=20;
  ls=2;
variables:
  float surface_temperature(lat,lon);
    surface_temperature:cell_methods="area: mean where land";
  float surface_upward_sensible_heat_flux(ls,lat,lon);
    surface_upward_sensible_heat_flux:coordinates="land_sea";
    surface_upward_sensible_heat_flux:cell_methods="area: mean where land_sea";
  char land_sea(ls,maxlen);
    land_sea:standard_name="area_type";
data:
  land_sea="land","sea";
```

If the *method* is `mean`, various ways of calculating the mean can be distinguished in the `cell_methods` attribute with a string of the form “mean where` *type1* [`over` *type2*]". Here, *type1* can be any of the possibilities allowed for *typevar* or *type* (as specified in the two paragraphs preceding above Example). The same options apply to *type2*, except it is not allowed to be the name of an auxiliary coordinate variable with a dimension greater than one (ignoring the dimension accommodating the maximum string length). A `cell_methods` attribute with a string of the form "`mean where` *type1* `over` *type2*" indicates the mean is calculated by summing over the *type1* portion of the cell and dividing by the area of the *type2* portion. In particular, a `cell_methods` string of the form "`mean where all_area_types over` *type2*" indicates the mean is calculated by summing over all types of area within the cell and dividing by the area of the *type2* portion. (Note that "`all_area_types” is one of the valid strings permitted for a variable with the `standard_name area_type`.) If "`over` *type2*" is omitted, the mean is calculated by summing over the *type1* portion of the cell and dividing by the area of this portion.

Example 7.7. Thickness of sea-ice and snow on sea-ice averaged over sea area.

```
variables:
  float sea_ice_thickness(lat,lon);
    sea_ice_thickness:cell_methods="area: mean where sea_ice over sea";
    sea_ice_thickness:standard_name="sea_ice_thickness";
    sea_ice_thickness:units="m";
  float snow_thickness(lat,lon);
    snow_thickness:cell_methods="area: mean where sea_ice over sea";
   snow_thickness:standard_name="lwe_thickness_of_surface_snow_amount";
    snow_thickness:units="m";
```

In the case of sea-ice thickness, the phrase “where sea_ice” could be replaced by “where all_area_types” without changing the meaning since the integral of sea-ice thickness over all area types is obviously the same as the integral over the sea-ice area only. In the case of snow thickness, “where sea_ice” differs from “where all_area_types” because “where sea_ice” excludes snow on land from the average.

7.3.4. Cell methods when there are no coordinates

To provide an indication that a particular cell method is relevant to the data without having to provide a precise description of the corresponding cell, the "*name*" that appears in a "*name*: *method*" pair may be an appropriate `standard_name` (which identifies the dimension) or the string, "*area*" (rather than the name of a scalar coordinate variable or a dimension with a coordinate variable). This convention cannot be used, however, if the name of a dimension or scalar coordinate variable is identical to *name*. There are two situations where this convention is useful.

First, it allows one to provide some indication of the method when the cell coordinate range cannot be precisely defined. For example, a climatological mean might be based on any data that exists, and, in general, the data might not be available over the same time periods everywhere. In this case, the time range would not be well defined (because it would vary, depending on location), and it could not be precisely specified through a time dimension’s bounds. Nevertheless, useful information can be conveyed by a `cell_methods` entry of "`time: mean`" (where `time`, it should be noted, is a valid `standard_name`). (As required by this convention, it is assumed here that for the data referred to by this `cell_methods` attribute, "time" is not a dimension or coordinate variable.)

Second, for a few special dimensions, this convention allows one to indicate (without explicitly defining the coordinates) that the method applies to the domain covering the entire permitted range of those dimensions. This is allowed only for longitude, latitude, and area (indicating a combination of horizontal coordinates). For longitude, the domain is indicated according to this provision by the string "longitude" (rather than the name of a longitude coordinate variable), and this implies that the method applies to all possible longitudes (i.e., from 0E to 360E). For latitude, the string "latitude" is used and implies the method applies to all possible latitudes (i.e., from 90S to 90N). For area, the string "area" is used and implies the method applies to the whole world.

In the second case if, in addition, the data variable has a dimension with a corresponding labeled axis that specifies a geographic region (Section 6.1.1, "Geographic Regions"), the implied range of

longitude and latitude is the valid range for each specified region, or in the case of `area` the domain is the geographic region. For example, there could be a `cell_methods` entry of "`longitude: mean`", where `longitude` is *not* the name of a dimension or coordinate variable (but is one of the special cases given above). That would indicate a mean over all longitudes. Note, however, that if in addition the data variable had a scalar coordinate variable with a `standard_name` of `region` and a value of `atlantic_ocean`, it would indicate a mean over longitudes that lie within the Atlantic Ocean, not all longitudes.

We recommend that whenever possible, cell bounds should be supplied by giving the variable a dimension of size one and attaching bounds to the associated coordinate variable.

7.4. Climatological Statistics

Climatological statistics may be derived from corresponding portions of the annual cycle in a set of years, e.g., the average January temperatures in the climatology of 1961-1990, where the values are derived by averaging the 30 Januarys from the separate years. Portions of the climatological cycle are specified by references to dates within the calendar year. However, a calendar year is not a well-defined unit of time, because it differs between leap years and other years, and among calendars. Nonetheless for practical purposes we wish to compare statistics for months or seasons from different calendars, and to make climatologies from a mixture of leap years and other years. Hence we provide special conventions for indicating dates within the climatological year. Climatological statistics may also be derived from corresponding portions of a range of days, for instance the average temperature for each hour of the average day in April 1997. In addition the two concepts may be used at once, for instance to indicate not April 1997, but the average April of the five years 1995-1999.

Climatological variables have a climatological time axis. Like an ordinary time axis, a climatological time axis may have a dimension of unity (for example, a variable containing the January average temperatures for 1961-1990), but often it will have several elements (for example, a climatological time axis with a dimension of 12 for the climatological average temperatures in each month for 1961-1990, a dimension of 3 for the January mean temperatures for the three decades 1961-1970, 1971-1980, 1981-1990, or a dimension of 24 for the hours of an average day). Intervals of climatological time are conceptually different from ordinary time intervals; a given interval of climatological time represents a set of subintervals which are not necessarily contiguous. To indicate this difference, a climatological time coordinate variable does not have a `bounds` attribute. Instead, it has a `climatology` attribute, which names a variable with dimensions (n,2), n being the dimension of the climatological time axis. Using the units and calendar of the time coordinate variable, element (i,0) of the climatology variable specifies the beginning of the first subinterval and element (i,1) the end of the last subinterval used to evaluate the climatological statistics with index i in the time dimension. The time coordinates should be values that are representative of the climatological time intervals, such that an application which does not recognise climatological time will nonetheless be able to make a reasonable interpretation.

The COARDS standard offers limited support for climatological time. For compatibility with COARDS, time coordinates should also be recognised as climatological if they have a `units` attribute of time-units relative to midnight on 1 January in year 0 i.e. `since 0-1-1` in udunits syntax, and provided they refer to the real-world calendar. We do not recommend this convention because (a) it does not provide any information about the intervals used to compute the climatology, and (b)

there is no standard for how dates since year 1 will be encoded with units having a reference time in year 0, since this year does not exist; consequently there may be inconsistencies among software packages in the interpretation of the time coordinates. Year 0 may be a valid year in non-real-world calendars, and therefore cannot be used to signal climatological time in such cases.

A climatological axis may use different statistical methods to represent variation among years, within years and within days. For example, the average January temperature in a climatology is obtained by averaging both within years and over years. This is different from the average January-maximum temperature and the maximum January-average temperature. For the former, we first calculate the maximum temperature in each January, then average these maxima; for the latter, we first calculate the average temperature in each January, then find the largest one. As usual, the statistical operations are recorded in the `cell_methods` attribute, which may have two or three entries for the climatological time dimension.

Valid values of the `cell_methods` attribute must be in one of the forms from the following list. The intervals over which various statistical methods are applied are determined by decomposing the date and time specifications of the climatological time bounds of a cell, as recorded in the variable named by the `climatology` attribute. (The date and time specifications must be calculated from the time coordinates expressed in units of "time interval since reference date and time".) In the descriptions that follow we use the abbreviations *y*, *m*, *d*, *H*, *M*, and *S* for year, month, day, hour, minute, and second respectively. The suffix *0* indicates the earlier bound and *1* the latter.

time: method1 `within years` *time: method2* `over years`

> *method1* is applied to the time intervals (mdHMS0-mdHMS1) within individual years and *method2* is applied over the range of years (y0-y1).

time: method1 `within days` *time: method2* `over days`

> *method1* is applied to the time intervals (HMS0-HMS1) within individual days and *method2* is applied over the days in the interval (ymd0-ymd1).

time: method1 `within days` *time: method2* `over days` *time: method3* `over years`

> *method1* is applied to the time intervals (HMS0-HMS1) within individual days and *method2* is applied over the days in the interval (md0-md1), and *method3* is applied over the range of years (y0-y1).

The methods which can be specified are those listed in Appendix E, Cell Methods and each entry in the `cell_methods` attribute may also, as usual, contain non-standardised information in parentheses after the method. For instance, a mean over ENSO years might be indicated by "`time: mean over years (ENSO years)`".

When considering intervals within years, if the earlier climatological time bound is later in the year than the later climatological time bound, it implies that the time intervals for the individual years run from each year across January 1 into the next year e.g. DJF intervals run from December 1 0:00 to March 1 0:00. Analogous situations arise for daily intervals running across midnight from one day to the next.

When considering intervals within days, if the earlier time of day is equal to the later time of day, then the method is applied to a full 24 hour day.

We have tried to make the examples in this section easier to understand by translating all time coordinate values to date and time formats. This is not currently valid CDL syntax.

Example 7.8. Climatological seasons

This example shows the metadata for the average seasonal-minimum temperature for the four standard climatological seasons MAM JJA SON DJF, made from data for March 1960 to February 1991.

```
dimensions:
  time=4;
  nv=2;
variables:
  float temperature(time,lat,lon);
    temperature:long_name="surface air temperature";
    temperature:cell_methods="time: minimum within years time: mean over years";
    temperature:units="K";
  double time(time);
    time:climatology="climatology_bounds";
    time:units="days since 1960-1-1";
  double climatology_bounds(time,nv);
data:  // time coordinates translated to date/time format
  time="1960-4-16", "1960-7-16", "1960-10-16", "1961-1-16" ;
  climatology_bounds="1960-3-1",  "1990-6-1",
                     "1960-6-1",  "1990-9-1",
                     "1960-9-1",  "1990-12-1",
                     "1960-12-1", "1991-3-1" ;
```

Example 7.9. Decadal averages for January

Average January precipitation totals are given for each of the decades 1961-1970, 1971-1980, 1981-1990.

```
dimensions:
  time=3;
  nv=2;
variables:
  float precipitation(time,lat,lon);
    precipitation:long_name="precipitation amount";
    precipitation:cell_methods="time: sum within years time: mean over years";
    precipitation:units="kg m-2";
  double time(time);
    time:climatology="climatology_bounds";
    time:units="days since 1901-1-1";
  double climatology_bounds(time,nv);
data:  // time coordinates translated to date/time format
  time="1965-1-15", "1975-1-15", "1985-1-15" ;
  climatology_bounds="1961-1-1", "1970-2-1",
                     "1971-1-1", "1980-2-1",
                     "1981-1-1", "1990-2-1" ;
```

Example 7.10. Temperature for each hour of the average day

Hourly average temperatures are given for April 1997.

```
dimensions:
  time=24;
  nv=2;
variables:
  float temperature(time,lat,lon);
    temperature:long_name="surface air temperature";
    temperature:cell_methods="time: mean within days time: mean over days";
    temperature:units="K";
  double time(time);
    time:climatology="climatology_bounds";
    time:units="hours since 1997-4-1";
  double climatology_bounds(time,nv);
data:  // time coordinates translated to date/time format
  time="1997-4-1 0:30", "1997-4-1 1:30", ... "1997-4-1 23:30" ;
  climatology_bounds="1997-4-1 0:00",  "1997-4-30 1:00",
                     "1997-4-1 1:00",  "1997-4-30 2:00",
                      ...
                      "1997-4-1 23:00", "1997-5-1 0:00" ;
```

Example 7.11. Extreme statistics and spell-lengths

Number of frost days during NH winter 2007-2008, and maximum length of spells of consecutive frost days. A "frost day" is defined as one during which the minimum temperature falls below freezing point (0 degC). This is described as a climatological statistic, in which the minimum temperature is first calculated within each day, and then the number of days or spell lengths meeting the specified condition are evaluated. In this operation, the standard name is also changed; the original data are `air_temperature`.

```
variables:
  float n1(lat,lon);
    n1:standard_name="number_of_days_with_air_temperature_below_threshold";
    n1:coordinates="threshold time";
    n1:cell_methods="time: minimum within days time: sum over days";
  float n2(lat,lon);
    n2:standard_name="spell_length_of_days_with_air_temperature_below_threshold";
    n2:coordinates="threshold time";
    n2:cell_methods="time: minimum within days time: maximum over days";
  float threshold;
    threshold:standard_name="air_temperature";
    threshold:units="degC";
  double time;
    time:climatology="climatology_bounds";
    time:units="days since 2000-6-1";
  double climatology_bounds(time,nv);
data: // time coordinates translated to date/time format
  time="2008-1-16 6:00";
  climatology_bounds="2007-12-1 6:00", "2000-8-2 6:00";
  threshold=0.;
```

Example 7.12. Temperature for each hour of the typical climatological day

This is a modified version of the previous example, "Temperature for each hour of the average day". It now applies to April from a 1961-1990 climatology.

```
variables:
  float temperature(time,lat,lon);
    temperature:long_name="surface air temperature";
    temperature:cell_methods="time: mean within days ",
      "time: mean over days time: mean over years";
    temperature:units="K";
  double time(time);
    time:climatology="climatology_bounds";
    time:units="days since 1961-1-1";
  double climatology_bounds(time,nv);
data:  // time coordinates translated to date/time format
  time="1961-4-1 0:30", "1961-4-1 1:30", ..., "1961-4-1 23:30" ;
  climatology_bounds="1961-4-1 0:00", "1990-4-30 1:00",
                     "1961-4-1 1:00", "1990-4-30 2:00",
                     ...
                     "1961-4-1 23:00", "1990-5-1 0:00" ;
```

Example 7.13. Monthly-maximum daily precipitation totals

Maximum of daily precipitation amounts for each of the three months June, July and August 2000 are given. The first daily total applies to 6 a.m. on 1 June to 6 a.m. on 2 June, the 30th from 6 a.m. on 30 June to 6 a.m. on 1 July. The maximum of these 30 values is stored under time index 0 in the precipitation array.

