Proposed netCDF conventions for climate data

Jonathan Gregory[1], Bob Drach[2] and Simon Tett[1]
(1) Hadley Centre, UK Met Office; (2) PCMDI, temporarily at the Hadley Centre

10th June 1997

1 Purposes

This standard defines a set of conventions adopted in order to promote the interchange and sharing of files created with the netCDF Application Programmer Interface (API). This standard is based upon version 2.4 of netCDF. Documentation of the netCDF API may be found in the "NetCDF Users' Guide", Version 2.4, February 1996, available from URL `http://www.unidata.ucar.edu/packages/netcdf/` or via anonymous ftp at `ftp.unidata.ucar.edu`.

This standard is intended for use with climate data, and was designed with data generated by GCMs particularly in mind. We recognise that there are limits to what a standard can practically cover; we restrict ourselves to issues which we believe to be of common and frequent concern in the design of climate metadata. This standard is mostly additional to the conventions sponsored by COARDS (`ftp://ftp.unidata.ucar.edu/netcdf/-Conventions/COARDS`). In the following standard, parts which are identical to or paraphrases of the COARDS standard are prefixed with COARDS, and new material with NEW. Material from the Unidata netCDF users' guide is marked UNIDATA. All Unidata recommendations are supported here unless noted to the contrary. Comments indicate the places where there are differences between the standards. Comments and examples are not part of the standard, and are given in *emphasised text.*

COARDS: This standard also refers to the udunits standard supported by Unidata. The udunits package is available via anonymous ftp at `ftp.unidata.ucar.edu`. NEW: See section 11 for details of how the package is used by this convention to define units for physical quantities.

2 Filename

COARDS: NetCDF files should have the file name extension `.nc`.

3 Data types

NEW: The netCDF data types `char`, `short`, `long`, `float`, and `double` are all acceptable. All numeric types are signed. The `byte` data type, which is functionally identical to `char`, is not recommended because its signedness may become ambiguous in future versions of netCDF. *The COARDS convention deprecates* `char`*, rather than* `byte`*.*

NEW: NetCDF does not support a character string type, so these have to be represented as `char` arrays. In this standard, we refer to them as type "string". A string array must be implemented as a two-dimensional character data variable, serving as a vector of fixed-length strings, the second dimension of its CDL declaration *(leading dimension in terms of Fortran)* being a constant `StringMaxLength`, recorded as a dimension in the netCDF file.

4 Attributes

COARDS: 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. UNIDATA: 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. NEW: When this standard defines string attributes which make take various prescribed values, the possible values are given in lower case. However, applications programs should not be sensitive to case in these attributes. *See Appendix A for a list of attributes described by this standard.*

5 Global attributes

COARDS: Although not mandatory, the Unidata-standard attribute `history` is recommended to record the evolution of the data contained within a netCDF file. Applications which process netCDF data can append their information to the `history` attribute.

COARDS: The Unidata-standard attribute `Conventions` is recommended to reference this standard.

NEW: Use of the string attributes `institution` and `production` is recommended. The attribute `institution` specifies who produced or supplied the data. *We prefer this name to "center" or "centre" because the two possible spellings could cause confusion.* The attribute `production` indicates how the data was produced. If it was model-generated, `production` should name the model and its version, as specifically as could be useful. If it is observational, `production` should characterise it *e.g.* `"surface observation"` *or* `"radiosonde"`. A further string attribute `comment` is proposed, to contain extra information about the file, and additional attributes may be included as required. *The Hadley Centre, for example, will include an attribute* `source` *to name the model integration.*

NEW: The `float` attribute `appendices` is recommended to record the version number of the appendices to this standard used by the application which generated the file (see section 12).

NEW: The `calendar` attribute (see section 24) may be recorded as a global attribute. The global `calendar` attribute is interpreted as a default for all time axes.

6 Variable names

COARDS: Variable names should begin with a letter and be composed of letters, digits, and underscores. NEW: 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.

7 Data variables

NEW: The netCDF variables which contain the physical data are referred to as "data variables", also referred to as "primary variables" by Unidata. Apart from the general naming rules for variables (above, section 6), the names of data variables are not standardised by these conventions (since files may in general contain multiple data variables of the same physical quantity).

8 Axes and dimensionality of a data variable

NEW: A data variable may have any number of dimensions, including zero, and the dimensions must all have different names. *COARDS strongly recommends limiting the number 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. *One example is a dimension in electromagnetic radiation wavelength for a data variable of radiative flux. Components of vector or tensor quantities could be contained in a single data variable by giving the variable a dimension over components. While there exist advantages for manipulating such a variable in memory, we see no strong advantage in introducing this complexity into the netCDF description, and do not recommend it.* Under certain circumstances, one may need more than one dimension in a particular quantity (see section 29 concerning multiple time axes). *For instance, a data 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.*

COARDS: If any or all of the dimensions of a data variable have the interpretations of "date or time" (`T`), "height or depth" (`Z`), "latitude" (`Y`), or "longitude" (`X`) then those dimensions should appear in the relative order `T`, then `Z`, then `Y`, then `X` in the CDL definition corresponding to the file. *N.B. In terms of Fortran, this means* `X` *is the first dimension of the array.* Non-spatiotemporal dimensions should be placed to the left of the spatiotemporal dimensions *i.e. as trailing dimensions in terms of Fortran.*

NEW: Each dimension of a data variable must have a coordinate variable supplying the coordinates of the points along that axis (section 9). The coordinates of points within the data variable are the simple ordered tuples formed by associating values from the coordinate variables.

NEW: Dimensions may be of any size, including unity. When a single value of some physical quantity applies to all the values in a data variable, the recommended means of attaching this information to the variable is by use of a singleton dimension (a dimension of size unity) with a one-element coordinate variable. *For example, a data variable on a pressure level would use a singleton pressure dimension to record the level.* Singleton dimensions also result from contractions, described in section 23.

9 Coordinate variables

NEW: A one-dimensional netCDF variable associated with an axis of one or more data variables is called a "coordinate variable". A coordinate variable whose dimension name is identical to its own name is referred to as a "main coordinate variable" in this standard, when it is necessary to distinguish it from other types of coordinate variable (sections 17 and 21). Each axis of a data variable must have a main coordinate variable, but other types are optional. Apart from the general naming rules for variables (above, section 6), the names of coordinate variables are not standardised by these conventions (since files may in general contain multiple coordinate variables of the same orientation).

