GDT netCDF conventions for climate data

Version 1.1

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

12th June 1998

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 `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. Although this is specifically a netCDF standard, we feel that most of the ideas are of wider application. Our main purpose is to propose a clear, adequate and flexible definition of the metadata needed for climate data. The metadata objects could be contained in file formats other than netCDF. Interconversion of the metadata between files of different formats will be facilitated if they are based on similar ideas.

This standard is mostly additional to the conventions sponsored by COARDS (`ftp://-ftp.unidata.ucar.edu/pub/netcdf/Conventions/COARDS`). In the following standard, parts which are identical to or paraphrases of the COARDS standard are prefixed with COARDS. Material introduced by us in our original proposal of 10th June 1997 is prefixed with OGDT and material newly introduced in this version 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 given in *emphasised type* and CDL examples given in ***`slanted typewriter type`*** are not part of the standard. Note that examples typically show only the details relevant to the point under discussion, and hence may be incomplete with respect to the provisions of the complete standard.

NEW: Successful transmission of data depends upon the receiver of the file having software that will correctly it. For this reason, a strategy that is as conservative as possible in the use of attributes and encoding techniques will best promote portability of data.

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`. OGDT: See section 11 for details of how the package is used by this convention to define units for physical quantities.

Useful comments from John Sheldon, Jan Polcher, Harvey Davies, John Caron, Steve Hankin and other contributors to the netCDF news group have influenced the development of this standard. We have made some changes to gain greater compatibility with the NCAR CSM netCDF standard.

2 Filename

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

3 Data types

OGDT: 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 is ambiguous in netCDF. *The COARDS convention deprecates* `char`*, rather than* `byte`*.*

OGDT: 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. OGDT: 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. NEW: 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 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.

NEW: The Unidata-standard attribute `Conventions` is recommended to reference this standard, containing the string `"GDT 1.1"`. This standard is registered with Unidata under the name "GDT" in the directory `ftp://ftp.unidata.ucar.edu/pub/netcdf/Conventions` and is available from `http://www-pcmdi.llnl.gov/drach/GDT_convention.html` and `http://www.met-office.gov.uk/sec5/CR_div/GDT_convention.html`.

OGDT: 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 to name the model integration.*

OGDT: 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). *This information could perhaps be recorded in the* `Conventions` *attribute, but having a separate attribute for it will allow applications to extract the information without having to parse a string.*

OGDT: 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. OGDT: 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. NEW: It is also recommended that variable names should be obviously meaningful, if possible, as this renders the file more effectively self-describing. However, nothing in this convention relies on the use of particular names for variables.

7 Data variables

OGDT: 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 Coordinate variables

OGDT: A one-dimensional netCDF variable associated with a dimension 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, 18, 19, 20 and 21). 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) *because this assumption is frequently made by software.*

9 Axes and dimensionality of a data variable

OGDT: 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. *Several examples can be found in this document. 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 30 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. *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.*

The reason for this convention is that these kinds of axes may have special meanings to particular applications. For instance, an application might want to plot a longitude–latitude map, or integrate vertically, or extract a timeseries. In the COARDS standard, the indication given by the order of dimensions and information in the attributes of the coordinate variables have to be used together to identify the required axes. For compatibility with COARDS, we uphold all these conventions, but we also introduce a new attribute to make the identification straightforward and unambiguous, as follows.

NEW: If the last four dimensions do not have the interpretations `TZYX` (in CDL order, omitting from the left if there are fewer than four dimensions), an `axis` attribute should be attached to the data variable. In other cases it is optional, but recommended. This attribute is a `char` array of size equal to the dimensionality of the data variable, having one element for each dimension (in CDL order), indicating the interpretation of that

dimension. The permitted characters are `T Z Y` and `X`, with the meanings given above, and `-` as a placeholder for a dimension which has none of these meanings. Each permitted letter may appear no more than once in the array. If a data variable has more than one dimension which could be given a certain interpretation, the `axis` attribute will therefore clarify which should be chosen. Note that if there are multiple time axes (section 30), of which only one is not collapsed, this single "climatological time" axis would normally be the designated `T`-axis. *If the* `axis` *attribute is included, the dimensions may be put in any order, but this should be avoided if possible because applications not able to use this attribute may not process the data correctly.*

Axes for an ordinary time-mean longitude–latitude–height variable:

```
dimensions:
  lat=18;
  lon=36;
  pressure=15;
  con_time=1;
variables:
  float xwind(con_time,pressure,lat,lon); // order T Z Y X
    xwind:axis="TZYX";
  float lon(lon);
  float lat(lat);
  float pressure(pressure);
  float con_time(con_time);
```

See section 11 for the description of the units and quantities of the variables and sections 14, 15, 16, 25 for details of longitude, latitude, vertical and time axes.