```
dimensions:
  time=3;
  nv=2;
variables:
  float precipitation(time,lat,lon);
    precipitation:long_name="Accumulated precipitation";
    precipitation:cell_methods="time: sum within days time: maximum over days";
    precipitation:units="kg";
  double time(time);
    time:climatology="climatology_bounds";
    time:units="days since 2000-6-1";
  double climatology_bounds(time,nv);
data:  // time coordinates translated to date/time format
  time="2000-6-16", "2000-7-16", "2000-8-16" ;
  climatology_bounds="2000-6-1 6:00:00", "2000-7-1 6:00:00",
                     "2000-7-1 6:00:00", "2000-8-1 6:00:00",
                     "2000-8-1 6:00:00", "2000-9-1 6:00:00" ;
```

Chapter 8. Reduction of Dataset Size

There are two methods for reducing dataset size: packing and compression. By packing we mean altering the data in a way that reduces its precision. By compression we mean techniques that store the data more efficiently and result in no precision loss. Compression only works in certain circumstances, e.g., when a variable contains a significant amount of missing or repeated data values. In this case it is possible to make use of standard utilities, e.g., UNIX `compress` or GNU `gzip` , to compress the entire file after it has been written. In this section we offer an alternative compression method that is applied on a variable by variable basis. This has the advantage that only one variable need be uncompressed at a given time. The disadvantage is that generic utilities that don't recognize the CF conventions will not be able to operate on compressed variables.

8.1. Packed Data

At the current time the netCDF interface does not provide for packing data. However a simple packing may be achieved through the use of the optional NUG defined attributes `scale_factor` and `add_offset` . After the data values of a variable have been read, they are to be multiplied by the `scale_factor` , and have `add_offset` added to them. If both attributes are present, the data are scaled before the offset is added. When scaled data are written, the application should first subtract the offset and then divide by the scale factor. The units of a variable should be representative of the unpacked data.

This standard is more restrictive than the NUG with respect to the use of the `scale_factor` and `add_offset` attributes; ambiguities and precision problems related to data type conversions are resolved by these restrictions. If the `scale_factor` and `add_offset` attributes are of the same data type as the associated variable, the unpacked data is assumed to be of the same data type as the packed data. However, if the `scale_factor` and `add_offset` attributes are of a different data type from the variable (containing the packed data) then the unpacked data should match the type of these attributes, which must both be of type `float` or both be of type `double` . An additional restriction in this case is that the variable containing the packed data must be of type `byte` , `short` or `int` . It is not advised to unpack an `int` into a `float` as there is a potential precision loss.

When data to be packed contains missing values the attributes that indicate missing values (`_FillValue` , `valid_min` , `valid_max` , `valid_range`) must be of the same data type as the packed data. See Section 2.5.1, "Missing Data" for a discussion of how applications should treat variables that have attributes indicating both missing values and transformations defined by a scale and/or offset.

8.2. Compression by Gathering

To save space in the netCDF file, it may be desirable to eliminate points from data arrays that are invariably missing. Such a compression can operate over one or more adjacent axes, and is accomplished with reference to a list of the points to be stored. The list is constructed by considering a mask array that only includes the axes to be compressed, and then mapping this array onto one dimension without reordering. The list is the set of indices in this one-dimensional mask of the required points. In the compressed array, the axes to be compressed are all replaced by a single axis, whose dimension is the number of wanted points. The wanted points appear along this dimension in the same order they appear in the uncompressed array, with the unwanted points

skipped over. Compression and uncompression are executed by looping over the list.

The list is stored as the coordinate variable for the compressed axis of the data array. Thus, the list variable and its dimension have the same name. The list variable has a string attribute `compress` , *containing a blank-separated list of the dimensions which were affected by the compression in the order of the CDL declaration of the uncompressed array* . The presence of this attribute identifies the list variable as such. The list, the original dimensions and coordinate variables (including boundary variables), and the compressed variables with all the attributes of the uncompressed variables are written to the netCDF file. The uncompressed variables can be reconstituted exactly as they were using this information.

Example 8.1. Horizontal compression of a three-dimensional array

We eliminate sea points at all depths in a longitude-latitude-depth array of soil temperatures. In this case, only the longitude and latitude axes would be affected by the compression. We construct a list `landpoint(landpoint)` containing the indices of land points.

```
dimensions:
  lat=73;
  lon=96;
  landpoint=2381;
  depth=4;
variables:
  int landpoint(landpoint);
    landpoint:compress="lat lon";
  float landsoilt(depth,landpoint);
    landsoilt:long_name="soil temperature";
    landsoilt:units="K";
  float depth(depth);
  float lat(lat);
  float lon(lon);
data:
  landpoint=363, 364, 365, ...;
```

Since `landpoint(0)=363` , for instance, we know that `landsoilt(*,0)` maps on to point 363 of the original data with dimensions `(lat,lon)` . This corresponds to indices `(3,75)` , i.e., `363 = 3*96 + 75` .

Example 8.2. Compression of a three-dimensional field

We compress a longitude-latitude-depth field of ocean salinity by eliminating points below the sea-floor. In this case, all three dimensions are affected by the compression, since there are successively fewer active ocean points at increasing depths.

```
variables:
  float salinity(time,oceanpoint);
  int oceanpoint(oceanpoint);
    oceanpoint:compress="depth lat lon";
  float depth(depth);
  float lat(lat);
  float lon(lon);
  double time(time);
```

This information implies that the salinity field should be uncompressed to an array with dimensions `(depth,lat,lon)` .

Chapter 9. Discrete Sampling Geometries

This chapter provides representations for **discrete sampling geometries** , such as time series, vertical profiles and trajectories. Discrete sampling geometry datasets are characterized by a dimensionality that is lower than that of the space-time region that is sampled; discrete sampling geometries are typically “paths” through space-time.

9.1. Features and feature types

Each type of discrete sampling geometry (point, time series, profile or trajectory) is defined by the relationships among its spatiotemporal coordinates. We refer to the type of discrete sampling geometry as its **featureType** . The term “ **feature** ” refers herein to a single instance of the **discrete sampling geometry** (such as a single time series). The representation of such features in a CF dataset was supported previous to the introduction of this chapter using a particular convention, which is still supported (that described by section 9.3.1). This chapter describes further conventions which offer advantages of efficiency and clarity for storing a collection of features in a single file. When using these new conventions, *the features contained within a collection must always be of the same type; and all the collections in a CF file must be of the same feature type* . (Future versions of CF may allow mixing of multiple feature types within a file.) Table 9.1 presents the feature types covered by this chapter.

featureType	**Description of a single feature with this discrete sampling geometry**	
	Form of a data variable containing values defined on a collection of these features	**Mandatory space-time coordinates for a collection of these features**
point	a single data point (having no implied coordinate relationship to other points)	
	data(i)	x(i) y(i) t(i)
timeSeries	a series of data points at the same spatial location with monotonically increasing times	
	data(i,o)	x(i) y(i) t(i,o)
trajectory	a series of data points along a path through space with monotonically increasing times	
	data(i,o)	x(i,o) y(i,o) t(i,o)
profile	an ordered set of data points along a vertical line at a fixed horizontal position and fixed time	
	data(i,o)	x(i) y(i) z(i,o) t(i)
timeSeriesProfile	a series of profile features at the same horizontal position with monotonically increasing times	
	data(i,p,o)	x(i) y(i) z(i,p,o) t(i,p)

featureType	**Description of a single feature with this discrete sampling geometry**	
trajectoryProfile	a series of profile features located at points ordered along a trajectory	
	data(i,p,o)	x(i,p) y(i,p) z(i,p,o) t(i,p)

Table 9.1. Logical structure and mandatory coordinates for discrete sampling geometry featureTypes.

In Table 9.1 the spatial coordinates x and y typically refer to longitude and latitude but other horizontal coordinates could also be used (see sections 4 and 5.6). The spatial coordinate z refers to vertical position. The time coordinate is indicated as t. The space-time coordinates that are indicated for each feature are mandatory. However a featureType may also include other space-time coordinates which are not mandatory (notably the z coordinate). The array subscripts that are shown illustrate only the *logical* structure of the data. The subscripts found in actual CF files are determined by the specific type of representations (see section 9.3).

The designation of dimensions as mandatory precludes the encoding of data variables where geo-positioning cannot be described as a discrete point location. Problematic examples include:

- time series that refer to a geographical region (e.g. the northern hemisphere), a volume (e.g. the troposphere), or a geophysical quantity in which geolocation information is inherent (e.g. the Southern Oscillation Index (SOI) is the difference between values at two point locations);
- vertical profiles that similarly represent geographically area-averaged values; and
- paths in space that indicate a geographically located feature, but lack a suitable time coordinate (e.g. a meteorological front).

Future versions of CF will generalize the concepts of geolocation to encompass these cases. As of CF version 1.6 such data can be stored using the representations that are documented here by two means: 1) by utilizing the orthogonal multidimensional array representation and omitting the featureType attribute; or 2) by assigning arbitrary coordinates to the mandatory dimensions. For example a globally-averaged latitude position (90s to 90n) could be represented arbitrarily (and poorly) as a latitude position at the equator.

9.2. Collections, instances and elements

In Table 9.1 the dimension with subscript i identifies a particular feature within a collection of features. It is called the **instance dimension** . One-dimensional variables in a Discrete Geometry CF file, which have *only* this dimension (such as x(i) y(i) and z(i) for a timeseries), are **instance variables** . Instance variables provide the metadata that differentiates individual features.

The subscripts o and p distinguish the data elements that compose a single feature. For example in a collection of **timeSeries** features, each time series instance, i, has data values at various times, o. In a collection of **profile** features, the subscript, o, provides the index position along the vertical axis of each profile instance. We refer to data values in a feature as its **elements** , and to the dimensions of o and p as **element dimensions** . Each feature can have its own set of element subscripts o and p. For instance, in a collection of timeSeries features, each individual timeSeries

can have its own set of times. The notation t(i,o) means there is a set of times with subscripts o for the elements of each feature i. Feature instances within a collection need not have the same numbers of elements. If the features do all have the same number of elements, and the sequence of element coordinates is identical for all features, savings in simplicity and space are achievable by storing only one copy of these coordinates. This is the essence of the orthogonal multidimensional representation (see section 9.3.1).

If there is only a single feature to be stored in a data variable, there is no need for an instance dimension and it is permitted to omit it. The data will then be one-dimensional, which is a special (degenerate) case of the multidimensional array representation. The instance variables will be scalar coordinate variables; the data variable and other auxiliary coordinate variables will have only an element dimension and not have an instance dimension, e.g. data(o) and t(o) for a single timeSeries.

9.3. Representations of collections of features in data variables

The individual features within a collection need not necessarily contain the same number of elements. For instance observed in situ time series will commonly contain unique numbers of time points, reflecting different deployment dates of the instruments. Other data sources, such as the output of numerical models, may commonly generate features of identical size. CF offers multiple representations to allow the storage to be optimized for the character of the data. Four types of representation are utilized in this chapter:

- two **multidimensional array representations** , in which each feature instance is allocated the identical amount of storage space. In these representations the instance dimension and the element dimension(s) are distinct CF coordinate axes (typical of coordinate axes discussed in chapter 4); and
- two **ragged array representations** , in which each feature is provided with the minimum amount of space that it requires. In these representations the instances of the individual features are stacked sequentially along the same array dimension as the elements of the features; we refer to this combined dimension as the **sample dimension** .

In the multidimensional array representations, data variables have both an instance dimension and an element dimension. The dimensions may be given in any order. If there is a need for either the instance or an element dimension to be the netCDF unlimited dimension (so that more features or more elements can be appended), then that dimension must be the outer dimension of the data variable i.e. the leading dimension in CDL.

In the ragged array representations, the instance dimension (`i`), which sequences the individual features within the collection, and the element dimension, which sequences the data elements of each feature (`o` and `p`), both occupy the same dimension (the sample dimension). If the sample dimension is the netCDF unlimited dimension, new data can be appended to the file.

In all representations, the instance dimension (which is also the sample dimension in ragged representations) may be set initially to a size that is arbitrarily larger than what is required for the features which are available at the time that the file is created. Allocating unused array space in this way (pre-filled with missing values — see also section 9.6, *Missing data*), can be useful as a

means to reserve space that will be available to add features at a later time.

9.3.1. Orthogonal multidimensional array representation

The **orthogonal multidimensional array representation** , the simplest representation, can be used if each feature instance in the collection has identical coordinates along the element axis of the features. For example, for a collection of the timeSeries that share a common set of times, or a collection of profiles that share a common set of vertical levels, this is likely to be the natural representation to use. In both examples, there will be longitude and latitude coordinate variables, x(i), y(i), that are one-dimensional and defined along the instance dimension.

Table 9.2 illustrates the storage of a data variable using the orthogonal multidimensional array representation. The data variable holds a collection of 4 features. The individual features, distinguished by color, are sequenced along the horizontal axis by the instance dimension indices, i1, i2, i3, i4. Each instance contains three elements, sequenced along the vertical with element dimension indices, o1, o2, o3. The i and o subscripts would be interchanged (i.e. Table 9.2 would be transposed) if the element dimension were the netCDF unlimited dimension.

(i1, o1)	(i2, o1)	(i3, o1)	(i4, o1)
(i1, o2)	(i2, o2)	(i3, o2)	(i4, o2)
(i1, o3)	(i2, o3)	(i3, o3)	(i4, o3)

Table 9.2 The storage of a data variable using the orthogonal multidimensional array representation (subscripts in CDL order).

The instance variables of a dataset corresponding to Table 9.2 will be one-dimensional with size 4 (for example, the latitude locations of timeSeries),

lat(i1)	lat(i2)	lat(i3)	lat(i4)

and the element coordinate axis will be one-dimensional with size 3 (for example, the time

time(o1)
time(o2)
time(o3)
time(o4)

coordinates that are shared by all of the timeSeries). This representation is consistent with the multidimensional fields described in chapter 5; the characteristic that makes it atypical from chapter 5 (though not incompatible) is that the instance dimension is a discrete axis (see section 4.5).

9.3.2. Incomplete multidimensional array representation

The **incomplete multidimensional array representation** can used if the features within a collection do not all have the same number of elements, but sufficient storage space is available to

allocate the number of elements required by the longest feature to all features. That is, features that are shorter than the longest feature must be padded with missing values to bring all instances to the same storage size. This representation sacrifices storage space to achieve simplicity for reading and writing.

Table 9.3 illustrates the storage of a data variable using the orthogonal multidimensional array representation. The data variable holds a collection of 4 features. The individual features, distinguished by color, are sequenced by the instance dimension indices, i1, i2, i3, i4. The instances contain respectively 2, 4, 3 and 6 elements, sequenced by the element dimension index with values of o1, o2, o3, …. The i and o subscripts would be interchanged (i.e. Table 9.3 would be transposed) if the element dimension were the netCDF unlimited dimension.

(i1, o1)	(i2, o1)	(i3, o1)	(i4, o1)
(i1, o2)	(i2, o2)	(i3, o2)	(i4, o2)
	(i2, o3)	(i3, o3)	(i4, o3)
	(i2, o4)		(i4, o4)
			(i4, o5)
			(i4, o6)

Table 9.3. The storage of data using the incomplete multidimensional array representation (subscripts in CDL order).

9.3.3. Contiguous ragged array representation

The **contiguous ragged array representation** can be used only if the size of each feature is known at the time that it is created. In this representation the data for each feature will be contiguous on disk, as shown in Table 9.4.

(i1, o1)
(i1, o2)
(i2, o1)
(i2, o2)
(i2, o3)
(i2, o4)
(i3, o1)
(i3, o2)
(i3, o3)
(i4, o1)
(i4, o2)
(i4, o3)

(i4, o4)
(i4, o5)
(i4, o6)

Table 9.4. The storage of data using the contiguous ragged representation (subscripts in CDL order).

In this representation, the file contains a **count variable** , which must be of type integer and

count(i1)	count(i2)	count(i3)	count(i4)
2	4	3	6

must have the instance dimension as its sole dimension. The count variable contains the number of elements that each feature has. This representation and its count variable are identifiable by the presence of an attribute, `sample_dimension` , found on the count variable, which names the sample dimension being counted. For indices that correspond to features, whose data have not yet been written, the count variable should have a value of zero or a missing value.

9.3.4. Indexed ragged array representation

The **indexed ragged array representation** stores the features interleaved along the sample dimension in the data variable as shown in Table 9.4. The canonical use case for this representation is the storage of real-time data streams that contain reports from many sources; the data can be written as it arrives.

(i1, o1)		0
(i2, o1)		1
(i3, o1)		2
(i4, o1)		3
(i4, o2)		3
(i2, o2)		1
(i4, o3)		3
(i4, o4)		3
(i1, o2)		0
(i2, o3)		1
(i3, o2)		2
(i4, o5)		3
(i3, o3)		2
(i2, o4)		1
(i4, o6)		3

Table 9.4 The storage of data using the indexed ragged representation (subscripts in CDL order). The left hand side of the table illustrates a data variable; the right hand side of the table contains the values of the index variable.

In this representation, the file contains an **index variable** , which must be of type integer, and must have the sample dimension as its single dimension. The index variable contains the zero-based index of the feature to which each element belongs. This representation is identifiable by the presence of an attribute, `instance_dimension` , on the index variable, which names the dimension of the instance variables. For those indices of the sample dimension, into which data have not yet been written, the index variable should be pre-filled with missing values.

9.4. The featureType attribute

A global attribute, **featureType** , is required for all Discrete Geometry representations except the orthogonal multidimensional array representation, for which it is highly recommended. The exception is allowed for backwards compatibility, as discussed in 9.3.1. A Discrete Geometry file may include arbitrary numbers of data variables, but (as of CF v1.6) all of the data variables contained in a single file must be of the single feature type indicated by the global `featureType` attribute, if it is present.1 The value assigned to the `featureType` attribute is case-insensitive; it must be one of the string values listed in the left column of Table 9.1.

9.5. Coordinates and metadata

Every feature within a Discrete Geometry CF file must be unambiguously associated with an extensible collection of instance variables that identify the feature and provide other metadata as needed to describe it. Every element of every feature must be unambiguously associated with its space and time coordinates and with the feature that contains it. The `coordinates` attribute must be attached to every data variable to indicate the spatiotemporal coordinate variables that are needed to geo-locate the data.

Where feasible a variable with the attribute **cf_role** should be included. The only acceptable values of cf_role for Discrete Geometry CF data sets are `timeseries_id` , `profile_id` , and `trajectory_id` . The variable carrying the cf_role attribute may have any data type. When a variable is assigned this attribute, it must provide a unique identifier for each feature instance. CF files that contain timeSeries, profile or trajectory featureTypes, should include only a single occurrence of a `cf_role` attribute; CF files that contain timeSeriesProfile or trajectoryProfile may contain two occurrences, corresponding to the two levels of structure in these feature types.

It is not uncommon for observational data to have two sets of coordinates for particular coordinate axes of a feature: a nominal point location and a more precise location that varies with the elements in the feature. For example, although an idealized vertical profile is measured at a fixed horizontal position and time, a realistic representation might include the time variations and horizontal drift that occur during the duration of the sampling. Similarly, although an idealized time series exists at a fixed lat-long position, a realistic representation of a moored ocean time series might include the “watch cycle” excursions of horizontal position that occur as a result of tidal currents.

CF Discrete Geometries provides a mechanism to encode both the nominal and the precise

positions, while retaining the semantics of the idealized feature type. Only the set of coordinates which are regarded as the nominal (default or preferred) positions should be indicated by the attribute `axis` , which should be assigned string values to indicate the orientations of the axes (`X` , `Y` , `Z` , or `T`). See example A9.2.3.2. Auxiliary coordinate variables containing the nominal and the precise positions should be listed in the relevant `coordinates` attributes of data variables. In orthogonal representations the nominal positions could be coordinate variables, which do not need to be listed in the `coordinates` attribute, rather than auxiliary coordinate variables.

Coordinate bounds may optionally be associated with coordinate variables and auxiliary coordinate variables using the bounds attribute, following the conventions described in section 7.1. Coordinate bounds are especially important for accurate representations of model output data using discrete geometry representations; they record the boundaries of the model grid cells.

If there is a vertical coordinate variable or auxiliary coordinate variable, it must be identified by the means specified in section 4.3. The use of the attribute `axis=Z` is recommended for clarity. A `standard_name` attribute (see section 3.3) that identifies the vertical coordinate is recommended, e.g. "altitude", "height", etc. . (See the CF Standard Name Table).

9.6. Missing Data

Auxiliary coordinate variables (spatial and time) must contain missing values to indicate a void in data storage in the file but must not have missing data for any other reason. This situation may arise for unused elements in the incomplete multidimensional array representation, and in any representation if the instance dimension is set to a larger size than the number of features currently stored. It is not permitted for auxiliary coordinate variables to have missing values for elements where there is non-missing data. Where *any* auxiliary coordinate variable contains a missing value, *all* other coordinate, auxiliary coordinate and data values corresponding to that element should *also* contain missing values. Data variables should (as usual) also contain missing values to indicate when there is no valid data available for the element, although the coordinates are valid.

Similarly, for indices where the instance variable identified by `cf_role` contains a missing value indicator, all other instance variable should also contain missing values.

Appendix A: Attributes

All CF attributes are listed here except for those that are used to describe grid mappings. See Appendix F for the grid mapping attributes.

The "Type" values are **S** for string, **N** for numeric, and **D** for the type of the data variable. The "Use" values are **G** for global, **C** for variables containing coordinate data, and **D** for variables containing non-coordinate data. "Links" indicates the location of the attribute"s original definition (first link) and sections where the attribute is discussed in this document (additional links as necessary).

Table A.1. Attributes

Attribute	Type	Use	Links	Description
`add_offset`	N	D	NUG appendix B, Section 8.1, "Packed Data"	If present for a variable, this number is to be added to the data after it is read by an application. If both `scale_factor` and `add_offset` attributes are present, the data are first scaled before the offset is added.
`ancillary_variables`	S	D	Section 3.4, "Ancillary Data"	Identifies a variable that contains closely associated data, e.g., the measurement uncertainties of instrument data.
`axis`	S	C	Chapter 4, Coordinate Types	Identifies latitude, longitude, vertical, or time axes.
`bounds`	S	C	Section 7.1, "Cell Boundaries"	Identifies a boundary variable.
`calendar`	S	C	Section 4.4.1, "Calendar"	Calendar used for encoding time axes.
`cell_measures`	S	D	Section 7.2, "Cell Measures"	Identifies variables that contain cell areas or volumes.
`cell_methods`	S	D	Section 7.3, "Cell Methods", Section 7.4, "Climatological Statistics"	Records the method used to derive data that represents cell values.
`cf_role`	C	C	Section 9.5, "Coordinates and metadata"	Identifies the roles of variables that identify features in discrete sampling geometries
`climatology`	S	C	Section 7.4, "Climatological Statistics"	Identifies a climatology variable.
`comment`	S	G, D	Section 2.6.2, "Description of file contents"	Miscellaneous information about the data or methods used to produce it.

Attribute	Type	Use	Links	Description
compress	S	C	Section 8.2, "Compression by Gathering", Section 5.3, "Reduced Horizontal Grid"	Records dimensions which have been compressed by gathering.
Conventions	S	G	NUG appendix B	Name of the conventions followed by the dataset.
coordinates	S	D	Chapter 5, Coordinate Systems, Section 6.1, "Labels", Section 6.2, "Alternative Coordinates"	Identifies auxiliary coordinate variables, label variables, and alternate coordinate variables.
_FillValue	D	C, D	NUG appendix B	A value used to represent missing or undefined data. Not allowed for coordinate data except in the case of auxiliary coordinate varibles in discrete sampling geometries.
featureType	C	G	Section 9.4, "The featureType attribute"	Specifies the type of discrete sampling geometry to which the data in the file belongs, and implies that all data variables in the file contain collections of features of that type.
flag_masks	D	D	Section 3.5, "Flags"	Provides a list of bit fields expressing Boolean or enumerated flags.
flag_meanings	S	D	Section 3.5, "Flags"	Use in conjunction with flag_values to provide descriptive words or phrases for each flag value. If multi-word phrases are used to describe the flag values, then the words within a phrase should be connected with underscores.
flag_values	D	D	Section 3.5, "Flags"	Provides a list of the flag values. Use in conjunction with flag_meanings.
formula_terms	S	C	Section 4.3.2, "Dimensionless Vertical Coordinate"	Identifies variables that correspond to the terms in a formula.
grid_mapping	S	D	Section 5.6, "Horizontal Coordinate Reference Systems, Grid Mappings, and Projections"	Identifies a variable that defines a grid mapping.
history	S	G	NUG appendix B	List of the applications that have modified the original data.

Attribute	Type	Use	Links	Description
instance_dimension	N	D	Section 9.3, "Representations of collections of features in data variables"	An attribute which identifies an index variable and names the instance dimension to which it applies. The index variable indicates that the indexed ragged array representation is being used for a collection of features.
institution	S	G, D	Section 2.6.2, "Description of file contents"	Where the original data was produced.
leap_month	N	C	Section 4.4.1, "Calendar"	Specifies which month is lengthened by a day in leap years for a user defined calendar.
leap_year	N	C	Section 4.4.1, "Calendar"	Provides an example of a leap year for a user defined calendar. It is assumed that all years that differ from this year by a multiple of four are also leap years.
long_name	S	C, D	NUG appendix B, Section 3.2, "Long Name"	A descriptive name that indicates a variable"s content. This name is not standardized.
missing_value	D	C, D	Section 2.5.1, "Missing Data"	A value used to represent missing or undefined data (deprecated by the NUG). Not allowed for coordinate data except in the case of auxiliary coordinate variables in discrete sampling geometries.
month_lengths	N	C	Section 4.4.1, "Calendar"	Specifies the length of each month in a non-leap year for a user defined calendar.
positive	S	C	[COARDS]	Direction of increasing vertical coordinate value.
references	S	G, D	Section 2.6.2, "Description of file contents"	References that describe the data or methods used to produce it.
sample_dimension	N	D	Section 9.3, "Representations of collections of features in data variables"	An attribute which identifies a count variable and names the sample dimension to which it applies. The count variable indicates that the contiguous ragged array representation is being used for a collection of features.

Attribute	Type	Use	Links	Description
scale_factor	N	D	NUG appendix B, Section 8.1, "Packed Data"	If present for a variable, the data are to be multiplied by this factor after the data are read by an application See also the add_offset attribute.
source	S	G, D	Section 2.6.2, "Description of file contents"	Method of production of the original data.
standard_error_multiplier	N	D	Appendix C, Standard Name Modifiers	If a data variable with a standard_name modifier of standard_error has this attribute, it indicates that the values are the stated multiple of one standard error.
standard_name	S	C, D	Section 3.3, "Standard Name"	A standard name that references a description of a variable"s content in the standard name table.
title	S	G	NUG appendix B	Short description of the file contents.
units	S	C, D	NUG appendix B, Section 3.1, "Units"	Units of a variable"s content.
valid_max	N	C, D	NUG appendix B	Largest valid value of a variable.
valid_min	N	C, D	NUG appendix B	Smallest valid value of a variable.
valid_range	N	C, D	NUG appendix B	Smallest and largest valid values of a variable.

Appendix B: Standard Name Table Format

The CF standard name table is an XML document (i.e., its format adheres to the XML 1.0 [XML] recommendation). The XML suite of protocols provides a reasonable balance between human and machine readability. It also provides extensive support for internationalization. See the W3C [W3C] home page for more information.

The document begins with a header that identifies it as an XML file:

```
<?xml version="1.0"?>
```

Next is the `standard_name_table` itself, which is bracketed by the tags `<standard_name_table>` and `</standard_name_table>`.

```
<standard_name_table
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="CFStandardNameTable.xsd">
```

The content (delimited by the `<standard_name_table>` tags) consists of, in order,

```
  <institution>Name of institution here ... </institution>
  <contact>E-mail address of contact person ... </contact>
```

followed by a sequence of `entry` elements which may optionally be followed by a sequence of `alias` elements. The `entry` and `alias` elements take the following forms:

```
  <entry id="an_id">
      Define the variable whose standard_name attribute has the value "an_id".
  </entry>
  <alias id="another_id">
      Provide alias for a variable whose standard_name attribute has the
        value "another_id".
  </alias>
```

The value of the `id` attribute appearing in the `entry` and `alias` tags is a case sensitive string, containing no whitespace, which uniquely identifies the entry relative to the table. *This is the value used for a variable's* `standard_name` *attribute.*

The purpose of the `entry` elements are to provide definitions for the `id` strings. Each `entry` element contains the following elements:

```
<entry id="an_id">
  <canonical_units>Representative units for the variable ... </canonical_units>
  <description>Description of the variable ... </description>
</entry>
```

`Entry` elements may optionally also contain the following elements:

```
<grib>GRIB parameter code</grib>
<amip>AMIP identifier string</amip>
```

Not all variables have equivalent AMIP or GRIB codes. ECMWF GRIB codes start with `E` , NCEP codes with `N` . Standard codes (in the range 1-127) are not prefaced. When a variable has more than one equivalent GRIB code, the alternatives are given as a blank-separated list.

The `alias` elements do not contain definitions. Rather they contain the value of the `id` attribute of an `entry` element that contains the sought after definition. The purpose of the `alias` elements are to provide a means for maintaining the table in a backwards compatible fashion. For example, if more than one `id` string was found to correspond to identical definitions, then the redundant definitions can be converted into aliases. It is not intended that the `alias` elements be used to accommodate the use of local naming conventions in the `standard_name` attribute strings. Each `alias` element contains a single element:

```
<alias id="an_id">
  <entry_id>Identifier of the defining entry ... </entry_id>
</alias>
```

Example B.1. A name table containing three entries

```
  <?xml version="1.0"?>
  <standard_name_table>
    <institution>Program for Climate Model Diagnosis and
Intercomparison</institution>
    <contact>support@pcmdi.llnl.gov</contact>
    <entry id="surface_air_pressure">
      <canonical_units>Pa</canonical_units>
      <grib>E134</grib>
      <amip>ps</amip>
      <description>
          The surface called "surface" means the lower boundary of the atmosphere.
      </description>
    </entry>
    <entry id="air_pressure_at_sea_level">
      <canonical_units>Pa</canonical_units>
      <grib>2 E151</grib>
      <amip>psl</amip>
      <description>
          Air pressure at sea level is the quantity often abbreviated
          as MSLP or PMSL. sea_level means mean sea level, which is close
          to the geoid in sea areas.
      </description>
    </entry>
    <alias id="mean_sea_level_pressure">
      <entry_id>air_pressure_at_sea_level</entry_id>
    </alias>
  </standard_name_table>
```

The definition of a variable with the `standard_name` attribute `surface_air_pressure` is found directly since the element with `id="surface_air_pressure"` is an `entry` element which contains the definition.

The definition of a variable with the `standard_name` attribute mean_sea_level_pressure is found indirectly by first finding the element with the `id="mean_sea_level_pressure"` , and then, since this is an alias element, by searching for the element with `id="air_pressure_at_sea_level"` as indicated by the value of the `entry_id` tag.

It is possible that new tags may be added in the future. Any applications that parse the standard table should be written so that unrecognized tags are gracefully ignored.

Appendix C: Standard Name Modifiers

In the *Units* column, *u* indicates units dimensionally equivalent to those for the unmodified standard name.

Table C.1. Standard Name Modifiers

Modifier	Units	Description
`detection_minimum`	*u*	The smallest data value which is regarded as a detectable signal.
`number_of_observations`	1	The number of discrete observations or measurements from which a data value has been derived.
`standard_error`	*u*	The uncertainty of the data value. The standard error includes both systematic and statistical uncertainty. By default it is assumed that the values supplied are for one standard error. If the values supplied are for some multiple of the standard error, the `standard_error` ancillary variable should have an attribute **`standard_error_multiplier`** stating the multiplication factor.
`status_flag`		Flag values indicating the quality or other status of the data values. The variable should have **`flag_values`** or **`flag_masks`** (or both) and **`flag_meanings`** attributes to show how it should be interpreted (Section 3.5, "Flags").

Appendix D: Dimensionless Vertical Coordinates

The definitions given here allow an application to compute dimensional coordinate values from the dimensionless ones and associated variables. The formulas are expressed for a gridpoint `(n,k,j,i)` where `i` and `j` are the horizontal indices, `k` is the vertical index and `n` is the time index. A coordinate variable is associated with its definition by the value of the `standard_name` attribute. The terms in the definition are associated with file variables by the `formula_terms` attribute. The `formula_terms` attribute takes a string value, the string being comprised of blank-separated elements of the form "`term: variable`", where `term` is a keyword that represents one of the terms in the definition, and `variable` is the name of the variable in a netCDF file that contains the values for that term. The order of elements is not significant.

The gridpoint indices are not formally part of the definitions, but are included to illustrate the indices that *might* be present in the file variables. For example, a vertical coordinate whose definition contains a time index is not necessarily time dependent in all netCDF files. Also, the definitions are given in general forms that may be simplified by omitting certain terms. A term that is omitted from the `formula_terms` attribute should be assumed to be zero.

Atmosphere natural log pressure coordinate