UNIDATA: The values in a main coordinate variable must be strictly monotonic (all values are different and either increasing or decreasing). NEW: If a main coordinate variable is merely a "dummy", not containing any physical information, its values should be set equal to their indices (0, 1, 2, ...). *This is the netCDF default for an omitted coordinate variable. This standard requires it to be made explicit.*

10 Coordinate systems

NEW: A coordinate system for the Earth's surface which is rectilinear but based on a polar axis other than the normal geographical axis is referred to as a "rotated grid". To describe rotated grids, a two-element `float` attribute `north_pole` is attached to the data variable, specifying the (longitude,latitude) coordinates of the rotated north pole. If the attribute is absent and relevant, it is assumed to have the value (0.,90.) i.e. the geographical north pole.

NEW: In some systems, the axes covering the Earth's surface do not define a rectilinear grid. We do not wish necessarily to exclude non-rectilinear systems. For the moment, this standard is undefined for these systems, and we invite comments from potential users on the appropriate definition. *The COARDS standard excludes non-rectilinear systems. In principle, any coordinate system can be handled, albeit clumsily, by replacing the relevant two or more axes by by a single axis which indexes the points, and providing ancillary coordinate variables to specify the coordinates, point by point (see section 17).*

11 Units

COARDS: The udunits package includes a file `udunits.dat`, which lists collections of unit names. The names given in the most recent version of this file and their plural forms will be regarded as acceptable unit names for this standard, with a few modifications which will be listed in Appendix C to this standard. *COARDS lists some modifications within the standard, but we would prefer to put in place a means to allow future modifications to be made easily.* NEW: Users of this standard should not define their own units, because this would make their files less portable; requests for new units should be directed to Unidata.

NEW: The udunits package also defines a means for linear transformation of units by a scale factor and an offset. This convention is allowed when it is natural to express a unit in such a form *e.g. density of sea-water in* $kg\,m^{-3}$ *in excess of 1000* $kg\,m^{-3}$, *which can be specified to udunits as* `"kg m-3 @ 1000"`. *COARDS does not permit the use of this facility.* It can also be used to specify a scale factor and/or offset for a dimensionless

quantity *e.g. sea-ice concentration in tenths (no unit).* It should not be used as a means of data compression, for which an alternative is provided (see section 34).

12 Physical quantity of a variable

NEW: These conventions standardise three attributes for specifying the physical quantity of data and coordinate variables. All of them are strings:

1. `units`, formatted as per the recommendations in the Unidata udunits package, with extensions for time (see section 26). This attribute is mandatory unless the quantity is dimensionless (a pure number). Dimensionless quantities will not generally have units; a unit string specifying a linear transformation alone is permissible. *For instance, a unit of tenths is given as* `"0.1"`. *There are a few defined dimensionless units, such as* `percent`. *There is no need for a wide variety of dimensionless units for quantities like sea-ice concentration, cloud fraction, probability and so on; this descriptive information is the* `quantity` *rather than the* `units`.

2. `quantity`, a short descriptive name. The `quantity` attribute is optional for most variables, but mandatory for coordinate variables of longitude, latitude, vertical axes and time (see sections 14, 15, 16 and 24). If the `quantity` attribute is present, it must be one from a list which constitutes Appendix E of this standard *(see below).* The use of a standard list aims to clarify whether data from different sources is comparable. The list will include all quantities which are used as coordinate variables. The list will also define the physical dimensions of the quantity by stating an appropriate unit. The `units` and the `quantity` of the data variable must be compatible. Appendix E will indicate the version number of the appendices at which each quantity was introduced. In conjunction with the `appendices` attribute (section 5), this enables the application to deduce the entire set of distinct quantities known to the application which generated the file.

3. `long_name`, a long descriptive name. It should not include the `units`.

The `long_name` is useful as the title of a plot of a data variable, or the title of the axis of a coordinate variable. In its absence, the `quantity` may be used, or, failing that, the name of the variable itself. In addition, other model-dependent attributes may included to define the quantity of a variable. *The Hadley Centre model will give each data variable integer* `stash` *and* `submodel` *attributes, for example.*

Whether two physical quantities are different or the same is often not a question with a well-defined answer. Certainly if they are the same, they must have the same unit, but various quantities with the same unit may have to be distinguished e.g. `temperature` *and* `soil temperature`. *In practice, the most specific description applicable should be used. We intend to expand the list of quantities in Appendix E on an ongoing basis in response to requests by users of this standard, since we cannot foresee all the possibilities, and we will err on the side of expansion, rather than restriction, when it is unclear whether a newly requested quantity is identical to one already listed. However, quantities which differ only with regard to surfaces (sections 30), linear unweighted contractions (section 23) or other distinctions which can represented simply by attributes described in later sections will not be regarded as distinct physical quantities e.g. net downward shortwave radiation at the surface, and net downward shortwave radiation at the top of the atmosphere.*

NEW: The optional `modulo` attribute of a data variable, if present, records a number which can be added or subtracted without altering the validity or physical significance of the quantity. *This is most likely to be useful for longitude coordinate axes, with a* `modulo` *of 360.*

NEW: We note that the Unidata-standard `FORTRAN_format` attribute may be useful for both coordinate and data variables.

13 Topology of an axis

NEW: An axis with "circular topology" is one which can be legitimately transformed by shifting all the points one place along the axis, moving the last point to the beginning, any number of times. The main coordinate variable of an axis with circular topology is distinguished by the presence of an attribute `topology="circular"`. *A longitude axis which circles the whole globe is an example.* The value `linear` or the absence of this attribute indicates an axis with "linear topology". The topology is indicated only by the main coordinate variable, but since it is the property of the axis it applies to any ancillary coordinate variables (section 17) as well.

NEW: When a circular axis is rotated, the main coordinate values must be altered in order to remain monotonic. Therefore the main coordinate variable requires a `modulo` (section 12).

14 Longitude dimension

COARDS: Coordinate variables representing longitudes must always explicitly include the `units` attribute; there is no default value. The `units` attribute will be a string formatted as per the recommendations in the Unidata udunits package. The recommended unit of longitude is `degrees_east` (eastward positive). Also acceptable are `degree_east`, `degree_E`, and `degrees_E`. The unit `degrees_west` (westward positive) is not recommended because it implies a negative conversion factor from `degrees_east`.