OGDT: The coordinates of points within the data variable are the simple ordered tuples formed by associating values from the coordinate variables (section 8). NEW: If a particular axis does not have a coordinate variable, the coordinate values are assumed to be equal to their indices along the axis, numbering from 0.

OGDT: 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. *The advantage of this method is that all the attributes of a coordinate variable (quantity, components, boundaries, etc.) can be used to describe the single-valued quantity.* Singleton dimensions also result from contractions, described in section 23.

Longitude–latitude field of temperature on a pressure level: *This would use a singleton pressure dimension to record the level, thus:*

```
dimensions:
  lon=96;
  lat=72;
  pressure=1; // single-valued coordinate variable
variables:
  float temperature(pressure,lat,lon); // axes in order Z Y X
```

```
    temperature:axis="ZYX";
  float pressure(pressure);
    pressure:long_name="pressure";
    pressure:units="kPa";
data:
  pressure=50.0; // Pressure level of 50 kPa = 500 mbar
```

The `units` *and* `long_name` *attributes are described in section 12.*

Surface air temperature: *Surface meteorological measurements are made at a certain defined height e.g. 1.5 m, which can be shown thus:*

```
variables:
  float temperature(height,lat,lon);
    temperature:axis="ZYX";
    temperature:long_name="atmospheric temperature";
    temperature:units="K";
  float height(height);
    height:long_name="height above the surface";
    height:units="m";
data:
  height=1.5;
```

The explicit height should not be given if the surface of measurement is included in the name of the quantity e.g. screen height.

10 Coordinate systems

OGDT: 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.

OGDT: 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 associated coordinate variables to specify the coordinates, point by point (see section 19).*

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.* OGDT: 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.

OGDT: 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.* This facility 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 two attributes for specifying the physical quantity of data and coordinate variables. Both of them are strings and defined by Unidata conventions.

OGDT: The `units` attribute is formatted as per the recommendations in the Unidata udunits package (see section 11), with extensions for time (see section 27). Case is significant in the `units`. This attribute is mandatory unless the quantity is dimensionless (a pure number). *There are a few defined dimensionless units, such as* `percent`, *but 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* `long_name` *rather than the* `units`. *A scale factor and/or offset may be specified quantity e.g. sea-ice concentration in tenths may be given as* `units="0.1f"`. *A dimensionless quantity with no scaling of offset may have* `units="1.0f"` *or* `units="unity"`.

NEW: The `long_name` is an attribute containing a descriptive name, which should not specify the `units`. Case is not significant in the `long_name`. This attribute is optional for most variables, but mandatory for coordinate variables of longitude, latitude, vertical axes and time (see sections 14, 15, 16 and 25). If the `long_name` attribute is present, it must be a quantity chosen from the list in Appendix D of this standard, optionally with additional information enclosed in parentheses `()` in the case that sufficient detail for local use cannot be given by a standardised description. The purpose of the list is to ensure that comparable data from different sources has the same description. Each quantity in Appendix D will be labelled with the version of the appendices at which it was introduced, enabling a generic application to deduce the complete set of quantities which was available to the application which generated the file.

Quantity attributes:

```
float tempt(pressure,lat,lon);
  tempt:long_name="atmospheric potential temperature "
    "(after timestep)";
  tempt:units="K";
```

`"atmospheric potential temperature"` *is the standard quantity, and* `"after time-`

step" is additional information, which a generic application can disregard.

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. `atmospheric potential temperature` *and* `soil temperature`*. In practice, the most specific description applicable should be used. We intend to expand Appendix D 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 new quantity is needed.*

NEW: The `subgrid` attribute (see section 22) can be regarded as a modifier of the `long_name`; it applies only to data variables, not coordinate variables. The `long_name` and `subgrid` attributes together define the physical dimensions of the quantity *(through information given in Appendices B and D)*, and the `units` must be consistent with this.

OGDT: The optional `modulo` attribute of a 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 (section 14), with a* `modulo` *of 360, and climatological axes of seasonal or diurnal phase (sections 27 and 30).*

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

OGDT: 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, which are codes identifying GCM diagnostic output quantities.*

13 Topology of an axis

OGDT: 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.