```
standard_name = "atmosphere_ln_pressure_coordinate"
```

Definition

```
p(k) = p0 * exp(-lev(k))
```

where `p(k)` is the pressure at gridpoint `(k)`, `p0` is a reference pressure, `lev(k)` is the dimensionless coordinate at vertical gridpoint `(k)`.

The format for the `formula_terms` attribute is

```
formula_terms = "p0: var1 lev: var2"
```

Atmosphere sigma coordinate

```
standard_name = "atmosphere_sigma_coordinate"
```

Definition

```
p(n,k,j,i) = ptop + sigma(k)*(ps(n,j,i)-ptop)
```

where `p(n,k,j,i)` is the pressure at gridpoint `(n,k,j,i)`, `ptop` is the pressure at the top of the model, `sigma(k)` is the dimensionless coordinate at vertical gridpoint `(k)`, and `ps(n,j,i)` is the surface pressure at horizontal gridpoint `(j,i)`and time `(n)`.

The format for the formula_terms attribute is

```
formula_terms = "sigma: var1 ps: var2 ptop: var3"
```

Atmosphere hybrid sigma pressure coordinate

```
standard_name = "atmosphere_hybrid_sigma_pressure_coordinate"
```

Definition

```
p(n,k,j,i) = a(k)*p0 + b(k)*ps(n,j,i)
```

or

```
p(n,k,j,i) = ap(k) + b(k)*ps(n,j,i)
```

where `p(n,k,j,i)` is the pressure at gridpoint `(n,k,j,i)`, `a(k)` or `ap(k)` and `b(k)` are components of the hybrid coordinate at level `k`, `p0` is a reference pressure, and `ps(n,j,i)` is the surface pressure at horizontal gridpoint `(j,i)` and time `(n)`. The choice of whether `a(k)` or `ap(k)` is used depends on model formulation; the former is a dimensionless fraction, the latter a pressure value. In both formulations, `b(k)` is a dimensionless fraction.

The format for the `formula_terms` attribute is

```
formula_terms = "a: var1 b: var2 ps: var3 p0: var4"
```

where `a` is replaced by `ap` if appropriate.

The hybrid sigma-pressure coordinate for level `k` is defined as `a(k)+b(k)` or `ap(k)/p0+b(k)`, as appropriate.

Atmosphere hybrid height coordinate

```
standard_name = "atmosphere_hybrid_height_coordinate"
```

Definition

```
z(n,k,j,i) = a(k) + b(k)*orog(n,j,i)
```

where `z(n,k,j,i)` is the height above the geoid (approximately mean sea level) at gridpoint `(k,j,i)` and `time (n)`, `orog(n,j,i)` is the height of the surface above the geoid at `(j,i)` and `time (n)`, and `a(k)` and `b(k)` are the coordinates which define hybrid height level `k`. `a(k)` has the dimensions of height and `b(i)` is dimensionless.

The format for the `formula_terms` attribute is

```
formula_terms = "a: var1 b: var2 orog: var3"
```

There is no dimensionless hybrid height coordinate. The hybrid height is best approximated as `a(k)` if a level-dependent constant is needed.

Atmosphere smooth level vertical (SLEVE) coordinate

```
standard_name = "atmosphere_sleve_coordinate"
```

Definition

```
z(n,k,j,i) = a(k)*ztop + b1(k)*zsurf1(n,j,i) + b2(k)*zsurf2(n,j,i)
```

where `z(n,k,j,i)` is the height above the geoid (approximately mean sea level) at gridpoint `(k,j,i)` and time `(n)`, `ztop` is the height of the top of the model, and `a(k)`, `b1(k)`, and `b2(k)` are the dimensionless coordinates which define hybrid level `k`. `zsurf1(n,j,i)` and `zsurf2(n,j,i)` are respectively the large and small parts of the topography. See Shaer et al [SCH02] for details.

The format for the `formula_terms` attribute is

```
formula_terms = "a: var1 b1: var2 b2: var3 ztop: var4 zsurf1: var5
                 zsurf2: var6"
```

The hybrid height coordinate for level `k` is defined as `a(k)*ztop` .

Ocean sigma coordinate

```
standard_name = "ocean_sigma_coordinate"
```

Definition

```
z(n,k,j,i) = eta(n,j,i) + sigma(k)*(depth(j,i)+eta(n,j,i))
```

where `z(n,k,j,i)` is height, positive upwards, relative to ocean datum (e.g. mean sea level) at gridpoint `(n,k,j,i)`, `eta(n,j,i)` is the height of the ocean surface, positive upwards, relative to ocean datum at gridpoint `(n,j,i)`, `sigma(k)` is the dimensionless coordinate at vertical gridpoint `(k)`,

and `depth(j,i)` is the distance from ocean datum to sea floor (positive value) at horizontal gridpoint `(j,i)`.

The format for the `formula_terms` attribute is

```
formula_terms = "sigma: var1 eta: var2 depth: var3"
```

Ocean s-coordinate

```
standard_name = "ocean_s_coordinate"
```

Definition

```
z(n,k,j,i) = eta(n,j,i)*(1+s(k)) + depth_c*s(k) +
             (depth(j,i)-depth_c)*C(k)
```

where

```
C(k) = (1-b)*sinh(a*s(k))/sinh(a) +
       b*[tanh(a*(s(k)+0.5))/(2*tanh(0.5*a)) - 0.5]
```

where `z(n,k,j,i)` is height, positive upwards, relative to ocean datum (e.g. mean sea level) at gridpoint `(n,k,j,i)`, `eta(n,j,i)` is the height of the ocean surface, positive upwards, relative to ocean datum at gridpoint `(n,j,i)`, `s(k)` is the dimensionless coordinate at vertical gridpoint `(k)`, and `depth(j,i)` is the distance from ocean datum to sea floor (positive value) at horizontal gridpoint `(j,i)`. The constants `a`, `b`, and `depth_c` control the stretching.

The format for the `formula_terms` attribute is

```
formula_terms = "s: var1 eta: var2 depth: var3 a: var4 b: var5 depth_c: var6"
```

Ocean sigma over z coordinate

```
standard_name = "ocean_sigma_z_coordinate"
```

Definition

```
for k <= nsigma:

  z(n,k,j,i) = eta(n,j,i) + sigma(k)*(min(depth_c,depth(j,i))+eta(n,j,i))

for k > nsigma:

  z(n,k,j,i) = zlev(k)
```

where `z(n,k,j,i)` is height, positive upwards, relative to ocean datum (e.g. mean sea level) at gridpoint `(n,k,j,i)`, `eta(n,j,i)` is the height of the ocean surface, positive upwards, relative to ocean datum at gridpoint `(n,j,i)`, `sigma(k)`is the dimensionless coordinate at vertical gridpoint `(k)` for `k <= nsigma`, and `depth(j,i)` is the distance from ocean datum to sea floor (positive value) at horizontal gridpoint `(j,i)`. Above depth `depth_c` there are `nsigma` layers.

The format for the `formula_terms` attribute is

```
formula_terms = "sigma: var1 eta: var2 depth: var3 depth_c: var4 nsigma: var5
                 zlev: var6"
```

Ocean double sigma coordinate

```
standard_name = "ocean_double_sigma_coordinate"
```

Definition

```
for k <= k_c:

  z(k,j,i)= sigma(k)*f(j,i)

for k > k_c:

  z(k,j,i)= f(j,i) + (sigma(k)-1)*(depth(j,i)-f(j,i))

f(j,i)= 0.5*(z1+ z2) + 0.5*(z1-z2)* tanh(2*a/(z1-z2)*(depth(j,i)-href))
```

where `z(k,j,i)` is height, positive upwards, relative to ocean datum (e.g. mean sea level) at gridpoint `(k,j,i)`, `sigma(k)` is the dimensionless coordinate at vertical gridpoint `(k)` for `k <= k_c`, and `depth(j,i)` is the distance from ocean datum to sea floor (positive value) at horizontal gridpoint `(j,i)`. `z1`, `z2`, `a`, and `href` are constants.

The format for the `formula_terms` attribute is

```
formula_terms = "sigma: var1 depth: var2 z1: var3 z2: var4 a: var5 href: var6
                 k_c: var7"
```

Appendix E: Cell Methods

In the *Units* column, *u* indicates the units of the physical quantity before the method is applied.

Table E.1. Cell Methods

cell_methods	Units	Description
point	*u*	The data values are representative of points in space or time (instantaneous). This is the default method for a quantity that is intensive with respect to the specified dimension.
sum	*u*	The data values are representative of a sum or accumulation over the cell. This is the default method for a quantity that is extensive with respect to the specified dimension.
maximum	*u*	Maximum
median	*u*	Median
mid_range	*u*	Average of maximum and minimum
minimum	*u*	Minimum
mean	*u*	Mean (average value)
mode	*u*	Mode (most common value)
standard_deviation	*u*	Standard deviation
variance	u^2	Variance

Appendix F: Grid Mappings

Each recognized grid mapping is described in one of the sections below. Each section contains: the valid name that is used with the `grid_mapping_name` attribute; a list of the specific attributes that may be used to assign values to the mapping's parameters; the standard names used to identify the coordinate variables that contain the mapping's independent variables; and references to the mapping's definition or other information that may help in using the mapping. Since the attributes used to set a mapping's parameters may be shared among several mappings, their definitions are contained in a table in the final section. The attributes which describe the ellipsoid and prime meridian may be included, when applicable, with any grid mapping.

We have used the FGDC "Content Standard for Digital Geospatial Metadata" [FGDC] as a guide in choosing the values for `grid_mapping_name` and the attribute names for the parameters describing map projections.

Albers Equal Area

```
grid_mapping_name = albers_conical_equal_area
```

Map parameters:

- `standard_parallel` - There may be 1 or 2 values.
- `longitude_of_central_meridian`
- `latitude_of_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute values `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the `PROJ.4` software package for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/albers_equal_area_conic.html.

Azimuthal equidistant

```
grid_mapping_name = azimuthal_equidistant
```

Map parameters:

- `longitude_of_projection_origin`
- `latitude_of_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute values `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the `PROJ.4` software package for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/azimuthal_equidistant.html.

Lambert azimuthal equal area

```
grid_mapping_name = lambert_azimuthal_equal_area
```

Map parameters:

- `longitude_of_projection_origin`
- `latitude_of_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute values `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the `PROJ.4` software package for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/lambert_azimuthal_equal_area.html.

Lambert conformal

```
grid_mapping_name = lambert_conformal_conic
```

Map parameters:

- `standard_parallel` - There may be 1 or 2 values.
- `longitude_of_central_meridian`
- `latitude_of_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute values `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the `PROJ.4` software package for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/lambert_conic_conformal_2sp.html.

Lambert Cylindrical Equal Area

```
grid_mapping_name = lambert_cylindrical_equal_area
```

Map parameters:

- `longitude_of_central_meridian`
- Either `standard_parallel` or `scale_factor_at_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute value `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the PROJ.4 software packages for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/cylindrical_equal_area.html ("Lambert Cylindrical Equal Area" or EPSG 9834 or EPSG 9835). Detailed formulas can be found in [Snyder] pages 76-85.

Latitude-Longitude

```
grid_mapping_name = latitude_longitude
```

This grid mapping defines the canonical 2D geographical coordinate system based upon latitude and longitude coordinates on a spherical Earth. It is included so that the figure of the Earth can be described.

Map parameters:

None.

Map coordinates:

The rectangular coordinates are longitude and latitude identified by the usual conventions (Section 4.1, "Latitude Coordinate" and Section 4.2, "Longitude Coordinate").

Mercator

```
grid_mapping_name = mercator
```

Map parameters:

- `longitude_of_projection_origin`
- Either `standard_parallel` (EPSG 9805) or `scale_factor_at_projection_origin` (EPSG 9804)

- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute value `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the PROJ.4 software packages for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/mercator_1sp.html ("Mercator (1SP)" or EPSG 9804) or http://www.remotesensing.org/geotiff/proj_list/mercator_2sp.html ("Mercator (2SP)" or EPSG 9805).

More information on formulas available in [OGP-EPSG_GN7_2].

Orthographic

```
grid_mapping_name = orthographic
```

Map parameters:

- `longitude_of_projection_origin`
- `latitude_of_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute value `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the PROJ.4 software packages for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/orthographic.html ("Orthographic" or EPSG 9840).

More information on formulas available in [OGP-EPSG_GN7_2].

Polar stereographic

```
grid_mapping_name = polar_stereographic
```

Map parameters:

- `straight_vertical_longitude_from_pole`
- `latitude_of_projection_origin` - Either +90. or -90.
- Either `standard_parallel` or `scale_factor_at_projection_origin`
- `false_easting`

- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute values `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the `PROJ.4` software package for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/polar_stereographic.html.

Rotated pole

```
grid_mapping_name = rotated_latitude_longitude
```

Map parameters:

- `grid_north_pole_latitude`
- `grid_north_pole_longitude`
- `north_pole_grid_longitude` - This parameter is option (default is 0).

Map coordinates:

The rotated latitude and longitude coordinates are identified by the `standard_name` attribute values `grid_latitude` and `grid_longitude` respectively.

Notes:

Stereographic

```
grid_mapping_name = stereographic
```

Map parameters:

- `longitude_of_projection_origin`
- `latitude_of_projection_origin`
- `scale_factor_at_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute values `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Formulas for the mapping and its inverse along with notes on using the `PROJ.4` software package for doing the calcuations may be found at

http://www.remotesensing.org/geotiff/proj_list/stereographic.html . See the section "Polar stereographic" for the special case when the projection origin is one of the poles.

Transverse Mercator

```
grid_mapping_name = transverse_mercator
```

Map parameters:

- `scale_factor_at_central_meridian`
- `longitude_of_central_meridian`
- `latitude_of_projection_origin`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute values `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Formulas for the mapping and its inverse along with notes on using the `PROJ.4` software package for doing the calcuations may be found at http://www.remotesensing.org/geotiff/proj_list/transverse_mercator.html.

Vertical perspective