NEW: A longitude coordinate variable must have an attribute `quantity="longitude"`.
COARDS: Such a variable is also identifiable from its units string. *The COARDS convention relies on the unit as the only way to identify a longitude variable. This standard uses the* `quantity`*, but requires the* `units` *to be specified as well for compatibility with COARDS.*

NEW: Longitude axes should have the attribute `modulo=360`, indicating that they may be interpreted modulo 360. *Thus, for example, -180, 180, and 540 are all valid representations of the International Dateline and 0 and 360 are both valid representations of the Prime Meridian. COARDS assumes that longitudes may always be treated in this way. Since we have introduced the* `modulo` *attribute, we require that it should be specified to indicate this. But the presence of a* `modulo` *does not mean that the axis necessarily has circular topology (section 9); a longitude axis covering only part of the globe cannot have its points rotated.* COARDS: Note that the sequence of numerical longitude values stored in the netCDF file must be monotonic in a non-modulo sense for a main coordinate variable of longitude.

15 Latitude dimension

COARDS: Coordinate variables representing latitudes must always explicitly include the `units` attribute; there is no default value. The `units` attribute will be a string formatted as per the recommendations in the Unidata udunits package. The recommended unit of latitude is `degrees_north`. Also acceptable are `degree_north`, `degree_N`, and `degrees_N`.

NEW: A latitude coordinate variable must have an attribute `quantity="latitude"`. COARDS: Such a variable is also identifiable from its units string. *The COARDS convention relies on the unit as the only way to identify a latitude variable. This standard uses the* `quantity`*, but requires the* `units` *to be specified as well for compatibility with COARDS.*

16 Vertical (height or depth) dimension

NEW: Whereas the two horizontal dimensions are usually longitude and latitude, whose direction is well defined, a variety of quantities may be used for the vertical axis. The `quantity` attribute is mandatory for the vertical axis. The direction of positive, whether up or down, may be useful for applications displaying or processing the data. For this reason Appendix E to this standard will state the usual direction of positive for all quantities which are commonly used as vertical axes. This information should be taken only as guidance, and may be overridden by the application.

The COARDS standard requires the units of the vertical axis to be selected from a defined list, in order that this axis can be recognised by its units. It gives special status to units of pressure, for which the direction of positive is defined, and introduces a mandatory `positive` *attribute for vertical axes with other units.*

We have adopted a different approach for a number of reasons. Firstly, to require units for the vertical axis means defining dimensionless units for any dimensionless quantity one might wish to use for that axis. This seems inconsistent with the treatment of a data variable which happened to contain a dimensionless physical quantity, for which the standard would not require that units be invented. Secondly, the `quantity` *attribute is more informative than the units, and for most practical axes will clarify the direction of positive. Thirdly, the direction of positive is mostly an issue for displaying the data, and is to some extent a matter of personal preference. If we introduce special treatment for the vertical axis, we should also include information about how any other axis should be displayed. For instance, when latitude is shown on the horizontal axis of a plot, is north on the left or the right? This is the same kind of question, but it strikes us as more a matter for a graphics application to consider than one to be recorded in the data structure. Fourthly, the vertical dimension is recognisable from the order of dimensions (see section 8), which allows any application expecting such a dimension to know where to find it without any further indication. The absence of the* `positive` *attribute means that vertical axes with coordinates other than pressure will not necessarily be recognised as such by COARDS software. If this is a concern, the* `positive` *attribute could optionally be included with one of the allowed values* `up` *or* `down`*, indicating the sense of the direction of positive.*

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: `units="meters"`*,* `quantity="depth"`*. If, on the other hand, the depth of 1000 meters were represented as* -1000 *then the value of the quantity attribute would have been*

`height`. *The COARDS* `positive` *attribute in these two cases would have values* `down` *and* `up` *respectively.*

17 Ancillary coordinate variables

NEW: A single axis might require more than one set of coordinate values, for various purposes described in sections 18, 19 and 20. The additional sets of values required to specify the axis should be contained in extra one-dimensional variables (formally two-dimensional for string-valued variables). We refer to these as "ancillary coordinate variables". Ancillary coordinate variables are recognised as being associated with their axes by having the same dimension, but distinguished from the main coordinate variable for the axis by not sharing the name of the dimension.

18 Component coordinate variables

NEW: A continuous physical coordinate may require more than one number to specify it at each point. We refer to these as "components". This standard requires that a main coordinate variable should nonetheless be supplied which represents a combination of the components that can be used to order the points on the axis. As usual, the main coordinate variable must be monotonic.

NEW: The values of the components are recorded in ancillary coordinate variables referred to as "component coordinate variables". Unlike the main coordinate variable, the components do not need to be monotonic. The names of the component coordinate variables are recorded as a comma-separated list in a `component` string attribute of the main coordinate variable.

An example is the hybrid vertical coordinate $\eta \equiv p/p_0 + \sigma$, *used in the Hadley Centre GCM. Atmospheric model levels are specified in terms of* (p, σ) *pairs, where* p *is pressure,* p_0 *is a constant and* σ *is fraction of surface pressure (which is variable). The* η *value is a linear combination of the two, which cannot be uniquely decomposed back into* (p, σ) *as the coordinate variable* `eta(eta)`*, say, with* p *and* σ *in ancillary coordinate variables* `pressure(eta)` *and* `sigma(eta)`*. The* `eta` *variable would have a string attribute* `component="pressure,sigma"`*. Information about how to relate the components to the combination would reside in the application.*

19 Associated coordinate variables

NEW: A qualitatively different situation arises where an axis has a number of alternative ways of being labelled, providing different kinds of information. We refer to the alternative sets as "associated coordinates" and store them in ancillary coordinate variables. The main coordinate variable records the names of these variables as a comma-separated list in an `associate` string attribute. The main coordinate variable must be monotonic, as usual, but associated coordinate variables need not be.

One example is a vertical axis where one wishes to store both the physical coordinate and the ordinal model level number. The latter could be recorded as an associated coordinate variable. In this instance, if the physical coordinate was the η *of the section 18, it would also have component coordinate variables. Another example is the value of a quantity along a one-dimensional trajectory. In such a case, we might have a coordinate*

variable containing distance along the trajectory and associated coordinate variables giving the latitude and longitude of each point.