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

Note that the `topology` *and* `modulo` *attributes convey different information. For instance, a longitude coordinate variable limited to values in the eastern hemisphere between the Greenwich meridian and the date-line (e.g. 0E, 25E, 120E, 130E, 180E) does not have circular topology. (This might be from a model of a limited area of the world.) When making a contour map of a field with such a longitude axis, one can interpolate anywhere within the eastern hemisphere to draw the contours, but it is not legitimate to interpolate over the western hemisphere and draw the rest of the world, which is simply missing. The implication of circular topology would be that one could put any longitude at all on the*

left-hand side of the map. However, this coordinate variable does have a modulo (of 360, as required in section 14), and the points can be labelled in any way which is equivalent under the modulo to the coordinates in the file. The coordinate values 0,25,120,130,180 are thus equivalent to −360,−335,−240,−230,−180.

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

OGDT: 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.* A global longitude axis should have the attribute `topology="circular"`. *Note that the presence of a `modulo` attribute does not mean that the axis necessarily has circular topology (section 13); a longitude axis covering only part of the globe cannot have its points rotated.* COARDS: 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.

Global longitude axis:

```
float lon(lon);
  lon:long_name="longitude";
  lon:modulo=360.0f;
  lon:topology="circular";
  lon:units="degrees_east";
```

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`. OGDT: A latitude coordinate variable must have an attribute `long_name="latitude"`. COARDS: Such a variable is identifiable from its `long_name` and its `units`. *The COARDS convention relies on the unit as the only way to identify a latitude variable. This standard*

uses the `long_name`*, but requires the* `units` *to be specified as well for compatibility with COARDS.*

Latitude axis:

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

16 Vertical (height or depth) dimension

OGDT: 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, if there is one. NEW: The axis to be regarded as the vertical axis must have both a `long_name` attribute (section 12) and a `positive` attribute, with one of the allowed values `up` or `down`, to indicate the sense of the direction of positive *since this information may be useful for applications which display the data.*

Vertical pressure axis:

```
dimensions:
  pressure=15;
variables:
  float pressure(pressure);
    pressure:long_name="pressure";
    pressure:positive="down";
    pressure:units="hPa";
data:
  pressure=850, 700, 500, 300, 200, 150, 100, 50, 30, 20, 10;
```

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 makes the `positive` *attribute mandatory 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 the coordinate variable. This is inconsistent with the treatment of a data variable; the standard does not require that dimensionless units be invented for dimensionless physical quantities in data variables. Secondly, the `long_name` *attribute is more informative than the units. Thirdly, the vertical dimension of a data variable can be identified from the* `axis` *attribute or the the order of dimensions (see section 9), allowing any application which expects such a dimension to find it without any further help.*

We are not entirely convinced that the direction of positive should really be recorded as part of the data structure. It is mostly an issue for displaying the data, and is to some extent a matter of personal preference. If such special treatment is given to the vertical axis, why is it not also recorded for other axes? 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. Nonetheless, we have required the `positive` *attribute for compatibility with COARDS.*

For example, if an oceanographic netCDF file encodes the depth of the surface as 0 and the depth of 1000 m as 1000 then the axis would use attributes as follows: `units="m"`, `long_name="depth below the surface"`, `positive="down"`. *If, on the other hand, the depth of 1000 m were represented as* -1000, *we would have* `long_name="height above the surface"`, `positive="up"`.

17 Ancillary variables

NEW: "Ancillary" variables supply additional sets of values belonging somehow with other variables (usually coordinate variables), referred to as "main" variables. Ancillary variables are needed for various purposes described in sections 18, 19 and 20. These sections also define attributes of the main variables which enable applications to retrieve the ancillary variables. The dimensions of an ancillary variable must all be dimensions of its main variable.

18 Component variables

OGDT: A continuous physical variable may require more than one number to specify it at each point. We refer to these as "components". The values of the components are recorded in ancillary variables referred to as "component variables". The names of the component variables are recorded as a blank-separated list in a `component` string attribute of the main variable. The dimensions of a component variable must be identical with those of its main variable. *OGDT restricted components to coordinate variables, but the concept has here been generalised.*

OGDT: When a coordinate variable has 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, this main coordinate variable must be monotonic, but the components do not need to be monotonic.