```
grid_mapping_name = vertical_perspective
```

Map parameters:

- `latitude_of_projection_origin`
- `longitude_of_projection_origin`
- `perspective_point_height`
- `false_easting`
- `false_northing`

Map coordinates:

The x (abscissa) and y (ordinate) rectangular coordinates are identified by the `standard_name` attribute value `projection_x_coordinate` and `projection_y_coordinate` respectively.

Notes:

Notes on using the `PROJ.4` software packages for computing the mapping may be found at http://www.remotesensing.org/geotiff/proj_list/geos.html. These notes assume the point of perspective is directly over the equator. A more general description of vertical perspective projection is given in [Snyder], pages 169-181.

In the following table the "Type" values are **S** for string and **N** for numeric.

Table F.1. Grid Mapping Attributes

Attribute	Type	Description
`earth_radius`	N	Used to specify the radius, in metres, of the spherical figure used to approximate the shape of the Earth. This attribute should be specified for those projected coordinate reference systems in which the X-Y cartesian coordinates have been derived using a spherical Earth approximation. If the cartesian coordinates were derived using an ellipsoid, this attribute should not be defined. Example: "6371007", which is the radius of the GRS 1980 Authalic Sphere.
`false_easting`	N	The value added to all abscissa values in the rectangular coordinates for a map projection. This value frequently is assigned to eliminate negative numbers. Expressed in the unit of the coordinate variable identified by the standard name `projection_x_coordinate`.
`false_northing`	N	The value added to all ordinate values in the rectangular coordinates for a map projection. This value frequently is assigned to eliminate negative numbers. Expressed in the unit of the coordinate variable identified by the standard name `projection_y_coordinate`.
`grid_mapping_name`	N	The name used to identify the grid mapping.
`grid_north_pole_latitude`	N	True latitude (degrees_north) of the north pole of the rotated grid.
`grid_north_pole_longitude`	N	True longitude (degrees_east) of the north pole of the rotated grid.
`inverse_flattening`	N	Used to specify the *inverse* flattening (*1/f*) of the ellipsoidal figure associated with the geodetic datum and used to approximate the shape of the Earth. The flattening (*f*) of the ellipsoid is related to the semi-major and semi-minor axes by the formula *f = (a-b)/a*. In the case of a spherical Earth this attribute should be omitted or set to zero. Example: 298.257222101 for the GRS 1980 ellipsoid. (Note: By convention the dimensions of an ellipsoid are specified using either the semi-major and semi-minor axis lengths, or the semi-major axis length and the inverse flattening. If all three attributes are specified then the supplied values must be consistent with the aforementioned formula.)
`latitude_of_projection_origin`	N	The latitude chosen as the origin of rectangular coordinates for a map projection. Domain: `-90.0 <= latitude_of_projection_origin <= 90.0`
`longitude_of_central_meridian`	N	The line of longitude at the center of a map projection generally used as the basis for constructing the projection. Domain: `-180.0 <= longitude_of_central_meridian < 180.0`

Attribute	Type	Description
longitude_of_prime_meridian	N	Specifies the longitude, with respect to Greenwich, of the prime meridian associated with the geodetic datum. The prime meridian defines the origin from which longitude values are determined. Not to be confused with the projection origin longitude (cf. longitude_of_projection_origin, a.k.a. central meridian) which defines the longitude of the map projection origin. Domain: -180.0 <= longitude_of_prime_meridian < 180.0 decimal degrees. Default = 0.0
longitude_of_projection_origin	N	The longitude chosen as the origin of rectangular coordinates for a map projection. Domain: -180.0 <= longitude_of_projection_origin < 180.0
north_pole_grid_longitude	N	Longitude (degrees) of the true north pole in the rotated grid.
perspective_point_height	N	Records the height, *in metres*, of the map projection perspective point above the ellipsoid (or sphere). Used by perspective-type map projections, for example the Vertical Perspective Projection, which may be used to simulate the view from a Meteosat satellite.
scale_factor_at_central_meridian	N	A multiplier for reducing a distance obtained from a map by computation or scaling to the actual distance along the central meridian. Domain: scale_factor_at_central_meridian > 0.0
scale_factor_at_projection_origin	N	A multiplier for reducing a distance obtained from a map by computation or scaling to the actual distance at the projection origin. Domain: scale_factor_at_projection_origin > 0.0
semi_major_axis	N	Specifies the length, *in metres*, of the semi-major axis of the ellipsoidal figure associated with the geodetic datum and used to approximate the shape of the Earth. Commonly denoted using the symbol *a*. In the case of a spherical Earth approximation this attribute defines the radius of the Earth. See also the inverse_flattening attribute.
semi_minor_axis	N	Specifies the length, *in metres*, of the semi-minor axis of the ellipsoidal figure associated with the geodetic datum and used to approximate the shape of the Earth. Commonly denoted using the symbol *b*. In the case of a spherical Earth approximation this attribute should be omitted (the preferred option) or else set equal to the value of the semi_major_axis attribute. See also the inverse_flattening attribute.

Attribute	Type	Description
`standard_parallel`	N	Specifies the line, or lines, of latitude at which the developable map projection surface (plane, cone, or cylinder) touches the reference sphere or ellipsoid used to represent the Earth. Since there is zero scale distortion along a standard parallel it is also referred to as a "latitude of true scale". In the situation where a conical developable surface intersects the reference ellipsoid there are two standard parallels, in which case this attribute can be used as a vector to record both latitude values, with the additional convention that the standard parallel nearest the pole (N or S) is provided first. Domain: `-90.0 <= standard_parallel <= 90.0`
`straight_vertical_longitude_from_pole`	N	The longitude to be oriented straight up from the North or South Pole. Domain: `-180.0 <= straight_vertical_longitude_from_pole < 180.0`

Appendix G: Revision History

14 June 2004

1. Added the section called “Lambert azimuthal equal area”.
2. the section called "Polar Stereographic" : Added `latitude_of_projection_origin` map parameter.

1 July 2004

1. Section 5.7, "Scalar Coordinate Variables" : Added note that use of scalar coordinate variables inhibits interoperability with COARDS conforming applications.
2. Example 5.11, "Multiple forecasts from a single analysis" : Added `positive` attribute to the scalar coordinate p500 to make it unambiguous that the pressure is a vertical coordinate value.

20 September 2004

1. Section 7.3, "Cell Methods" : Changed several incorrect occurances of the cell method `"standard deviation"` to `"standard_deviation"`.

22 October 2004

1. Added Example 5.7, "Lambert conformal projection".

25 November 2005

1. the section called "Atmosphere hybrid height coordinate" : Fixed definition of atmosphere hybrid height coordinate.

21 March 2006

1. Added the section called "Azimuthal equidistant".
2. Added the section called "Atmosphere natural log pressure coordinate".

17 January 2008

1. Preface : Changed text to refer to rules of CF governance, and provisional status.
2. Chapter 4, Coordinate Types , Chapter 5, Coordinate Systems : Made changes regarding use of the axis attribute to identify horizontal coordinate variables.
3. Changed document version to 1.1.

4 May 2008

1. Section 5.6, "Horizontal Coordinate Reference Systems, Grid Mappings, and Projections", Appendix F, Grid Mappings : Additions and revisions to CF grid mapping attributes to support the specification of coordinate reference system properties (Trac ticket #18).
2. Table 3.1, "Supported Units" : Corrected Prefix for Factor "1e-2" from "deci" to "centi". (Trac ticket #25).
3. Changed document version to 1.2.

15 July 2008

1. Section 3.5, "Flags" , Appendix A, Attributes , Appendix C, Standard Name Modifiers : Enhanced the Flags definition to support bit field notation using a `flag_masks` attribute. (Trac ticket #26).

2. Changed document version to 1.3.

9 October 2008

1. Fixed defect in Example 4.3, “Atmosphere sigma coordinate”. (Trac ticket #30).
2. Fixed defect in Chapter 5, Coordinate Systems. (Trac ticket #32).

7 November 2008

1. Fixed defect in wording of Chapter 5, Coordinate Systems. (Trac ticket #35).
2. Fixed defect related to subsection headings in Appendix D, Dimensionless Vertical Coordinates. (Trac ticket #36).

10 December 2008

1. Changes related to removing ambiguity in Section 7.3, "Cell Methods". (Trac ticket #17).
2. Changed document version to 1.4.

11 December 2008

1. Added grid mappings Lambert Cylindrical Equal Area, Mercator, and Orthographic to Appendix F, Grid Mappings. (Trac ticket #34).

12 December 2008

1. Fixed defect in Mercator section of Appendix F, Grid Mappings by updating to version 12 of Grid Map Names (see http://cf-trac.llnl.gov/trac/wiki/GridMapNames?version=12).

27 February 2009

1. Fixed defect by clarifying that coordinates indicate gridpoint location in Chapter 4, Coordinate Types. (Trac ticket #44).
2. Fixed defect of outdated Conventions attribute. (Trac ticket #45).

25 October 2010

Minor revisions requested by Jonathan Gregory. Revisions 33 and 49 were closed after discussions; the rest had elicited no objections.

1. Ticket 33, cell_methods for statistical indices
2. Ticket 49, clarification of flag_meanings attribute
3. Ticket 58, remove deprecation of "missing_value" attribute
4. Ticket 57, fix for broken URLs in CF Conventions document
5. Ticket 56, typo in CF conventions doc
6. Ticket 51, syntax consistency for dimensionless vertical coordinate definitions
7. Ticket 47, error in example 7.4
8. Changed document version to 1.5.
9. New chapter, ticket 37 Changed document version to 1.6.

22 June 2011

Ticket 37. Added Chapter 9, Discrete Sampling Geometries, and a related Appendix H, and revised several other chapters.

5 December 2011

In Appendix H (Annotated Examples of Descrete Geometries), updated standard names "station_description" and "station_wmo_id" to "platform_name" and "platform_id".

Appendix H: Annotated Examples of Discrete Geometries

H.1. Point Data

To represent data at scattered locations and times with no implied relationship among of coordinate positions, both data and coordinates must share the same (sample) instance dimension. Because each feature contains only a single data element, there is no need for a separate element dimension. The representation of point features is a special, degenerate case of the standard four representations. The `coordinates` attribute is used on the data variables to unambiguously identify the relevant space and time auxiliary coordinate variables.

Example H.1. Point data.

```
dimensions:
   obs = 1234 ;

variables:
   double time(obs) ;
       time:standard_name = “time”;
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon(obs) ;
       lon:standard_name = "longitude";
       lon:long_name = "longitude of the observation";
       lon:units = "degrees_east";
   float lat(obs) ;
       lat:standard_name = "latitude";
       lat:long_name = "latitude of the observation" ;
       lat:units = "degrees_north" ;
   float alt(obs) ;
       alt:long_name = "vertical distance above the surface" ;
       alt:standard_name = "height" ;
       alt:units = "m";
       alt:positive = "up";
       alt:axis = "Z";

   float humidity(obs) ;
       humidity:standard_name = "specific_humidity" ;
       humidity:coordinates = "time lat lon alt" ;
   float temp(obs) ;
       temp:standard_name = "air_temperature" ;
       temp:units = "Celsius" ;
       temp:coordinates = "time lat lon alt" ;

attributes:
   :featureType = "point";
```

In this example, the humidity(i) and temp(i) data are associated with the coordinate values time(i), lat(i), lon(i), and alt(i). The obs dimension may optionally be the netCDF unlimited dimension of the netCDF file.

H.2. Time Series Data

Data may be taken over periods of time at a set of discrete point, spatial locations called stations (see also discussion in 9.1). The set of elements at a particular station is referred to as a timeSeries feature and a data variable may contain a collection of such features. The instance dimension in the case of timeSeries specifies the number of time series in the collection and is also referred to as the station dimension. The instance variables, which have just this dimension, including latitude and longitude for example, are also referred to as station variables and are considered to contain

information describing the stations. The station variables may contain missing values, allowing one to reserve space for additional stations that may be added at a later time, as discussed in section 9.6. In addition,

- It is strongly recommended that there should be a station variable (which may be of any type) with the attribute `cf_role="timeseries_id"` , whose values uniquely identify the stations.
- It is recommended that there should be station variables with standard_name attributes " `platform_name` ", " `surface_altitude` " and “ `platform_id` ” when applicable.

All the representations described in section 9.3 can be used for time series. The global attribute `featureType="timeSeries"` (case-insensitive) must be included.

H.2.1. Orthogonal multidimensional array representation of time series

If the time series instances have the same number of elements and the time values are identical for all instances, you may use the orthogonal multidimensional array representation. This has either a one-dimensional coordinate variable, time(time), provided the time values are ordered monotonically, or a one-dimensional auxiliary coordinate variable, time(o), where o is the element dimension. In the former case, listing the time variable in the `coordinates` attributes of the data variables is optional.

Example H.2. Timeseries with common element times in a time coordinate variable using the orthogonal multidimensional array representation.

```
dimensions:
  station = 10 ;  // measurement locations
  time = UNLIMITED ;
variables:
  float humidity(station,time) ;
    humidity:standard_name = "specific humidity" ;
    humidity:coordinates = "lat lon alt" ;
  double time(time) ;
    time:standard_name = "time";
    time:long_name = "time of measurement" ;
    time:units = "days since 1970-01-01 00:00:00" ;
  float lon(station) ;
    lon:standard_name = "longitude";
    lon:long_name = "station longitude";
    lon:units = "degrees_east";
  float lat(station) ;
    lat:standard_name = "latitude";
    lat:long_name = "station latitude" ;
    lat:units = "degrees_north" ;
  float alt(station) ;
    alt:long_name = "vertical distance above the surface" ;
    alt:standard_name = "height" ;
    alt:units = "m";
    alt:positive = "up";
    alt:axis = "Z";
  char station_name(station, name_strlen) ;
    station_name:long_name = "station name" ;
    station_name:cf_role = "timeseries_id";
attributes:
    :featureType = "timeSeries";
```

In this example, `humidity(i,o)` is element o of time series i, and associated with the coordinate values `time(o)` , `lat(i)` , and `lon(i)` . Either the instance (station) or the element (time) dimension may optionally be the netCDF unlimited dimension.

H.2.2. Incomplete multidimensional array representation of time series

Much of the simplicity of the orthogonal multidimensional representation can be preserved even in cases where individual time series have different time coordinate values. All time series must be allocated the amount of staorage needed by the longest, so the use of this representation will trade off simplicity against storage space in some cases.

Example H.3. Timeseries of station data in the incomplete multidimensional array representation.

```
dimensions:
   station = UNLIMITED ;
   obs = 13 ;

variables:
   float lon(station) ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
       lon:units = "degrees_east";
   float lat(station) ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;
   float alt(station) ;
       alt:long_name = "vertical distance above the surface" ;
       alt:standard_name = "height" ;
       alt:units = "m";
       alt:positive = "up";
       alt:axis = "Z";
   char station_name(station, name_strlen) ;
       station_name:long_name = "station name" ;
       station_name:cf_role = "timeseries_id";
   int station_info(station) ;
       station_info:long_name = "any kind of station info" ;
   float station_elevation(station) ;
       station_elevationalt:long_name = "height above the geoid" ;
       station_elevationalt:standard_name = "surface_altitude" ;
       station_elevationalt:units = "m";

   double time(station, obs) ;
       time:standard_name = "time";
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
       time:missing_value = -999.9;
   float humidity(station, obs) ;
       humidity:standard_name = “specific_humidity” ;
       humidity:coordinates = "time lat lon alt" ;
       humidity:_FillValue = -999.9;
   float temp(station, obs) ;
       temp:standard_name = “air_temperature” ;
       temp:units = "Celsius" ;
       temp:coordinates = "time lat lon alt" ;
       temp:_FillValue = -999.9;

attributes:
       :featureType = "timeSeries";
```

In this example, the humidity(i,o) and temp(i,o) data for element o of time series i are associated with the coordinate values time(i,o), lat(i), lon(i) and alt(i). Either the instance

(station) dimension or the element (obs) dimension could be the unlimited dimension of a netCDF file. Any unused elements of the data and auxiliary coordinate variables must contain the missing data flag value(section 9.6).

H.2.3. Single time series, including deviations from a nominal fixed spatial location

When the intention of a data variable is to contain only a single time series, the preferred encoding is a special case of the multidimensional array representation.

Example H.4. A single timeseries.