20 Bundles

NEW: If several data arrays containing the same physical quantity have one or more identical axes, but are distinguished by the values of other singleton coordinate variables, it may be convenient to store them in the same data variable. The common axes of the separate arrays become axes of the combined variable. One or more additional axes are introduced to "bundle up" the separate arrays. Such an axis does not correspond to a continuous physical coordinate. It acts simply as an index of the bundled-up arrays. *For instance, the Hadley Centre GCM can generate timeseries of the values of quantities at individual points. Typically, timeseries from many different points are produced of the same quantity at the same sampling times. It is natural to contain this information in a data variable with two dimensions. One dimension is the common time axis, specifying the sampling times, which are the same for all the points sampled. The other dimension is not a continuous physical coordinate; it is simply being used to "bundle up" the timeseries, the points being irregularly scattered in a space of two or more dimensions. This axis might have a meaningful coordinate variable, such as a station number (in which case it must be monotonic), but otherwise it may be no more than an index.*

NEW: The singleton values of the separate arrays are recorded in associated coordinate variables for the bundling dimension. They should not be interpreted as continuous coordinates. *In the timeseries example, the latitude and longitude of the points would be recorded in associated coordinate variables. It might be desirable to have a string-valued coordinate variable as well, to name the points. For a set of temperature timeseries, suppose that the dimension* `points` *specified the number of points, and* `times` *the number of sampling times. The data variable might be* `temperature(times,points)` *and the time axis* `times(times)`*. The main coordinate variable for the* `points` *axis would be* `points(points)`*. The associated coordinate variables would be* `latitude(points)`*,* `longitude(points)` *and* `names(points,StringMaxLength)`*. This same form could be used for observed timeseries from stations.*

Another application could be that of vertical profiles at sets of points; for example, a vertical temperature profile through the ocean, or data from a radiosonde. One dimension is the height or depth, with the physical vertical coordinate. The other has a main coordinate variable containing just an index, with associated coordinate variables recording the geographical location of the points.

This section raises the question of how best to store a single timeseries, or a single vertical profile. Following the scheme of this section, it could be contained in a two-dimensional data variable with the bundling axis being of size unity. The associated information such as latitude or longitude would then be stored in singleton coordinate variables, all associated with the same dimension. Alternatively, these values could be recorded as separate singleton dimensions (following section 8). We have no recommendation for this. Either scheme could be appropriate; which is more natural perhaps depends on how the data was extracted from the continuous axes.

21 Boundary coordinate variables

NEW: Along a dimension, the values might relate to points (at the coordinate values) or to contiguous or non-contiguous cells. The boundaries of the cells should be defined as well as the cell coordinate values. The convention is to define an additional two-dimensional "boundary coordinate variable" with a left-hand dimension *(trailing dimension in Fortran terms)* of size two. The values for which this dimension has index 0 (numbering from 0 i.e. in C notation) supply the boundaries with the smaller coordinate values, and those with index 1 the large values, where "smaller" and "larger" refer simply to numerical comparison, not to a physical direction. The name of the boundary coordinate variable is recorded in a string attribute `bounds` of the main coordinate variable. We recommend that the it should be named by the coordinate dimension with the prefix `bounds_`. *Thus, for instance, a coordinate variable* `lat(lat)` *might have attribute* `bounds="bounds_lat"` *pointing to a boundary coordinate variable* `bounds_lat(2,lat)` *(declared as in CDL). In C notation,* `lat[0]` *gives the coordinate of the first point,* `bounds_lat[0][0]` *its lower boundary,* `bounds_lat[1][0]` *its upper boundary. In Fortran notation, the declarations are* `lat(lat)` *and* `bounds_lat(lat,2)`*, and the relevant elements are* `lat(1)`*,* `bounds_lat(1,1)`*,* `bounds_lat(1,2)`*.* Supplying upper and lower boundaries separately allows for the possibility that the cells might not be contiguous; they might even overlap.

NEW: Boundary coordinate variables are recommended if the main coordinate values are not evenly spaced, or if the dimension has a size of unity. If the coordinates are evenly spaced, and boundaries are not specified, generic applications may assume that the main coordinates lie at the centres of their cells. Boundary coordinate variables may be supplied for ancillary coordinate variables as well as main coordinate variables. The upper and lower boundaries of ancillary coordinates are ordered so as to correspond with those of the main coordinate.

22 Point values versus average values

NEW: A `subcell` attribute may be given to the main coordinate variable for a dimension to specify whether the data values represent a point value (`subcell="point"`) or an average over the cell (`subcell="cell"`) in that dimension. It is conceivable that the interpretation as point or average values might differ for the various dimensions. *An important example of the need to distinguish arises for timeseries, which might be instantaneous values (such as is usual with pressure measured at synoptic stations) or values representing a time-mean (such as the rainfall rate derived from the total accumulated over the observation period).*

23 Contracted dimensions

NEW: A contracted dimension is one in which all data points share the entire contracted axis. It indicates the relationship of the data variable which is being described to another variable of higher dimensionality. A contracted dimension will have a size of unity, a single representative coordinate value and boundary coordinate values supplying the range of the uncontracted axis. If the uncontracted axis had the attribute `subcell="point"`, the boundaries of the contracted axis will be the extreme point coordinate values of the uncontracted axis; if the uncontracted axis had `subcell="cell"`, the boundaries will be

the extreme boundary coordinate values.

NEW: The `contraction` attribute of the coordinate variable specifies the operation which contracted the axis; permitted values are detailed in Appendix B and include `sum`, `mean`, `max`, `min`, `sd`, `var`. Contractions are limited to common and well-defined statistical operations performed using the values of the data variable and possibly the coordinate variable, which may be needed to determine statistical weights. If weights have been used, the quantity used for weighting may be recorded in a `weight` attribute, and should be one of the quantities listed in Appendix E (section 12). Note that some operations, such as taking a weighted sum or a variance, imply that the quantity in the data variable does not have the same physical unit as the quantity which had the uncontracted axis. In this case, the `units` and `quantity` should be altered; the `contraction` does not suffice. This standard also states that two variables related by a non-linear contraction, such as the standard deviation, necessarily have different `quantity` attributes, even if they have the same `units`. This will be reflected in Appendix E. *For instance, "orographic height" and "standard deviation of orographic height" are regarded as different quantities, while "maximum orographic height" is treated as the same quantity as "orographic height".*

NEW: The optional `min_interval` and `max_interval` attributes specify the minimum and maximum distance between two adjacent coordinates of the uncontracted axis. Equality implies that the uncontracted coordinates were evenly spaced.