Hybrid vertical coordinate: *A vertical coordinate* $\eta \equiv p/p_0 + \sigma$ *is used in some atmospheric GCMs. 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). *We would record this coordinate variable thus:*

```
float eta(eta); // main coordinate variable
  eta:component="pressure sigma";
float pressure(eta);
float sigma(eta);
```

A generic application would treat the component and main coordinates as independent information. The extra knowledge required to relate them would reside in any specific application which needed it.

19 Associated variables

NEW: An axis of a data variable, or two or more axes in combination, may have alternative sets of coordinate values. These alternative sets are recorded in ancillary variables of the data variable, and referred to as "associated" variables. The data variable records the names of associated variables as a blank-separated list in an `associate` string attribute. This attribute may alternatively and equivalently be named `coordinates`. *This possibility is included for compatibility with the CSM standard. However, in this standard it is deprecated because of possible confusion with the normal definition of "coordinate variable" (section 8) and because the use of associated variables is wider than just coordinate variables in the usual sense.* An associated variable may be named by more than one data variable in this way. If an associated variable itself has an `associate` attribute, variables named by this attribute are also regarded as being associated with the data variable. An associated variable must have dimensions which are all dimensions of any data variable with which it is associated; the associated variable can be regarded as a function of the indices along these axes. The values of an associated variable do not have to be monotonic. A generic application is not required to make any use of associated variables. Associated variables are not indicated in the `axis` attribute of the data variable (section 9).

Vertical axis: *Many associated variables will be one-dimensional, giving alternative sets of values for a single axis. One example is a vertical axis where one wishes to store both the physical coordinate and the ordinal model level number:*

```
dimensions:
  lat=90;
  sigma=19;
variables:
  float xwind(sigma,lat); // 2D data variable
    xwind:associate="model_level";
    xwind:axis="ZY";
  float lat(lat);
    lat:long_name="latitude";
  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";
```

Surface height: *Pressure is given at the surface, but we also wish to define the height of the surface itself, in case some subsequent application needs to convert to sea-level pressure, for example.*

```
  float pressure(lat,lon);
    pressure:associate="z";
    pressure:long_name="pressure at the surface";
    pressure:units="Pa";
  float z(lat,lon);
```

```
    z:long_name="height of the surface above sea level";
    z:units="m";
```

Trajectory: *The value of a quantity along a one-dimensional trajectory. In such a case, we might have a coordinate variable containing time of travel and associated coordinate variables giving the latitude and longitude of each point:*

```
dimensions:
  day=10; // 10 sample times along a trajectory
variables:
  float hice(day); // sea-ice thickness measured as the floe drifts
    hice:associate="lon lat";
    hice:axis="T";
  float day(day); // number of days since the beginning of the journey
  float lon(day); // longitude at each time
  float lat(day); // latitude at each time
```

The main coordinate variable (`day`*) must be monotonic, but the associated coordinates are not necessarily. An important application rather similar to this one is described in section 20. Note that* `lat` *and* `lon` *cannot be indicated as the* `X` *and* `Y` *coordinates in the* `axis` *attribute (section 9). This is reasonable because, even though they have the interpretations of latitude and longitude, they are not independent dimensions in the sense normally expected by an application which might look for such axes.*

An alternative convention would be to associate `lon` *and* `lat` *to the coordinate variable* `day` *instead of to the data variable. This would have the effect that all data variables that share the* `day` *axis would automatically share the associated coordinates, whereas in the current convention the association has to be repeated on each data variable. We choose to follow the current convention for two reasons. Firstly, it allows the possibility that the same main coordinate variable might occur in combination with various different sets of associated coordinate variables. In this example, there might be more than one trajectory, with the same* `day` *coordinates but different longitude–latitude positions; it is therefore more flexible to allow different associations for each data variable. Secondly, association on the data variable, rather than the coordinate variable, is the only possibility for multidimensional coordinate variables, such as we discuss in the next example.*

Transformed coordinates: *Associated variables of more than one dimension can be used to describe alternative coordinate systems. For example, vertical profiles of atmospheric humidity might be available on a regular longitude–latitude grid, but we might also wish to give the national grid coordinates of each point. The national grid x- and y-coordinates are each functions of both latitude and longitude; the x-coordinate does not correspond specifically to longitude, nor the y- to latitude. The appropriate representation is therefore:*

```
dimensions:
  lon=10;
  lat=20;
  pressure=15;
variables:
```

```
float humidity(pressure,lat,lon);
  humidity:associate="x y";
float pressure(pressure);
float lon(lon); // 1D main coordinate variable
  lon:long_name="longitude";
float lat(lat);
  lat:long_name="latitude";
float x(lat,lon); // 2D associated coordinate variable
  x:long_name="UK national grid eastings";
float y(lat,lon);
  y:long_name="UK national grid northings";
```