```
dimensions:
   time = 100233 ;
   name_strlen = 23 ;

variables:
   float lon ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
       lon:units = "degrees_east";
   float lat ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;
   float alt ;
       alt:long_name = "vertical distance above the surface" ;
       alt:standard_name = "height" ;
       alt:units = "m";
       alt:positive = "up";
       alt:axis = "Z";
   char station_name(name_strlen) ;
       station_name:long_name = "station name" ;
       station_name:cf_role = "timeseries_id";

   double time(time) ;
       time:standard_name = "time";
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
       time:missing_value = -999.9;
   float humidity(time) ;
       humidity:standard_name = “specific_humidity” ;
       humidity:coordinates = "time lat lon alt" ;
       humidity:_FillValue = -999.9;
   float temp(time) ;
       temp:standard_name = “air_temperature” ;
       temp:units = "Celsius" ;
       temp:coordinates = "time lat lon alt" ;
       temp:_FillValue = -999.9;

attributes:
       :featureType = "timeSeries";
```

While an idealized time series is defined at a single, stable point location, there are examples of time series, such as cabled ocean surface mooring measurements, in which the precise position of the observations varies slightly from a nominal fixed point. In the following example we show how the spatial positions of such a time series should be encoded in CF. Note that although this example shows only a single time series, the technique is applicable to all of the representations.

Example H.5. A single timeseries with time-varying deviations from a nominal point spatial location

```
dimensions:
   time = 100233 ;
   name_strlen = 23 ;

variables:
   float lon ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
       lon:units = "degrees_east";
       lon:axis = “X”;
   float lat ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;
       lat: axis = “Y” ;
   float precise_lon (time);
       precise_lon:standard_name = "longitude";
       precise_lon:long_name = "station longitude";
       precise_lon:units = "degrees_east";
   float precise_lat (time);
       precise_lat:standard_name = "latitude";
       precise_lat:long_name = "station latitude" ;
       precise_lat:units = "degrees_north" ;
   float alt ;
       alt:long_name = "vertical distance above the surface" ;
       alt:standard_name = "height" ;
       alt:units = "m";
       alt:positive = "up";
       alt:axis = "Z";
   char station_name(name_strlen) ;
       station_name:long_name = "station name" ;
       station_name:cf_role = "timeseries_id";

   double time(time) ;
       time:standard_name = "time";
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
       time:missing_value = -999.9;
   float humidity(time) ;
       humidity:standard_name = “specific_humidity” ;
       humidity:coordinates = "time lat lon alt precise_lon precise_lat" ;
       humidity:_FillValue = -999.9;
   float temp(time) ;
       temp:standard_name = “air_temperature” ;
       temp:units = "Celsius" ;
       temp:coordinates = "time lat lon alt precise_lon precise_lat " ;
       temp:_FillValue = -999.9;
```

```
    attributes:
           :featureType = "timeSeries";
```

H.2.4. Contiguous ragged array representation of time series

When the time series have different lengths and the data values for entire time series are available to be written in a single operation, the contiguous ragged array representation is efficient.

Example H.6. Timeseries of station data in the contiguous ragged array representation.

```
dimensions:
   station = 23 ;
   obs = 1234 ;

variables:
   float lon(station) ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
       lon:units = "degrees_east";
   float lat(station) ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;
   float alt(station) ;
       alt:long_name = "vertical distance above the surface" ;
       alt:standard_name = "height" ;
       alt:units = "m";
       alt:positive = "up";
       alt:axis = "Z";
   char station_name(station, name_strlen) ;
       station_name:long_name = "station name" ;
       station_name:cf_role = "timeseries_id";
   int station_info(station) ;
       station_info:long_name = "some kind of station info" ;
   int row_size(station) ;
       row_size:long_name = "number of observations for this station " ;
       row_size:sample_dimension = "obs" ;

   double time(obs) ;
       time:standard_name = "time";
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float humidity(obs) ;
       humidity:standard_name = “specific_humidity” ;
       humidity:coordinates = "time lat lon alt" ;
       humidity:_FillValue = -999.9;
   float temp(obs) ;
       temp:standard_name = “air_temperature” ;
       temp:units = "Celsius" ;
       temp:coordinates = "time lat lon alt" ;
       temp:_FillValue = -999.9;

attributes:
       :featureType = "timeSeries";
```

The data humidity(o) and temp(o) are associated with the coordinate values time(o), lat(i), lon(i), and alt(i), where i indicates which time series. Time series i comprises the data elements from

```
rowStart(i) to rowStart(i) + row_size(i) - 1
```

where

```
rowStart(i) = 0 if i = 0
rowStart(i) = rowStart(i-1) + row_size(i-1) if i > 0
```

The variable, `row_size` , is the count variable containing the length of each time series feature. It is identified by having an attribute with name `sample_dimension` whose value is name of the sample dimension (`obs` in this example). The sample dimension could optionally be the netCDF unlimited dimension. The variable bearing the `sample_dimension` attribute must have the instance dimension (`station` in this example) as its single dimension, and must be of type integer. This variable implicitly partitions into individual instances all variables that have the sample dimension. The auxiliary coordinate variables `lat` , `lon` , `alt` and `station_name` are station variables.

H.2.5. Indexed ragged array representation of time series

When time series with different lengths are written incrementally, the indexed ragged array representation is efficient.

Example H.7. Timeseries of station data in the indexed ragged array representation.

```
dimensions:
   station = 23 ;
   obs = UNLIMITED ;

variables:
   float lon(station) ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
       lon:units = "degrees_east";
   float lat(station) ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;
   float alt(station) ;
       alt:long_name = "vertical distance above the surface" ;
       alt:standard_name = "height" ;
       alt:units = "m";
       alt:positive = "up";
       alt:axis = "Z";
   char station_name(station, name_strlen) ;
       station_name:long_name = "station name" ;
       station_name:cf_role = "timeseries_id";
   int station_info(station) ;
       station_info:long_name = "some kind of station info" ;

   int stationIndex(obs) ;
       stationIndex:long_name = "which station this obs is for" ;
       stationIndex:instance_dimension= "station" ;
   double time(obs) ;
       time:standard_name = "time";
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float humidity(obs) ;
       humidity:standard_name = “specific_humidity” ;
       humidity:coordinates = "time lat lon alt" ;
       humidity:_FillValue = -999.9;
   float temp(obs) ;
       temp:standard_name = “air_temperature” ;
       temp:units = "Celsius" ;
       temp:coordinates = "time lat lon alt" ;
       temp:_FillValue = -999.9;

attributes:
       :featureType = "timeSeries";
```

The humidity(o) and temp(o) data are associated with the coordinate values time(o), lat(i), lon(i), and alt(i), where i = stationIndex(o) is a zero-based index indicating which time series. Thus, time(0), humidity(0) and temp(0) belong to the element of the `station` dimension that is indicated by `stationIndex(0)` ; time(1), humidity(1) and temp(1) belong to element

`stationIndex(1)` of the `station` dimension, etc.

The variable, `stationIndex` , is identified as the index variable by having an attribute with name of `instance_dimension` whose value is the instance dimension (`station` in this example). The variable bearing the `instance_dimension` attribute must have the sample dimension (`obs` in this example) as its single dimension, and must be type integer. This variable implicitly assigns the station to each value of any variable having the sample dimension. The sample dimension need not be the netCDF unlimited dimension, though it commonly is.

H.3. Profile Data

A series of connected observations along a vertical line, like an atmospheric or ocean sounding, is called a profile. For each profile, there is a single time, lat and lon. A data variable may contain a collection of profile features. The instance dimension in the case of profiles specifies the number of profiles in the collection and is also referred to as the **profile dimension** . The instance variables, which have just this dimension, including latitude and longitude for example, are also referred to as **profile variables** and are considered to be information about the profiles. It is strongly recommended that there always be a profile variable (of any data type) with `cf_role` attribute " `profile_id` ", whose values uniquely identify the profiles. The profile variables may contain missing values. This allows one to reserve space for additional profiles that may be added at a later time, as discussed in section 9.6. All the representations described in section 9.1.3 can be used for profiles. The global attribute `featureType="profile"` (case-insensitive) should be included if all data variables in the file contain profiles.

H.3.1. Orthogonal multidimensional array representation of profiles

If the profile instances have the same number of elements and the vertical coordinate values are identical for all instances, you may use the orthogonal multidimensional array representation. This has either a one-dimensional coordinate variable, z(z), provided the vertical coordinate values are ordered monotonically, or a one-dimensional auxiliary coordinate variable, alt(o), where o is the element dimension. In the former case, listing the vertical coordinate variable in the **coordinates** attributes of the data variables is optional.

Example H.8. Atmospheric sounding profiles for a common set of vertical coordinates stored in the orthogonal multidimensional array representation.

```
dimensions:
   z = 42 ;
   profile = 142 ;

variables:
   int profile(profile) ;
         profile:cf_role = "profile_id";
   double time(profile);
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon(profile);
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
   float lat(profile);
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;

   float z(z) ;
       z:standard_name = “altitude”;
       z:long_name = "height above mean sea level" ;
       z:units = "km" ;
       z:positive = "up" ;
       z:axis = "Z" ;

   float pressure(profile, z) ;
       pressure:standard_name = "air_pressure" ;
       pressure:long_name = "pressure level" ;
       pressure:units = "hPa" ;
       pressure:coordinates = "time lon lat altz" ;

   float temperature(profile, z) ;
       temperature:standard_name = "surface_temperature" ;
       temperature:long_name = "skin temperature" ;
       temperature:units = "Celsius" ;
       temperature:coordinates = "time lon lat altz" ;

   float humidity(profile, z) ;
       humidity:standard_name = "relative_humidity" ;
       humidity:long_name = "relative humidity" ;
       humidity:units = "%" ;
       humidity:coordinates = "time lon lat altz" ;

attributes:
   :featureType = "profile";
```

The pressure(i,o), temperature(i,o), and humidity(i,o) data for element o of profile i are associated with the coordinate values time(i), lat(i), and lon(i). The vertical coordinate for

element o in each profile is altitude z(o). Either the instance (profile) or the element (z) dimension could be the netCDF unlimited dimension.

H.3.2. Incomplete multidimensional array representation of profiles

If there are the same number of levels in each profile, but they do not have the same set of vertical coordinates, one can use the incomplete multidimensional array representation, which the vertical coordinate variable is two-dimensional e.g. replacing z(z) in Example H.8, "Atmospheric sounding profiles for a common set of vertical coordinates stored in the orthogonal multidimensional array representation." with alt(profile,z). This representation also allows one to have a variable number of elements in different profiles, at the cost of some wasted space. In that case, any unused elements of the data and auxiliary coordinate variables must contain missing data values (section 9.6).

H.3.3. Single profile

When a single profile is stored in a file, there is no need for the profile dimension; the data arrays are one-dimensional. This is a special case of the orthogonal multidimensional array representation (9.3.1).

Example H.9. Data from a single atmospheric sounding profile.

```
dimensions:
   z = 42 ;

variables:
   int profile ;
       profile:cf_role = "profile_id";

   double time;
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon;
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
   float lat;
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;

   float z(z) ;
       z:standard_name = “altitude”;
       z:long_name = "height above mean sea level" ;
       z:units = "km" ;
       z:positive = "up" ;
       z:axis = "Z" ;

   float pressure(z) ;
       pressure:standard_name = "air_pressure" ;
       pressure:long_name = "pressure level" ;
       pressure:units = "hPa" ;
       pressure:coordinates = "time lon lat z" ;

   float temperature(z) ;
       temperature:standard_name = "surface_temperature" ;
       temperature:long_name = "skin temperature" ;
       temperature:units = "Celsius" ;
       temperature:coordinates = "time lon lat z" ;

   float humidity(z) ;
       humidity:standard_name = "relative_humidity" ;
       humidity:long_name = "relative humidity" ;
       humidity:units = "%" ;
       humidity:coordinates = "time lon lat z" ;

attributes:
   :featureType = "profile";
```

The pressure(o), temperature(o), and humidity(o) data is associated with the coordinate values time, z(o), lat, and lon. The profile variables time, lat and lon, shown here as scalar, could

alternatively be one-dimensional time(profile), lat(profile), lon(profile) if a size-one profile dimension were retained in the file.

H.3.4. Contiguous ragged array representation of profiles

When the number of vertical levels for each profile varies, and one can control the order of writing, one can use the contiguous ragged array representation. The canonical use case for this is when rewriting raw data, and you expect that the common read pattern will be to read all the data from each profile.

Example H.10. Atmospheric sounding profiles for a common set of vertical coordinates stored in the contiguous ragged array representation.

```
dimensions:
   obs = UNLIMITED ;
   profile = 142 ;

variables:
   int profile(profile) ;
       profile:cf_role = "profile_id";
   double time(profile);
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon(profile);
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
   float lat(profile);
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;
    int rowSize(profile) ;
       rowSize:long_name = "number of obs for this profile " ;
       rowSize:sample_dimension = "obs" ;

   float z(obs) ;
       z:standard_name = “altitude”;
       z:long_name = "height above mean sea level" ;
       z:units = "km" ;
       z:positive = "up" ;
       z:axis = "Z" ;

   float pressure(obs) ;
       pressure:standard_name = "air_pressure" ;
       pressure:long_name = "pressure level" ;
       pressure:units = "hPa" ;
       pressure:coordinates = "time lon lat z" ;
```

```
      float temperature(obs) ;
          temperature:standard_name = "surface_temperature" ;
          temperature:long_name = "skin temperature" ;
          temperature:units = "Celsius" ;
          temperature:coordinates = "time lon lat z" ;

      float humidity(obs) ;
          humidity:standard_name = "relative_humidity" ;
          humidity:long_name = "relative humidity" ;
          humidity:units = "%" ;
          humidity:coordinates = "time lon lat z" ;

   attributes:
      :featureType = "profile";
```

The pressure(o), temperature(o), and humidity(o) data is associated with the coordinate values time(i), z(o), lat(i), and lon(i), where i indicates which profile. All elements for one profile are contiguous along the sample dimension. The sample dimension (obs) may be the unlimited dimension or not. All variables that have the instance dimension (profile) as their single dimension are considered to be information about the profiles.

The count variable (row_size) contains the number of elements for each profile, and is identified by having an attribute with name "sample_dimension" whose value is the sample dimension being counted. It must have the profile dimension as its single dimension, and must be type integer. The elements are associated with the profile using the same algorithm as in H.2.4.

H.3.5. Indexed ragged array representation of profiles

When the number of vertical levels for each profile varies, and one cannot write them contiguously, one can use the indexed ragged array representation. The canonical use case is when writing real-time data streams that contain reports from many profiles, arriving randomly. If the sample dimension is the unlimited dimension, this allows data to be appended to the file.

Example H.11. Atmospheric sounding profiles for a common set of vertical coordinates stored in the indexed ragged array representation.

```
   dimensions:
      obs = UNLIMITED ;
      profiles = 142 ;

   variables:
      int profile(profile) ;
          profile:cf_name = "profile_id";
      double time(profile);
          time:standard_name = "time";
          time:long_name = "time" ;
          time:units = "days since 1970-01-01 00:00:00" ;
```

```
      float lon(profile);
          lon:standard_name = "longitude";
          lon:long_name = "longitude" ;
          lon:units = "degrees_east" ;
      float lat(profile);
          lat:standard_name = "latitude";
          lat:long_name = "latitude" ;
          lat:units = "degrees_north" ;

      int parentIndex(obs) ;
          parentIndex:long_name = "index of profile " ;
          parentIndex:instance_dimension= "profile" ;

       float z(obs) ;
          z:standard_name = “altitude”;
          z:long_name = "height above mean sea level" ;
          z:units = "km" ;
          z:positive = "up" ;
          z:axis = "Z" ;

      float pressure(obs) ;
          pressure:standard_name = "air_pressure" ;
          pressure:long_name = "pressure level" ;
          pressure:units = "hPa" ;
          pressure:coordinates = "time lon lat z" ;

      float temperature(obs) ;
          temperature:standard_name = "surface_temperature" ;
          temperature:long_name = "skin temperature" ;
          temperature:units = "Celsius" ;
          temperature:coordinates = "time lon lat z" ;

      float humidity(obs) ;
          humidity:standard_name = "relative_humidity" ;
          humidity:long_name = "relative humidity" ;
          humidity:units = "%" ;
          humidity:coordinates = "time lon lat z" ;

   attributes:
      :featureType = "profile";
```

The pressure(o), temperature(o), and humidity(o) data are associated with the coordinate values time(i), z(o), lat(i), and lon(i), where i indicates which profile. The sample dimension (obs) may be the unlimited dimension or not. The profile index variable (parentIndex) is identified by having an attribute with name of "instance_dimension" whose value is the profile dimension name. It must have the sample dimension as its single dimension, and must be type integer. Each value in the profile index variable is the zero-based profile index that the element belongs to. The elements are associated with the profiles using the same algorithm as in H.2.5.

H.4. Trajectory Data

Data may be taken along discrete paths through space, each path constituting a connected set of points called a trajectory, for example along a flight path, a ship path or the path of a parcel in a Lagrangian calculation. A data variable may contain a collection of trajectory features. The instance dimension in the case of trajectories specifies the number of trajectories in the collection and is also referred to as the **trajectory dimension** . The instance variables, which have just this dimension, are also referred to as **trajectory variables** and are considered to be information about the trajectories. It is strongly recommended that there always be a trajectory variable (of any data type) with the attribute `cf_role="trajectory_id"` attribute, whose values uniquely identify the trajectories. The trajectory variables may contain missing values. This allows one to reserve space for additional trajectories that may be added at a later time, as discussed in section 9.6. All the representations described in section 9.3 can be used for trajectories. The global attribute `featureType="trajectory"` (case-insensitive) should be included if all data variables in the file contain trajectories.

H.4.1. Multidimensional array representation of trajectories

When storing multiple trajectories in the same file, and the number of elements in each trajectory is the same, one can use the multidimensional array representation. This representation also allows one to have a variable number of elements in different trajectories, at the cost of some wasted space. In that case, any unused elements of the data and auxiliary coordinate variables must contain missing data values (section 9.6).

Example H.12. Trajectories recording atmospheric composition in the incomplete multidimensional array representation.