The most obvious example of a contracted dimension is the time information belonging to an array where time is not otherwise a dimension. The beginning and ending times of the period over which a time-mean (for example) has been calculated form the lower and upper time boundaries of a contracted dimension of time. (See below, section 24, for more on time axes.) Zonal and meridional means are further common examples of contracted axes, whose boundary coordinate variables would specify the ranges of longitude and latitude over which the mean had been formed.

NEW: In some cases, several axes are contracted simultaneously. If the operation cannot be peformed on them separately, probably because it involves a non-linear operation, the contraction has to be recorded as affecting more than one axis at a time. This is done by replacing all the affected dimensions with one new singleton dimension carrying the `contraction` attribute. The information from the individual contracted axes is recorded on singleton coordinate variables associated with the contracted axis (see section 19). *For instance, the standard deviation involves a non-linear operation, so an area-weighted standard deviation must be performed on both horizontal axes at once. On calculating the area-weighted standard deviation of an array of* `temperature(lat,lon)`*, we might record the result in* `sd_temperature(area)`*, with* `area` *as a dimension of size unity, having a dummy coordinate variable* `area(area)` *with attributes* `contraction="sd"` *and* `weight="area"` *and associated coordinate variables* `lat(area)` *and* `lon(area)` *with boundaries, units, etc.*

24 Time axes

NEW: A "time axis" is one which represents date and time, which we will refer to hereafter just as "time". Time is usually defined by a set of up to six numbers (year, month, day, hour, minute, second) of various data types. It would be possible to define it thus in a netCDF file, using six ancillary coordinate variables. However, it is more efficient and for many purposes more convenient to encode a time into a single number, giving the elapsed interval since a certain reference. The units of the interval and the reference time are

matters of convention.

NEW: A "calendar" defines the set of valid dates (year-month-day combinations). The standard calendar is the Gregorian (the calendar of udunits), but climate models do not always use this. *For instance, in the calendar of the Hadley Centre GCM, all months have 30 days.* The elapsed interval between two dates will not necessarily be the same in two different calendars, because there may be different numbers of valid dates between them. *For example, the interval between 1 February 1996 and 1 March 1996 is 29 days in the Gregorian calendar, but 30 days in the Hadley Centre model calendar, since 30 February is a valid date in the latter.* Therefore the encoding of a time into an elapsed interval will depend on the calendar, and it is necessary to know the calendar when converting.

NEW: This standard permits the use of the standard calendar (below, section 25) and of other calendars (section 26). In addition, we propose and recommend a new calendar (section 27), which has the advantage of being able to encode a date which is legal in any of the other calendars, and which can therefore be used in all cases.

COARDS: Time coordinate variables must always explicitly include the `units` attribute; there is no default value. NEW: The `quantity` is also mandatory, having one of the values `time` or `unitime` (see section 27 concerning the latter). A time coordinate variable will be identifiable by its `units` and `quantity`. The `quantity` in conjunction with the `calendar` attribute, when required, will indicate the calendar in use and how it has been encoded. The `calendar` attribute is described in the following sections. If a time coordinate variable has no `calendar` attribute, the global `calendar` attribute (section 5), if present, applies to it. The attribute `time_format` can be included to specify a format for printing the date and time, according to the conventions of the Unix (TM) `date` command.

25 Gregorian calendar

COARDS: Intervals between two times in the standard Gregorian calendar can be calculated by the udunits package. *This calendar is actually the mixed Gregorian/Julian calendar system, as followed in England. Dates prior to 1582-10-15 are assumed to use the Julian calendar.* Time coordinates in this calendar may have units of time formatted as per the recommendations in the Unidata udunits package, which specify a unit and a reference time *e.g. a unit of* `seconds since 1992-10-8 15:15:42.5` *indicates seconds since 8 October 1992 at 3 hours, 15 minutes and 42.5 seconds in the afternoon, in Universal Coordinated Time (time zones can also be handled).*

NEW: The file `udunits.dat` defines second, minute, hour and day as units of time. Units of months and years must not be used, because both are of variable length. Since udunits defines a year as exactly 365 days and a month as exactly a twelfth of a year, use of these units will not always give the expected results. *For example,* `1 month since 1997-4-1 0:0:0` *is treated by udunits as* `30.4368 days since 1997-4-1 0:0:0`*, not* `1997-5-1 0:0:0`*.* Note, however, that this standard deals only with the encoding of time in the netCDF file. An application which processes such a file may have an interface which handles intervals defined in months or years, if desired.

NEW: This standard recommends that Gregorian times be given in units of `days since 1-1-1` i.e. midnight on 1 January 1 AD, in data type `double`. *(Note that udunits treats 0 AD as identical to 1 AD.)* This data type gives a precision of about 16 decimal digits, which means that it can resolve tenths of a second for years of up to around 10,000 AD. The larger the year, the worse the absolute precision. If this precision is not sufficient,

use of a different reference time, in order to make the year smaller, is recommended.

NEW: If there is no `calendar` attribute applying to a time coordinate variable in any standard unit of time, the coordinates are assumed to be in the normal Gregorian calendar, encoded following the udunits convention. This can be made explicit by setting `calendar` to `standard` or `gregorian`.

26 Non-Gregorian calendars

NEW: It is recommended that times in other calendars should be encoded in data type `double` with a `units` string of `days since 1-1-1`, interpreted as days since midnight on 1 January of year 1 in the calendar concerned. If a different base time is chosen, the udunits syntax "`days since` date-and-time" should be used. The udunits package itself can process only the standard calendar. An extension will be required for other calendars. Shorter units than days may be used, but units of months and years are not allowed, as above.

NEW: Apart from the Gregorian, calendars recognised by this standard are `julian` for the Julian calendar (in which all years divisible by four are leap years), `noleap` for a calendar with 365 days in every year, and `360` when each month has 30 days in every year. If any other calendar is used, a suitable description should appear in the `calendar` attribute, but generic applications cannot be expected to be able to encode and decode times in the calendar concerned.