This tells us that `humidity[*][10][5]` *is the vertical profile of humidity at the point with latitude* `lat[10]` *and longitude* `lon[5]`*, which is at national grid x-coordinate* `x[10][5]` *and y-coordinate* `y[10][5]`*.*

No main coordinate variables: *A related situation is when the 2D grid is staggered or transformed geometrically in some way (other than rotation—see section 10), so that it is not possible or easy to give one-dimensional coordinate variables for the axes. In this case, there would be no main coordinate variables, and the default of plain indexes would apply. The physical coordinates are functions of the 2D gridpoint indices, and would be given in associated variables just as above:*

```
dimensions:
  x=90;
  y=45;
variables:
  float orog(y,x); // 2D variable on a horizontal grid
    orog:associate="lon lat";
    orog:axis="--";
    orog:long_name="height of the surface above sea-level";
    orog:units="m";
  float lon(y,x); // 2D coordinate variable on the same grid
  float lat(y,x);
```

The `lat` *and* `lon` *variables are not indicated as the* `X` *and* `Y` *coordinates in the* `axis` *variable (section 9). An application which referred to latitude and longitude coordinates would not generally expect them to be two-dimensional. If it can handle this situation, it should identify these axes by their* `long_name` *and* `units`*.*

3D associated coordinates: *These could be used if one wished to describe a field with alternative 3D coordinate systems, for instance on both a regular Cartesian grid, and in cylindrical or spherical coordinates. The values of the alternative coordinates would be given on the Cartesian grid. In the spherical case, for instance*

```
float temperature(z,y,x); // 3D variable on a Cartesian grid
  temperature:associate="radius theta phi";
float radius(z,y,x);
float theta(z,y,x);
float phi(z,y,x);
```

NEW: A particular technical application of a one-dimensional associated coordinate is to deal with the limitation of netCDF to a single unlimited dimension. If several data variables have unlimited axes of different lengths or physical significance, they can all share a nominal unlimited dimension, and each have associated variables specifying the meaning of the axis.

More than one unlimited axis: *Consider a file which contains data variables with unlimited axes measuring elapsed time with different sampling frequency, and hence of different lengths.*

```
dimensions:
  time_counter=UNLIMITED;
variables:
  float sw(time_counter); // sampled every 3 hours
    sw:associate="time_3h";
    sw:axis="T";
    sw:long_name="vertical component of "
      "shortwave radiative flux density";
    sw:units="W m-2";
  float latent(time_counter); // sampled every 30 minutes
    latent:associate="time_30min";
    latent:axis="T";
    latent:long_name="latent heat flux density";
    latent:units="W m-2";
  float time_3h(time_counter)";
    time_3h:long_name="elapsed time";
    time_3h:units="h";
  float time_30min(time_counter);
    time_30min:long_name="elapsed time";
    time_30min:units="min";
```

20 Bundles

OGDT: 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.

OGDT: 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.

Timeseries: *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. Thus:

```
dimensions:
  points=15; // measurement locations
  times=20; // sampling times
variables:
  float snowdepth(times,points);
    snowdepth:associate="lon lat sitename";
    snowdepth:axis="T-";
  float lon(points); // longitude of sites
  float lat(points); // latitude of sites
  char sitename(points,StringMaxLength); // string array of sitenames
  double times(times); // times of measurement
```

See section 24 concerning the time coordinates. This same form could be used for observed timeseries from stations. The bundling axis (`points`) is simply an index.

Vertical profiles: *A similar application is that of vertical profiles at sets of points; for example, scattered vertical temperature profiles through the ocean, or data from various radiosonde stations.*

```
dimensions:
  station=10; // measurement locations
  pressure=11; // pressure levels
variables:
  float humidity(pressure,station);
    humidity:associate="lon lat";
    humidity:axis="Z-";
  int station(station); // station numbers
  float lon(station); // longitude of stations
  float lat(station); // latitude of stations
  float pressure(pressure)
```

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 9). 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 variables

OGDT: 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 point coordinate values. The convention is to define an additional two-dimensional "boundary 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. Supplying upper and lower boundaries separately allows for the possibility that the cells might not be contiguous; they might even overlap. If a lower boundary value is equal to the `valid_min` for the coordinate variable (section 31), the cell has no lower boundary. If an upper boundary value is equal to the `valid_max`, the cell has no upper boundary. The name of the boundary 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_`.