```
dimensions:
   obs = 1000 ;
   trajectory = 77 ;

variables:
   char trajectory(trajectory, name_strlen) ;
     trajectory:cf_role = "trajectory_id";
     trajectory:long_name = "trajectory name" ;
   int trajectory_info(trajectory) ;
       trajectory_info:long_name = "some kind of trajectory info"

   double time(trajectory, obs) ;
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon(trajectory, obs) ;
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
   float lat(trajectory, obs) ;
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;

   float z(trajectory, obs) ;
       z:standard_name = “altitude”;
       z:long_name = "height above mean sea level" ;
       z:units = "km" ;
       z:positive = "up" ;
        z:axis = "Z" ;

   float O3(trajectory, obs) ;
       O3:standard_name = “mass_fraction_of_ozone_in_air”;
       O3:long_name = "ozone concentration" ;
       O3:units = "1e-9" ;
       O3:coordinates = "time lon lat z" ;

   float NO3(trajectory, obs) ;
       NO3:standard_name = “mass_fraction_of_nitrate_radical_in_air”;
       NO3:long_name = "NO3 concentration" ;
       NO3:units = "1e-9" ;
       NO3:coordinates = "time lon lat z" ;

attributes:
   :featureType = "trajectory";
```

The NO3(i,o) and O3(i,o) data for element o of trajectory i are associated with the coordinate values time(i,o), lat(i,o), lon(i,o), and z(i,o). Either the instance (trajectory) or the element (obs) dimension could be the netCDF unlimited dimension. All variables that have trajectory as their

only dimension are considered to be information about that trajectory.

If the trajectories all have the same set of times, the time auxiliary coordinate variable could be one-dimensional time(obs), or replaced by a one-dimensional coordinate variable time(time), where the size of the time dimension is now equal to the number of elements of each trajectory. In the latter case, listing the time coordinate variable in the coordinates attribute is optional.

H.4.2. Single trajectory

When a single trajectory is stored in the data variable, there is no need for the trajectory dimension and the arrays are one-dimensional. This is a special case of the multidimensional array representation.

Example H.13. A single trajectory recording atmospheric composition.

```
dimensions:
   time = 42;

variables:
   char trajectory(name_strlen) ;
       trajectory:cf_role = "trajectory_id";

   double time(time) ;
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon(time) ;
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
   float lat(time) ;
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;
   float z(time) ;
       z:standard_name = “altitude”;
       z:long_name = "height above mean sea level" ;
       z:units = "km" ;
       z:positive = "up" ;
        z:axis = "Z" ;

   float O3(time) ;
       O3:standard_name = “mass_fraction_of_ozone_in_air”;
       O3:long_name = "ozone concentration" ;
       O3:units = "1e-9" ;
       O3:coordinates = "time lon lat z" ;

   float NO3(time) ;
       NO3:standard_name = “mass_fraction_of_nitrate_radical_in_air”;
       NO3:long_name = "NO3 concentration" ;
       NO3:units = "1e-9" ;
       NO3:coordinates = "time lon lat z" ;

attributes:
   :featureType = "trajectory";
```

The NO3(o) and O3(o) data are associated with the coordinate values time(o), z(o), lat(o), and lon(o). In this example, the time coordinate is ordered, so time values are contained in a coordinate variable i.e. time(time) and time is the element dimension. The time dimension may be unlimited or not.

Note that structurally this looks like unconnected point data as in example 9.5. The presence of the featureType = "trajectory" global attribute indicates that in fact the points are connected along a trajectory.

H.4.3. Contiguous ragged array representation of trajectories

When the number of elements for each trajectory varies, and one can control the order of writing, one can use the contiguous ragged array representation. The canonical use case for this is when rewriting raw data, and you expect that the common read pattern will be to read all the data from each trajectory.

Example H.14. Trajectories recording atmospheric composition in the contiguous ragged array representation.

```
dimensions:
   obs = 3443;
   trajectory = 77 ;

variables:
   char trajectory(trajectory, name_strlen) ;
         trajectory:cf_role = "trajectory_id";
   int rowSize(trajectory) ;
       rowSize:long_name = "number of obs for this trajectory " ;
       rowSize:sample_dimension = "obs" ;

   double time(obs) ;
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon(obs) ;
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
   float lat(obs) ;
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;
   float z(obs) ;
       z:standard_name = “altitude”;
       z:long_name = "height above mean sea level" ;
       z:units = "km" ;
       z:positive = "up" ;
        z:axis = "Z" ;

   float O3(obs) ;
       O3:standard_name = “mass_fraction_of_ozone_in_air”;
       O3:long_name = "ozone concentration" ;
       O3:units = "1e-9" ;
       O3:coordinates = "time lon lat z" ;

   float NO3(obs) ;
       NO3:standard_name = “mass_fraction_of_nitrate_radical_in_air”;
       NO3:long_name = "NO3 concentration" ;
       NO3:units = "1e-9" ;
       NO3:coordinates = "time lon lat z" ;

attributes:
   :featureType = "trajectory";
```

The O3(o) and NO3(o) data are associated with the coordinate values time(o), lat(o), lon(o), and alt(o). All elements for one trajectory are contiguous along the sample dimension. The sample dimension (obs) may be the unlimited dimension or not. All variables that have the instance dimension (trajectory) as their single dimension are considered to be information about that

trajectory.

The count variable (row_size) contains the number of elements for each trajectory, and is identified by having an attribute with name "sample_dimension" whose value is the sample dimension being counted. It must have the trajectory dimension as its single dimension, and must be type integer. The elements are associated with the trajectories using the same algorithm as in H.2.4.

H.4.4. Indexed ragged array representation of trajectories

When the number of elements at each trajectory vary, and the elements cannot be written in order, one can use the indexed ragged array representation. The canonical use case is when writing real-time data streams that contain reports from many trajectories. The data can be written as it arrives; if the flatsample dimension is the unlimited dimension, this allows data to be appended to the file.

Example H.15. Trajectories recording atmospheric composition in the indexed ragged array representation.

```
   dimensions:
      obs = UNLIMITED ;
      trajectory = 77 ;

   variables:
      char trajectory(trajectory, name_strlen) ;
          trajectory:cf_role = "trajectory_id";

      int trajectory_index(obs) ;
          trajectory_index:long_name = "index of trajectory this obs belongs to "
;
          trajectory_index:instance_dimension= "trajectory" ;
      double time(obs) ;
          time:standard_name = "time";
          time:long_name = "time" ;
          time:units = "days since 1970-01-01 00:00:00" ;
      float lon(obs) ;
          lon:standard_name = "longitude";
          lon:long_name = "longitude" ;
          lon:units = "degrees_east" ;
      float lat(obs) ;
          lat:standard_name = "latitude";
          lat:long_name = "latitude" ;
          lat:units = "degrees_north" ;
      float z(obs) ;
          z:standard_name = “altitude”;
          z:long_name = "height above mean sea level" ;
          z:units = "km" ;
          z:positive = "up" ;
          z:axis = "Z" ;

      float O3(obs) ;
          O3:standard_name = “mass_fraction_of_ozone_in_air”;
          O3:long_name = "ozone concentration" ;
          O3:units = "1e-9" ;
          O3:coordinates = "time lon lat z" ;

      float NO3(obs) ;
          NO3:standard_name = “mass_fraction_of_nitrate_radical_in_air”;
          NO3:long_name = "NO3 concentration" ;
          NO3:units = "1e-9" ;
          NO3:coordinates = "time lon lat z" ;

   attributes:
      :featureType = "trajectory";
```

The O3(o) and NO3(o) data are associated with the coordinate values time(o), lat(o), lon(o), and alt(o). All elements for one trajectory will have the same trajectory index value. The sample dimension (obs) may be the unlimited dimension or not.

The index variable (trajectory_index) is identified by having an attribute with name of "instance_dimension" whose value is the trajectory dimension name. It must have the sample dimension as its single dimension, and must be type integer. Each value in the trajectory_index variable is the zero-based trajectory index that the element belongs to. The elements are associated with the trajectories using the same algorithm as in H.2.5.

H.5. Time Series of Profiles

When profiles are taken repeatedly at a station, one gets a time series of profiles (see also section H.2 for discussion of stations and time series). The resulting collection of profiles is called a timeSeriesProfile. A data variable may contain a collection of such timeSeriesProfile features, one feature per station. The instance dimension in the case of a timeSeriesProfile is also referred to as the **station dimension** . The instance variables, which have just this dimension, including latitude and longitude for example, are also referred to as **station variables** and are considered to contain information describing the stations. The station variables may contain missing values. This allows one to reserve space for additional stations that may be added at a later time, as discussed in section 9.6. In addition,

- It is strongly recommended that there should be a station variable (which may be of any type) with `cf_role` attribute "`timeseries_id`", whose values uniquely identify the stations.
- It is recommended that there should be station variables with standard_name attributes "`platform_name`", "`surface_altitude`" and “`platform_id`” when applicable.

TimeSeriesProfiles are more complicated than timeSeries because there are two element dimensions (profile and vertical). Each time series has a number of profiles from different times as its elements, and each profile has a number of data from various levels as its elements. It is strongly recommended that there always be a variable (of any data type) with the profile dimension and the `cf_role` attribute " `profile_id` ", whose values uniquely identify the profiles.

H.5.1. Multidimensional array representations of time series profiles

When storing time series of profiles at multiple stations in the same data variable, if there are the same number of time points for all timeSeries, and the same number of vertical levels for every profile, one can use the multidimensional array representation:

Example H.16. Time series of atmospheric sounding profiles from a set of locations stored in a multidimensional array representation.

```
dimensions:
   station = 22 ;
   profile = 3002 ;
   z = 42 ;

variables:
   float lon(station) ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
```

```
        lon:units = "degrees_east";
    float lat(station) ;
        lat:standard_name = "latitude";
        lat:long_name = "station latitude" ;
        lat:units = "degrees_north" ;
    char station_name(station, name_strlen) ;
        station_name:cf_role = "timeseries_id" ;
        station_name:long_name = "station name" ;
    int station_info(station) ;
        station_name:long_name = "some kind of station info" ;

    float alt(station, profile , z) ;
        alt:standard_name = “altitude”;
        alt:long_name = "height above mean sea level" ;
        alt:units = "km" ;
        alt:positive = "up" ;
         alt:axis = "Z" ;

    double time(station, profile ) ;
        time:standard_name = "time";
        time:long_name = "time of measurement" ;
        time:units = "days since 1970-01-01 00:00:00" ;
        time:missing_value = -999.9;

    float pressure(station, profile , z) ;
        pressure:standard_name = "air_pressure" ;
        pressure:long_name = "pressure level" ;
        pressure:units = "hPa" ;
        pressure:coordinates = "time lon lat alt" ;

    float temperature(station, profile , z) ;
        temperature:standard_name = "surface_temperature" ;
        temperature:long_name = "skin temperature" ;
        temperature:units = "Celsius" ;
        temperature:coordinates = "time lon lat alt" ;

    float humidity(station, profile , z) ;
        humidity:standard_name = "relative_humidity" ;
        humidity:long_name = "relative humidity" ;
        humidity:units = "%" ;
        humidity:coordinates = "time lon lat alt" ;

attributes:
 :featureType = "timeSeriesProfile";
```

The pressure(i,p,o), temperature(i,p,o), and humidity(i,p,o) data for element o of profile p at station i are associated with the coordinate values time(i,p), z(i,p,o), lat(i), and lon(i). Any of the three dimensions could be the netCDF unlimited dimension, if it might be useful to be able enlarge it.

If all of the profiles at any given station have the same set of vertical coordinates values, the vertical auxiliary coordinate variable could be dimensioned alt(station, z). If all the profiles have the same set of vertical coordinates, the vertical auxiliary coordinate variable could be one-dimensional alt(z), or replaced by a one-dimensional coordinate variable z(z), provided the values are ordered monotonically. In the latter case, listing the vertical coordinate variable in the coordinates attribute is optional.

If the profiles are taken at all stations at the same set of times, the time auxiliary coordinate variable could be one-dimensional time(profile), or replaced by a one-dimensional coordinate variable time(time), where the size of the time dimension is now equal to the number of profiles at each station. In the latter case, listing the time coordinate variable in the coordinates attribute is optional.

If there is only a single set of levels and a single set of times, the multidimensional array representation is formally orthogonal:

Example H.17. Time series of atmospheric sounding profiles from a set of locations stored in an orthogonal multidimensional array representation.

```
dimensions:
  station = 10 ;  // measurement locations
  pressure = 11 ; // pressure levels
  time = UNLIMITED ;
variables:
  float humidity(time,pressure,station) ;
    humidity:standard_name = “specific_humidity” ;
    humidity:coordinates = "lat lon" ;
  double time(time) ;
    time:standard_name = "time";
    time:long_name = "time of measurement" ;
    time:units = "days since 1970-01-01 00:00:00" ;
  float lon(station) ;
    lon:long_name = "station longitude";
    lon:units = "degrees_east";
  float lat(station) ;
    lat:long_name = "station latitude" ;
    lat:units = "degrees_north" ;
  float pressure(pressure) ;
    pressure:standard_name = "air_pressure" ;
    pressure:long_name = "pressure" ;
    pressure:units = "hPa" ;
    pressure:axis = "Z" ;
```

`humidity(p,o,i)` is associated with the coordinate values `time(p)` , `pressure(o)` , `lat(i)` , and `lon(i)` . The number of profiles equals the number of times.

At the cost of some wasted space, the multidimensional array representation also allows one to

have a variable number of profiles for different stations, and varying numbers of levels for different profiles. In these cases, any unused elements of the data and auxiliary coordinate variables must contain missing data values (section 9.6).

H.5.2. Time series of profiles at a single station

If there is only one station in the data variable, there is no need for the station dimension:

Example H.18. Time series of atmospheric sounding profiles from a single location stored in a multidimensional array representation.