27 Unimonth calendar

NEW: This standard proposes a new means of representing times, namely the "unimonth" calendar, which assumes all months have 100 days. A time coordinate variable in the unimonth calendar is indicated by a `quantity` attribute of `unitime`. This standard recommends that times should be given in `days since 1-1-1` in data type `double` in the unimonth calendar. Dates can be converted into this unitime form and back unambiguously, according to

$$\begin{aligned} \texttt{days} &= ((\texttt{year}-1)\times 12+(\texttt{month}-1))\times 100+\texttt{day\_in\_month}-1 \\ &+ (\texttt{hour}+(\texttt{minute}+\texttt{second}/60)/60)/24. \end{aligned}$$

This formula defines the time `0 days` as midnight on 1 January in year 1. *We are not, of course, suggesting that the unimonth calendar should be adopted as the calendar of a climate model, or in place of the Gregorian calendar. It is introduced only to provide a convenient means of encoding times.*

NEW: The main reason for proposing this convention is that all dates (year, month, day) in any of the calendars recognised by this standard (sections 25 and 26) are valid in the unimonth calendar. Hence any such dates can be converted into and out of unitime. Moreover, no knowledge of the specific calendar is required to do this. This knowledge is still required for calculating some intervals of time (see below, section 28), so when the unimonth calendar is used to encode a time axis into unitime, the `calendar` attribute should be used to record the original calendar.

28 Intervals of time

NEW: An interval of time is the difference between two times. The `quantity` is as for absolute time; an interval is distinguished by the lack of a reference time in its `units` string *e.g. plain* `seconds`*, compared with* `seconds since 1992-10-8 15:15:42.5`*. Both data variables and coordinate variables might contain intervals of time, mostly probably expressing elapsed time since some event e.g. days since the onset of the monsoon.* A further specialised application is in the representation of climatological time using multiple time axes, described below (section 29).

NEW: To be unambiguous, a time interval must be expressed in units which have a well-defined value. We identify two different kinds of time interval. For intervals defined as numbers of days or shorter units of time, there is no problem, as these units are well defined by udunits. We recommend the use of days with data type `double`, but allow the use of shorter units of time instead. However, months and years are not well defined (as discussed above, section 24), so must not be used as units of intervals of time. We may nonetheless have need to express time intervals in terms of years and months (and possibly a remainder of days or less), so we need a means of encoding these unambiguously. *A climatological season, for example, is three months.*

NEW: Any interval of time expressed as a number of years, months, days, hours, minutes and seconds can be unambiguously converted to an interval in unitime (see above, section 24) and back without knowledge of the calendar. This standard recommends that intervals of the second kind above (involving months or years) should be enocoded in data type `double` in days of unitime, according to the formula

$$\begin{aligned} \texttt{days} &= (\texttt{years} \times 12 + \texttt{months}) \times 100 + \texttt{days\_in\_month} \\ &+ (\texttt{hours} + (\texttt{minutes} + \texttt{seconds}/60)/60)/24. \end{aligned}$$

Shorter units than days can also be used. Intervals of the first kind (not involving months or years) less than 100 days in length can also be represented in unitime.

An example may help to clarify these two kinds of interval further. The interval between 1 February 1996 and 1 March 1996 is 29 days in the Gregorian calendar, but 30 days in the Hadley Centre model calendar. If we wish to record this interval as a number of days, we can do so as `29 days` *or* `30 days` *respectively, of either ordinary time or unitime. If, however, our intention is to record that it is one month, we encode it as* `100 days` *of unitime. The interval between 1 February and 1 March is* `100 days` *of unitime (i.e. one month) in all years in all calendars. The importance of the distinction is that it tells us how to add an interval to an absolute time.*

29 Multiple time axes and climatological time

NEW: There is no bar on a data variable having more than one dimension in a particular quantity, so long as the dimensions have different names. A particular need for this arises for multiple time dimensions, when there has been a contraction over at least one of them. This provides a means of describing "climatological time", in which the data does not apply to a specific time or times.

NEW: This need must be described firstly with reference to the original uncontracted dimension spanning a period of time which contains repetitions of a cycle or cycles *(most often the seasonal or diurnal cycles).* This dimension can be decomposed into an absolute

time, labelling the beginning of the cycle or cycles, and offsets giving the phases within the cycles. The absolute time is coded just as for a normal time axis (section 24). We recommend that the offsets be specified in days, as described above for intervals of time (section 28), with a `quantity` of `time` or `unitime`, as appropriate. Offsets in shorter units than days are also acceptable. The first offset is relative to the absolute time, the second offset is relative to the first offset, etc. (In practice, there is not likely to be more than one offset.) These time dimensions should be successive dimensions of the data variable, in order to indicate their grouping, and should appear in order, with the absolute time dimension first in the sense of a CDL declaration (*last in Fortran terms*). Each offset time variable should have a string attribute `wrt` ("with respect to"), naming the next time dimension in the group. The first of them might have `subcell="point"` or `subcell="cell"`, and might have boundary coordinate values, but all the others are taken as point values. The actual time of a data value is obtained by adding up the values of all its time coordinates. Unless the offsets are all exact numbers of months and years, this addition may require knowledge of the calendar, which is an attribute of the absolute time axis. The `calendar` attribute is not required for the offsets.

To take an example, consider a time dimension which has coordinates at daily intervals from 1 June to 30 June in each of the five years 1990–1994. This dimension has a size of $30 \times 5 = 150$. *The first coordinate is 1 June 1990 and the last is 30 June 1994. This single time dimension can be decomposed into two: an absolute dimension of size 5 and values at yearly intervals from 1 June 1990 to 1 June 1994, and an offset of size 30 and values of 0, 1, . . . , 29 days. The data value for 13 June 1992 appears at indexes 2 and 12 (numbering from 0) in these two dimensions, respectively, and its time coordinate is obtained by adding 12 days to 1 June 1992. A decomposition into three might be appropriate if the original time coordinate had values at, say, 3-hourly intervals. The third dimension would then have a size of 8 and values of 0, 3, 6, . . . , 21 hours (expressed in days). Such a representation would be appropriate for quantities measured instantaneously at 3-hourly intervals. If, on the other hand, they were means over successive intervals of 3 hours, the third dimension would have* `subcell="cell"`*, lower time boundaries of 0, 3, 6, . . . , 21 hours and upper time boundaries of 3, 6, 9, . . . , 24 hours.*