Boundaries for a one-dimensional latitude coordinate variable:

```
float lat(lat);
  lat:bounds="bounds_lat";
float bounds_lat(2,lat);
```

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)`*.*

Albedo as a function of wavelength and snow cover: *Characteristic values of albedo are given for various wavelength bands, dependent also on snowdepth.*

```
dimensions:
  lambda=4; // number of shortwave frequency bands
  snowdepth=10; // number of snowdepth categories
variables:
  float albedo(lambda,snowdepth); // no units for albedo
    albedo:axis="--";
    albedo:long_name="surface albedo";
  float lambda(lambda);
    lambda:bounds="bounds_lambda";
    lambda:long_name="wavelength";
    lambda:units="nm";
  float bounds_lambda(2,lambda);
  float snowdepth(snowdepth);
    snowdepth:bounds="bounds_snowdepth";
    snowdepth:long_name="mass per unit area of lying snow";
    snowdepth:units="kg m-2";
    snowdepth:valid_max=1e9;
  float bounds_snowdepth(2,snowdepth);
data:
  lambda=250, 385, 570, 795;
  bounds_lambda=175, 320, 450, 690,
                320, 450, 690, 900;
```

```
  snowdepth=0.05, 0.15, 0.35, 0.75, 1.25, 1.75, ..., 450.0, 1000.0;
  bounds_snowdepth=0.0, 0.1, 0.2, 0.5, 1.0, 1.5, ..., 400.0, 500.0,
                   0.1, 0.2, 0.5, 1.0, 1.5, 2.0, ..., 500.0, 1e9;
```

A first index of 0, for instance, gives albedo values for the wavelength range 175–320 nm. The deepest snowdepth class has no upper bound; any value above 500 falls into this class.

Probability density function of precipitation amounts:

```
dimensions:
  ppn=10;
variables:
  float pdf(ppn,lat,lon);
    pdf:axis="-YX";
    pdf:long_name="probability density of "
      "depth of water-equivalent precipitation";
    pdf:units="mm-1";
  float ppn(ppn);
    ppn:units="mm";
    ppn:long_name="depth of water-equivalent precipitation";
    ppn:bounds="bounds_ppn";
  float bounds_ppn(2,ppn);
data:
  bounds_ppn=0.0, 0.1, 0.2, 0.5, ..., 0.1, 0.2, 0.35, 1.0, ...;
```

`pdf[3][10][12]` *gives the probability density of precipitation amounts between 0.5 and 1.0 mm falling at the location* `lat[10] lon[12]`.

OGDT: Boundary 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 variables may be supplied for ancillary 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.

Boundary values for a hybrid vertical coordinate: *The atmospheric column is here divided into three cells in the vertical; from the surface to* $\sigma = 0.7$*; from there to 20 kPa, and finally to the top of the atmosphere, using the hybrid vertical coordinate introduced in an example in section 18.*

```
dimensions:
  eta=3;
variables:
  float(eta);
    eta:long_name="pressure-sigma hybrid";
    eta:component="pressure sigma";
    eta:bounds="bounds_eta";
    eta:positive="down";
  float bounds_eta(2,eta);
  float pressure(eta);
```

```
    pressure:units="kPa";
    pressure:long_name="pressure";
    pressure:bounds="bounds_pressure";
  float bounds_pressure(2,eta);
  float sigma(eta);
    sigma:long_name="sigma";
    sigma:bounds="bounds_sigma";
  float bounds_sigma(2,eta);
data:
  eta=0.75, 0.45, 0.1;
  bounds_eta=1.0, 0.7, 0.2,
             0.7, 0.2, 0.0;
  pressure=0.0, 10.0, 10.0;  // does not need to be monotonic
  bounds_pressure=0.0, 0.0, 20.0;
                  0.0, 20.0, 0.0;
  sigma=0.75, 0.35, 0.0;
  bounds_sigma=1.0, 0.7, 0.0;
               0.7, 0.0, 0.0;
```

NEW: Boundary variables may be given for associated multidimensional coordinate variables (section 19). Each dimension of the main variable requires an extra dimension of size 2 in the boundary variable. These extra dimensions are placed on the left *(right in Fortran terms)* of the coordinate dimensions, and in the same order as the coordinate dimensions.