```
dimensions:
   profile = 30 ;
   z = 42 ;

variables:
   float lon ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
       lon:units = "degrees_east";
   float lat ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;
   char station_name(name_strlen) ;
       station_name:cf_role = "timeseries_id" ;
       station_name:long_name = "station name" ;
   int station_info;
       station_name:long_name = "some kind of station info" ;

   float alt(profile , z) ;
       alt:standard_name = “altitude”;
       alt:long_name = "height above mean sea level" ;
       alt:units = "km" ;
       alt:axis = "Z" ;
       alt:positive = "up" ;

   double time(profile ) ;
       time:standard_name = "time";
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
       time:missing_value = -999.9;

   float pressure(profile , z) ;
       pressure:standard_name = "air_pressure" ;
       pressure:long_name = "pressure level" ;
       pressure:units = "hPa" ;
       pressure:coordinates = "time lon lat alt" ;

   float temperature(profile , z) ;
```

```
        temperature:standard_name = "surface_temperature" ;
        temperature:long_name = "skin temperature" ;
        temperature:units = "Celsius" ;
        temperature:coordinates = "time lon lat alt" ;

   float humidity(profile , z) ;
        humidity:standard_name = "relative_humidity" ;
        humidity:long_name = "relative humidity" ;
        humidity:units = "%" ;
        humidity:coordinates = "time lon lat alt" ;

attributes:
 :featureType = "timeSeriesProfile";
```

The pressure(p,o), temperature(p,o), and humidity(p,o) data for element o of profile p are associated with the coordinate values time(p), alt(p,o), lat, and lon. If all the profiles have the same set of vertical coordinates, the vertical auxiliary coordinate variable could be one-dimensional alt(z), or replaced by a one-dimensional coordinate variable z(z), provided the values are ordered monotonically. In the latter case, listing the vertical coordinate variable in the coordinates attribute is optional.

H.5.3. Ragged array representation of time series profiles

When the number of profiles and levels for each station varies, one can use a ragged array representation. Each of the two element dimensions (time and vertical) could in principle be stored either contiguous or indexed, but this convention supports only one of the four possible choices. This uses the contiguous ragged array representation for each profile (9.5.43.3), and the indexed ragged array representation to organise the profiles into time series (9.3.54). The canonical use case is when writing real-time data streams that contain profiles from many stations, arriving randomly, with the data for each entire profile written all at once.

Example H.19. Time series of atmospheric sounding profiles from a set of locations stored in a ragged array representation.

```
dimensions:
   obs = UNLIMITED ;
   profiles = 1420 ;
   stations = 42;

variables:
   float lon(station) ;
       lon:standard_name = "longitude";
       lon:long_name = "station longitude";
       lon:units = "degrees_east";
   float lat(station) ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;
```

```
    float alt(station) ;
        alt:long_name = "altitude above MSL" ;
        alt:units = "m" ;
    char station_name(station, name_strlen) ;
        station_name:long_name = "station name" ;
        station_name:cf_role = "timeseries_id";
    int station_info(station) ;
        station_info:long_name = "some kind of station info" ;

    int profile(profile) ;
        profile:cf_role = "profile_id";
    double time(profile);
        time:standard_name = "time";
        time:long_name = "time" ;
        time:units = "days since 1970-01-01 00:00:00" ;
    int station_index(profile) ;
        station_index:long_name = "which station this profile is for" ;
        station_index:instance_dimension = "station" ;
    int row_size(profile) ;
        row_size:long_name = "number of obs for this profile " ;
        row_size:sample_dimension = "obs" ;

    float z(obs) ;
        z:standard_name = “altitude”;
        z:long_name = "height above mean sea level" ;
        z:units = "km" ;
        z:axis = "Z" ;
         z:positive = "up" ;

    float pressure(obs) ;
        pressure:standard_name = "air_pressure" ;
        pressure:long_name = "pressure level" ;
        pressure:units = "hPa" ;
        pressure:coordinates = "time lon lat z" ;

    float temperature(obs) ;
        temperature:standard_name = "surface_temperature" ;
        temperature:long_name = "skin temperature" ;
        temperature:units = "Celsius" ;
        temperature:coordinates = "time lon lat z" ;

    float humidity(obs) ;
        humidity:standard_name = "relative_humidity" ;
        humidity:long_name = "relative humidity" ;
        humidity:units = "%" ;
        humidity:coordinates = "time lon lat z" ;

  attributes:
    :featureType = "timeSeriesProfile";
```

The pressure(o), temperature(o), and humidity(o) data for element o of profile p at station i are associated with the coordinate values time(p), z(o), lat(i), and lon(i).

The index variable (station_index) is identified by having an attribute with name of instance_dimension whose value is the instance dimension name (station in this example). The index variable must have the profile dimension as its sole dimension, and must be type integer. Each value in the index variable is the zero-based station index that the profile belongs to i.e. profile p belongs to station i=station_index(p), as in section H.2.5.

The count variable (row_size) contains the number of elements for each profile, which must be written contiguously. The count variable is identified by having an attribute with name sample_dimension whose value is the sample dimension (obs in this example) being counted. It must have the profile dimension as its sole dimension, and must be type integer. The number of elements in profile p is recorded in row_size(p), as in section H.2.4. The sample dimension need not be the netCDF unlimited dimension, though it commonly is.

H.6. Trajectory of Profiles

When profiles are taken along a trajectory, one gets a collection of profiles called a trajectoryProfile. A data variable may contain a collection of such trajectoryProfile features, one feature per trajectory. The instance dimension in the case of a trajectoryProfile is also referred to as the **trajectory dimension** . The instance variables, which have just this dimension, are also referred to as **trajectory variables** and are considered to contain information describing the trajectories. The trajectory variables may contain missing values. This allows one to reserve space for additional trajectories that may be added at a later time, as discussed in section 9.6. TrajectoryProfiles are more complicated than trajectories because there are two element dimensions. Each trajectory has a number of profiles as its elements, and each profile has a number of data from various levels as its elements. It is strongly recommended that there always be a variable (of any data type) with the profile dimension and the `cf_role` attribute " `profile_id` ", whose values uniquely identify the profiles.

H.6.1. Multidimensional array representation of trajectory profiles

If there are the same number of profiles for all trajectories, and the same number of vertical levels for every profile, one can use the multidimensional representation:

Example H.20. Time series of atmospheric sounding profiles along a set of trajectories stored in a multidimensional array representation.

```
dimensions:
   trajectory = 22 ;
   profile = 33;
   z = 42 ;

variables:
   int trajectory (trajectory ) ;
       trajectory:cf_role = "trajectory_id" ;
```

```
      float lon(trajectory, profile) ;
          lon:standard_name = "longitude";
          lon:units = "degrees_east";
      float lat(trajectory, profile) ;
          lat:standard_name = "latitude";
          lat:long_name = "station latitude" ;
          lat:units = "degrees_north" ;

      float alt(trajectory, profile , z) ;
          alt:standard_name = “altitude”;
          alt:long_name = "height above mean sea level" ;
          alt:units = "km" ;
          alt:positive = "up" ;
          alt:axis = "Z" ;

      double time(trajectory, profile ) ;
          time:standard_name = "time";
          time:long_name = "time of measurement" ;
          time:units = "days since 1970-01-01 00:00:00" ;
          time:missing_value = -999.9;

      float pressure(trajectory, profile , z) ;
          pressure:standard_name = "air_pressure" ;
          pressure:long_name = "pressure level" ;
          pressure:units = "hPa" ;
          pressure:coordinates = "time lon lat alt" ;

      float temperature(trajectory, profile , z) ;
          temperature:standard_name = "surface_temperature" ;
          temperature:long_name = "skin temperature" ;
          temperature:units = "Celsius" ;
          temperature:coordinates = "time lon lat alt" ;

      float humidity(trajectory, profile , z) ;
          humidity:standard_name = "relative_humidity" ;
          humidity:long_name = "relative humidity" ;
          humidity:units = "%" ;
          humidity:coordinates = "time lon lat alt" ;

   attributes:
    :featureType = "trajectoryProfile";
```

The pressure(i,p,o), temperature(i,p,o), and humidity(i,p,o) data for element o of profile p along trajectory i are associated with the coordinate values time(i,p), alt(i,p,o), lat(i,p), and lon(i,p). Any of the three dimensions could be the netCDF unlimited dimension, if it might be useful to be able enlarge it.

If all of the profiles along any given trajectory have the same set of vertical coordinates values, the vertical auxiliary coordinate variable could be dimensioned alt(trajectory, z). If all the

profiles have the same set of vertical coordinates, the vertical auxiliary coordinate variable could be one-dimensional alt(z), or replaced by a one-dimensional coordinate variable z(z), provided the values are ordered monotonically. In the latter case, listing the vertical coordinate variable in the coordinates attribute is optional.

If the profiles are taken along all the trajectories at the same set of times, the time auxiliary coordinate variable could be one-dimensional time(profile), or replaced by a one-dimensional coordinate variable time(time), where the size of the time dimension is now equal to the number of profiles along each trajectory. In the latter case, listing the time coordinate variable in the coordinates attribute is optional.

At the cost of some wasted space, the multidimensional array representation also allows one to have a variable number of profiles for different trajectories, and varying numbers of levels for different profiles. In these cases, any unused elements of the data and auxiliary coordinate variables must contain missing data values (section 9.6).

H.6.2. Profiles along a single trajectory

If there is only one trajectory in the data variable, there is no need for the trajectory dimension:

Example H.21. Time series of atmospheric sounding profiles along a trajectory stored in a multidimensional array representation.

```
dimensions:
   profile = 33;
   z = 42 ;

variables:
   int trajectory;
       trajectory:cf_role = "trajectory_id" ;

   float lon(profile) ;
       lon:standard_name = "longitude";
       lon:units = "degrees_east";
   float lat(profile) ;
       lat:standard_name = "latitude";
       lat:long_name = "station latitude" ;
       lat:units = "degrees_north" ;

   float alt(profile, z) ;
       alt:standard_name = “altitude”;
       alt:long_name = "height above mean sea level" ;
       alt:units = "km" ;
       alt:positive = "up" ;
        alt:axis = "Z" ;

   double time(profile ) ;
       time:standard_name = "time";
       time:long_name = "time of measurement" ;
       time:units = "days since 1970-01-01 00:00:00" ;
       time:missing_value = -999.9;

   float pressure(profile, z) ;
       pressure:standard_name = "air_pressure" ;
       pressure:long_name = "pressure level" ;
       pressure:units = "hPa" ;
       pressure:coordinates = "time lon lat alt" ;

   float temperature(profile, z) ;
       temperature:standard_name = "surface_temperature" ;
       temperature:long_name = "skin temperature" ;
       temperature:units = "Celsius" ;
       temperature:coordinates = "time lon lat alt" ;

   float humidity(profile, z) ;
       humidity:standard_name = "relative_humidity" ;
       humidity:long_name = "relative humidity" ;
       humidity:units = "%" ;
       humidity:coordinates = "time lon lat alt" ;

attributes:
 :featureType = "trajectoryProfile";
```

The pressure(p,o), temperature(p,o), and humidity(p,o) data for element o of profile p are associated with the coordinate values time(p), alt(p,o), lat(p), and lon(p). If all the profiles have the same set of vertical coordinates, the vertical auxiliary coordinate variable could be one-dimensional alt(z), or replaced by a one-dimensional coordinate variable z(z), provided the values are ordered monotonically. In the latter case, listing the vertical coordinate variable in the coordinates attribute is optional.

H.6.3. Ragged array representation of trajectory profiles

When the number of profiles and levels for each trajectory varies, one can use a ragged array representation. Each of the two element dimensions (along a projectory, within a profile) could in principle be stored either contiguous or indexed, but this convention supports only one of the four possible choices. This uses the contiguous ragged array representation for each profile (9.3.3), and the indexed ragged array representation to organise the profiles into time series (9.3.4). The canonical use case is when writing real-time data streams that contain profiles from many trajectories, arriving randomly, with the data for each entire profile written all at once.

Example H.22. Time series of atmospheric sounding profiles along a set of trajectories stored in a ragged array representation.

```
dimensions:
   obs = UNLIMITED ;
   profiles = 142 ;
   section = 3;

variables:
   int trajectory(trajectory) ;
       section:standard_namecf_role = "trajectory_id" ;

   double time(profile);
       time:standard_name = "time";
       time:long_name = "time" ;
       time:units = "days since 1970-01-01 00:00:00" ;
   float lon(profile);
       lon:standard_name = "longitude";
       lon:long_name = "longitude" ;
       lon:units = "degrees_east" ;
   float lat(profile);
       lat:standard_name = "latitude";
       lat:long_name = "latitude" ;
       lat:units = "degrees_north" ;
   int row_size(profile) ;
       row_size:long_name = "number of obs for this profile " ;
       row_size:sample_dimension = "obs" ;
   int trajectory_index(profile) ;
       trajectory_index:long_name = "which trajectory this profile is for" ;
       trajectory_index:instance_dimension= "trajectory" ;
```

```
        float z(obs) ;
           z:standard_name = “altitude”;
           z:long_name = "height above mean sea level" ;
           z:units = "km" ;
           z:positive = "up" ;
           z:axis = "Z" ;

       float pressure(obs) ;
           pressure:standard_name = "air_pressure" ;
           pressure:long_name = "pressure level" ;
           pressure:units = "hPa" ;
           pressure:coordinates = "time lon lat z" ;

       float temperature(obs) ;
           temperature:standard_name = "surface_temperature" ;
           temperature:long_name = "skin temperature" ;
           temperature:units = "Celsius" ;
           temperature:coordinates = "time lon lat z" ;

       float humidity(obs) ;
           humidity:standard_name = "relative_humidity" ;
           humidity:long_name = "relative humidity" ;
           humidity:units = "%" ;
           humidity:coordinates = "time lon lat z" ;

    attributes:
       :featureType = "trajectoryProfile";
```

The pressure(o), temperature(o), and humidity(o) data for element o of profile p along trajectory i are associated with the coordinate values time(p), z(o), lat(p), and lon(p).

The index variable (trajectory_index) is identified by having an attribute with name of instance_dimension whose value is the instance dimension name (trajectory in this example). The index variable must have the profile dimension as its sole dimension, and must be type integer. Each value in the index variable is the zero-based trajectory index that the profile belongs to i.e. profile p belongs to trajectory i=trajectory_index(p), as in section H.2.5.

The count variable (row_size) contains the number of elements for each profile, which must be written contiguously. The count variable is identified by having an attribute with name sample_dimension whose value is the sample dimension (obs in this example) being counted. It must have the profile dimension as its sole dimension, and must be type integer. The number of elements in profile p is recorded in row_size(p), as in section H.2.4. The sample dimension need not be the netCDF unlimited dimension, though it commonly is.

Bibliography

References

- [COARDS] Conventions for the standardization of NetCDF Files . Sponsored by the "Cooperative Ocean/Atmosphere Research Data Service," a NOAA/university cooperative for the sharing and distribution of global atmospheric and oceanographic research data sets . May 1995.
- [FGDC] Content Standard for Digital Geospatial Metadata . Federal Geographic Data Committee, FGDC-STD-001-1998 .
- [NetCDF] NetCDF Software Package . UNIDATA Program Center of the University Corporation for Atmospheric Research .
- [NUG] NetCDF User's Guide for Fortran: An Access Interface for Self-Describing Portable Data; version 3 . Russ Rew, Glenn Davis, Steve Emmerson, and Harvey Davies. June 1997.
- [OGP-EPSG] OGP Surveying & Positioning Committee and EPSG Geodetic Parameter Registry .
- [OGP-EPSG_GN7_2] OGP Surveying and Positioning Guidance Note 7, part 2: Coordinate Conversions and Transformations including Formulas .
- [SCH02] C Schaer, D Leuenberger, and O Fuhrer. 2002. " A new terrain-following vertical coordiante formulation for atmospheric prediction models ". *Monthly Weather Review* . 130 . 2459-2480.
- [Snyder] Map Projections: A Working Manual . USGS Professional Paper 1395.
- [UDUNITS] UDUNITS Software Package . UNIDATA Program Center of the University Corporation for Atmospheric Research .
- [W3C] World Wide Web Consortium (W3C) .
- [XML] Extensible Markup Language (XML) 1.0 . T. Bray, J. Paoli, and C.M. Sperberg-McQueen. 10 February 1998 .