NEW: Once a time dimension has been decomposed in this way, any of the resulting time dimensions can be contracted, as for any other dimension. That is the main reason for performing the decomposition, and it is most useful when it is the leading dimensions which are contracted, since a contracted trailing time dimension can be merged into a preceding uncontracted time dimension with no loss of information. After contraction, boundary coordinate variables should record the first and last times of the contracted dimension, and `min_interval` and `max_interval` the extreme spacings between the coordinates. The combination of a contracted absolute time dimension and one or more offset-time dimensions indicates climatological time. *COARDS recommends use of year 0 to indicate climatological time. We do not favour this convention. Firstly, it does not provide any way of recording which years were used to make the climatology. Secondly, udunits treats year 0 and year 1 as identical (which is reasonable because year 0 does not exist—there is no year between 1 AD and 1 BC).*

In the example above, we might contract the first time axis. It would then have a size of one, boundaries of 1 June 1990 and 1 June 1994, and minimum and maximum intervals of one year. The data variable would contain 5-year means for the individual days of June. If we contracted the second dimension as well, it would have a size of one also, boundaries of 0 and 29 days (or 0 and 1 month, for time-means over the days rather

than instantaneous values), minimum and maximum intervals of 1 day. This would be a 5-year June mean. The first contracted dimension tells us it is made from five years, and the second that it uses one month from each year. If we had retained the third dimension uncontracted, we would have a set of values for times at 3-hourly intervals within an average June day.

30 Special surfaces

NEW: A data variable may be defined as existing on a surface, the general description of a surface being a set of points which have two or more independent dimensions and other dimensions which are single-valued functions of the independent dimensions. Existence on a surface is a property of the data variable, and should be indicated by giving it a `surface` attribute identifying the surface, one from a list which constitutes Appendix D of this standard. To provide a longer description of the surface than the `surface` attribute itself, a `surface_name` attribute may be included. *For example, a data variable of pressure at mean sea level would have an attribute* `surface="sea level"`. *We intend to expand Appendix D in response to requests, as for Appendix E. Note that surfaces which can be defined by the numeric value of physical quantities will not be included. For example, the surface 1.5 m above the ground is not a special surface in the sense of this section, as a variable can be described as being on this surface by giving it a singleton height coordinate with a value of 1.5 m.*

NEW: If a plain description is insufficient to characterise the surface, then the values of the dependent coordinates which define it must be supplied as additional data variables. These variables should be named as a comma-separated list in an attribute `surface_coords`. *If it were necessary to know the depth of mean sea-level below ground, this could be done by specifying* `surface_coords="mslheight"` *and including a data variable* `mslheight(lat,lon)` *(CDL declaration),* `lat` *and* `lon` *also being dimensions of the pressure variable.*

31 Invalid values in a data variable

NEW: Invalid values are not permitted in a coordinate variable, so this section applies only to data variables. Invalid values are any which fall outside the valid range or equal the fill value, as indicated by the Unidata-standard attributes described here. An invalid value indicates bad data i.e. a software problem, which is a different circumstance from unknown or missing data (see section 32).

UNIDATA: The attribute `valid_min` is a scalar specifying the minimum valid value for a variable. The attribute `valid_max` specifies the maximum valid value, while `valid_range` is a vector of two numbers specifying the minimum and maximum valid values, in that order, equivalent to specifying values for both `valid_min` and `valid_max` attributes. Any of these attributes define the valid range. The attribute `valid_range` must not be defined if either `valid_min` or `valid_max` is defined. Generic applications should treat values outside the valid range as invalid. The type of each `valid_range`, `valid_min` and `valid_max` attribute should match the type of its variable. *The Unidata special treatment of* `byte` *type is not included here as we do not recommend use of that type (see section 3).*

COARDS: A scalar attribute with the name `_FillValue` and of the same type as its variable is used as the fill value for the variable. The netCDF package defines a default

fill value for each type of variable, so it is not necessary to define your own `_FillValue` attribute if the default is suitable. The purpose of the fill value is to save the applications programmer the work of prefilling the data and also to eliminate the duplicate writes that result from netCDF filling in undefined data with its default fill value, only to be immediately overwritten by the programmer's preferred value. This value is considered to be a special value that indicates undefined 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. NEW: In cases where the data variable is packed using the `scale_factor` and `add_offset` attributes (section 34), the `_FillValue` attribute applies the numbers as packed, so they must be checked against it before unpacking.

NEW: If none of `valid_min`, `valid_max` or `valid_range` is defined then generic applications should define a valid range by using the fill value (whether defined explicitly or by default); if the fill value is positive then it defines a valid maximum, otherwise it defines a valid minimum. For integer types, there should be a difference of 1 between the fill value and this valid minimum or maximum. For floating point types, the valid extreme should have a magnitude which is half the magnitude of the fill value. *We recommend a factor of two, rather than a difference of one bit, because it is easier for applications programmers. There is no special treatment for* `byte` *as we do not recommend that type (see section 3).*

32 Missing values in a data variable

NEW: Missing values are not permitted in a coordinate variable, so this section applies only to data variables. The `missing_value` attribute indicates a value that is used for data that are unknown or "missing". This attribute is not be treated in any special way by the netCDF API, unlike the `_FillValue` attribute (section 31). The `missing_value` should be outside the valid range (section 31), so that generic applications will treat it appropriately. The netCDF data type of the `missing_value` attribute should match the netCDF data type of the data variable that it describes. In cases where the data variable is packed via the `scale_factor` and `add_offset` attributes (section 34), the `missing_value` attribute matches the type of and should be compared with the data after unpacking. *This standard is unlike COARDS in giving a particular interpretation to the distinction between* `missing_value` *and* `_FillValue`*.*

33 Compression by gathering

NEW: To save space in the netCDF file, it may be desirable to eliminate points from data arrays which 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 axes to be compressed might not be all the axes. For example, we might wish to compress a three-dimensional longitude-latitude-depth array of soil temperatures by eliminating sea points at all depths. In this case, only the longitude and latitude axes would be affected by the compression.* The list is constructed by considering a mask array which has just the axes to be compressed, and 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.