Boundaries for a two-dimensional latitude coordinate variable:

```
  float lat(y,x);
    lat:bounds="bounds_lat";
  float bounds_lat(2,2,y,x);
```

so `bounds[0][0][4][5]` *contains the latitude of the lower left (smaller x and y) corner of gridbox* `[4][5]`, `bounds[0][1][4][5]` *the lower right corner,* `bounds[1][0][4][5]` *upper left and* `bounds[1][1][4][5]` *upper right. In Fortran, the indices of the boxes would be* `(6,5,1,1)`, `(6,5,2,1)`, `(6,5,1,2)`, `(6,5,2,2)` *respectively.*

22 Representation of subgrid variation

NEW: Since a data variable usually represents a physical quantity which varies continuously along the axes, in reality there will generally be variation of the quantity between adjacent gridpoints. The data variable can give only one value for each cell, despite this subgrid variation. For many purposes, this can be taken as a "representative" value, and it is not necessary to define precisely how it relates to the subgrid variation.

NEW: To be explicit about how each data value reflects subgrid variation along a particular axis, use the `subgrid` attribute of the data variable. *The most important application of this attribute is to contracted or collapsed axes, described in section 23.* This is a string attribute comprising a list of blank-separated words. In this list, "name:

method" indicates that subgrid variation along the axis with the dimension whose "name" is given is represented by the specified "method". The method, which may be several words, should be one of the permitted values detailed in Appendix B, which include `mean`, `maximum`, `minimum`, `mid-range`, `standard deviation`, `variance`, `mode`, `median`, `cell`, `point`. Case and punctuation are not significant in the method. *Like Appendix D, Appendix B will be expanded on request by users of this standard.* Some methods imply a change of units of the data variable, and this also is specified by Appendix B. *In the above list, this is true for* `variance`. The method `point` indicates that the data values apply exactly at the coordinate values, and do not at all represent the variation between adjacent gridpoints along the axis concerned. The method `cell` indicates that each value should be regarded as a property of the whole cell along the axis concerned e.g. a sum or integral. The method can be differently specified for the various dimensions. *It must be remembered that the method applies only to the axis indicated. If a precipitation value in a longitude–latitude gridbox is given the method* `maximum` *for these axes, for instance, it means that it is the maximum within these spatial cells, and does not imply that it is also the maximum in time.*

NEW: The absence of any specification means that generic applications may regard the data values as representative in whatever way suits their needs. *For quantities calculated at gridpoints by numerical models, this kind of vagueness is unavoidable. If a model provides a longitude–latitude field of temperature at gridpoints, an application used to draw a contour plot of the field will generally assume that the temperatures apply at points, and will use some interpolation scheme to compute values between them. An application which calculates the mean of the field, however, will probably assume that the temperatures are gridbox means, and average them by weighting each with its area. Both of these approaches are valid. The finite-difference scheme by definition does not have any information about subgrid variation, and may itself treat the values in both ways; it might calculate gradients between them, regarding them as points, or enforce conservation properties, regarding them as means. It would be unusual to regard the values as extrema, however, unless this was explicitly indicated.*

Subgrid time variation in timeseries: *Consider 12-hourly timeseries of pressure, temperature and precipitation from a number of stations, where pressure is measured instantaneously, temperature extremes over the preceding period are recorded by maximum and minimum thermometers, and precipitation is accumulated in a rain gauge. For a period of 48 hours from 6 a.m. on 19th April 1998, the data is structured as follows:*

```
dimensions:
  instanttime=5; // 5 instantaneous measurements at 12-hour intervals
  periodtime=4; // 4 intervening 12-hour periods
  station=10;
variables:
  float pressure(station,instanttime);
    pressure:axis="-T";
    pressure:long_name="pressure";
    pressure:subgrid="instanttime: point";
    pressure:units="kPa";
  float maxtemp(station,periodtime);
    maxtemp:axis="-T";
    maxtemp:long_name="temperature";
```

```
    maxtemp:subgrid="periodtime: maximum";
    maxtemp:units="K";
  float ppn(station,periodtime);
    ppn:axis="-T";
    ppn:long_name="depth of water-equivalent precipitation";
    ppn:subgrid="periodtime: cell";
    ppn:units="mm";
  double instanttime(instanttime);
    instanttime:long_name="time";
    instanttime:units="h since 1998-19-4 6:0:0";
  double periodtime(periodtime);
    periodtime:bounds="bounds_periodtime";
    periodtime:long_name="time";
    periodtime:units="h since 1998-19-4 6:0:0";
  double bounds_periodtime(2,periodtime);
data:
  instanttime=0., 12., 24., 36., 48.;
  periodtime=6., 18., 30., 42.;
  bounds_periodtime= 0., 12., 24., 36.,
                     12., 24., 36., 48.;
```