NEW: 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 `compressed`, containing a comma-separated list of the dimensions which were affected by the compression in the order of the CDL declaration of the uncompressed array. The list, the compressed arrays and the original dimensions are written to the archived netCDF file. The uncompressed arrays can be reconstituted using this information. *In the example above, suppose the original data variable is* `soilt(level,lat,lon)`. *We construct a list* `landpoint(landpoint)` *containing the indices of land points in a mask array* `landmask(lat,lon)`. *We compress the data variable to a new variable* `landsoilt(level,landpoint)` *and record the original dimensions in an attribute* `compressed="lat,lon"` *of the* `landpoint` *list variable.*

34 Compression using a scale and offset

COARDS: This standard endorses the use of the optional Unidata-standard attributes `scale_factor` and `add_offset` for data and coordinate variables. These attributes can be used to provide simple number compression (packing), to store low-resolution floating-point data as small integers in a netCDF file. After the data values of the variable have been read in, they are to be multiplied by the `scale_offset`, and have `add_offset` added to them. If both `scale_factor` and `add_offset` 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. NEW: This procedure is concerned only with storage. It does not affect the unit of the quantity. *For instance, a pressure variable with values in the range 900.0–1100.0 Pa could be converted to short integers in the range* ± 20000 *by subtracting 1000 and dividing by 0.005 i.e. multiplying by 200. The units of the compressed variable are still recorded as pascals.*

COARDS: This standard is more restrictive than the netCDF Users' Guide 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 no restrictions apply; 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 associated variable (containing the packed data) then in files adhering to this standard the associated variable may only be of type `short` or `long`. *We exclude* `byte` *on grounds discussed in section 3.* The attributes `scale_factor` and `add_offset` (which must match in data type) must be of type `float` or `double`. The data type of the attributes should match the intended type of the unpacked data. (It is not advised to unpack a `long` into a `float` as there is a potential precision loss.) *Users should note that Unidata may provide a built-in means of packing data in netCDF files in future.*

A Attributes

Attribute	Section(s)	Description
add_offset	31 34	Additive offset for packing data
appendices	4	Version number of these appendices
associate	19	Identifies variables containing alternative sets of coordinates
bounds	21	Identifies a variable containing boundary coordinate values
calendar	4 24 25 26 27 29	Calendar used for encoding time axes
comment	4	Additional information about the file
component	18	Identifies variables containing components of a coordinate variable
compressed	33	Records the dimensions which have been compressed by gathering
contraction	23	Records how an axis was contracted to a point
Conventions	4	Identifies the netCDF standard
_FillValue	31	Indicator of invalid data
FORTRAN_format	12	Format for printing the values of a variable
history	4	Evolution of the data in the file
institution	4	Who made or supplied the data
long_name	12	Long description of a physical quantity
max_interval	23 29	The maximum separation between points on an axis before contraction
min_interval	23 29	The minimum separation between points on an axis before contraction
modulo	12	Arithmetic modulo of a coordinate variable
north_pole	10	Geographical location of rotated North Pole
positive	16	Direction of positive for a vertical axis
production	4	How the data was produced
quantity	12 14 15 16 24 27	Short description of a physical quantity
scale_factor	31 34	Multiplicative factor for packing data
subcell	22 29	Indicates whether data values are points samples or cell averages
surface	30	Identifies a surface on which a quantity is defined
surface_coords	30	Identifies dependent variables defining a surface
surface_name	30	Description of a surface
topology	13	Topology of an axis (circular or not)
time_format	24	Format for printing a time and date
units	12 14 15 24 28	Units of a physical quantity
valid_max	31	Largest valid value of a variable
valid_min	31	Smallest valid value of a variable
valid_range	31	Smallest and largest valid values of a variable
weight	23	Weighting for points used when performing a statistical contraction of an axis
wrt	29	Indicates the time axis to which a time offset applies

B Contractions

Contraction	Units	Quantity	Description
`max`	u	N	Maximum
`min`	u	N	Minimum
`mean`	u	N	Mean
`sd`	u	Y	Standard deviation
`sum` without `weight`	u	N	Sum
`sum` with `weight`	$u \times w$	Y	Weighted sum
`var`	u^2	Y	Variance

Units: u means the units of the original quantity, and w the units of the weighting quantity
Quantity: "N" means the quantity does not change, "Y" that it does change

C Modifications to `udunits.dat`

None at present. COARDS: The unit `degrees` is not permitted, because it creates ambiguities when attempting to differentiate longitude and latitude coordinate variables. This unit does not appear in the current version of the file.

D Surfaces

This Appendix is not yet available. It will contain entries such as:

Surface	Description
`surface`	Surface of the land or sea
`sea level`	Mean sea-level
`TOA`	Top of the atmosphere

Surface: Case, spaces and punctuation are not significant in the name of the surface.
Description: This is an explanation of the surface for the purpose of this Appendix only. Applications should not rely on this Appendix as a lookup table for explanations, however. If the surface needs a description, it should be attached to the variable in a `surface_name` attribute.

E Quantities

This Appendix is not yet available. As well as existing as part of this standard, it will be made available as a flat ASCII file for use by applications programs. It will contain entries such as:

Version	Quantity	Unit	Positive	Description
1.0	`height`	m	U	Height above the surface of the land or sea
1.0	`depth`	m	D	Depth below the surface of the land or sea
1.0	`pressure`	Pa	D	Pressure
1.0	`longitude`	degree_east	–	Longitude
1.0	`latitude`	degree_north	–	Latitude
1.0	`time`	d	–	Time (in any calendar except unimonth)
1.0	`unitime`	d	–	Time (in the unimonth calendar)
1.0	`temperature`	K	–	Temperature
1.0	`soil temperature`	K	–	Temperature of the soil
1.0	`precipitation rate`	$\mathrm{kg\,m^{-2}\,s^{-1}}$	–	Rate of precipitation in all phases together

Version: The version of the appendices at which this quantity was introduced.
Quantity: Case, spaces and punctuation are not significant in the name of the quantity.
Positive: For quantities commonly used as vertical axes, "U" means that values increase upwards along the axis, "D" means downwards. No value is indicated for quantities not used in this way.
Description: This is an explanation of the quantity for the purpose of this Appendix only. Applications should not rely on this Appendix as a lookup table for explanations, however. If the quantity needs a description, it should be attached to the variable in a `long_name` attribute.