It is not appropriate give a subgrid method for the station axis, since this is a bundling axis (section 20) and not a continuous physical coordinate. The instantaneous and period measurements have to have different time axes both because of their different dimension, and because they do not coincide. If the pressure measurements were made at times halfway between the others (noon and midnight), the time axes could be shared. Since the precipitation is given as an amount, it is a sum over the interval of time by definition. It could instead have been expressed as a rate in `mm h-1`*, for instance, in which case its subgrid method would be* `mean` *rather than* `cell`*.*

Thickness (geopotential difference): *The "thickness" is the difference in geopotential height between two pressure surfaces in the atmosphere. This quantity is by definition one which relates to the whole extent of its cell in the vertical dimension.*

```
variables:
  float thickness(pressure,lat,lon);
    thickness:long_name="thickness";
    thickness:subgrid="pressure: cell";
    thickness:units="m2 s-2";
  float pressure(pressure);
    pressure:bounds="bounds_pressure";
    pressure:long_name="pressure";
    pressure:units="hPa";
  float bounds_pressure(2,pressure);
```

Here, `bounds_pressure[0][0]` *and* `bounds_pressure[1][0]` *will be the upper and lower pressure bounds of the thickness field* `thickness[0][*][*]`*.*

NEW: If more than one subgrid method is to be indicated, they should be arranged in the order of the dimensions of the data variable, unless their order is significant and

different from this. The right-most operation is assumed to have been applied first. *Suppose a quantity varies in both longitude and time (dimensions* `lon` *and* `time`*) within each gridbox. Values which represent the time-average of the zonal maximum are labelled* `subgrid="time: mean lon: maximum"`*, 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 labelled* `subgrid="lon: maximum time: mean"`*.*

NEW: 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, which should appear in the order of the dimensions of the data variable, though this order is not significant. Dimensions should be grouped in this way only if there is an essential difference from treating them individually. *For instance, the subgrid standard deviation of topographic height within a longitude–latitude gridbox would have* `subgrid="lat: lon: standard deviation"`*. This is not the same as* `subgrid="lat: standard deviation lon: 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.*

NEW: To indicate more precisely how the `subgrid` method was applied, extra information may be included in parentheses `()` after the identification of the method. This information is not standardised and may be ignored by a generic application. *A mean over latitude, for instance, may be area-weighted. This could be indicated as* `"lat: mean (area-weighted)"`*.*

NEW: The `subgrid` attribute cannot be used to show how a value reflects variation over a coordinate which does not have a dimension in the data variable. This should be done in the `long_name` instead. It is generally more informative and precise to introduce a singleton dimension specifically for this purpose, however. *For example, we could describe a quantity in its* `long_name` *as being simply a temporal variance, but it would be more informative to record it as a subgrid method, by giving the variable a singleton time dimension, which could also be used to should the range of times it covers and the time-interval of the data from which the variance was calculated. See also section 23.*

23 Contracted dimensions

NEW: A contracted axis is one which is formed by aggregating the values of an axis with a larger dimension into a smaller number of groups. In the commonest case, the dimension is collapsed completely to a singleton dimension (i.e. a size of unity, section 9), where all data points share the entire collapsed axis. OGDT: The collapsed dimension indicates the relationship of the data variable which is being described to another variable of higher dimensionality. NEW: The boundaries of the cells along a contracted axis will be the outside boundaries of the groups of cells along the uncontracted axis, or the outside coordinates if boundaries were not given. The main coordinate values of a contracted axis will be values representative of the coordinate ranges spanned by the groups. OGDT: A collapsed dimension has a single representative main coordinate value and boundary coordinate values supplying the complete range of the uncollapsed axis. NEW: These boundaries will be the extreme boundary coordinate values of the uncollapsed axis, or the extreme main coordinate values if boundaries were not supplied. *A very important application of collapsed axes is to indicate climatological time. This is discussed in section*

30.

NEW: The `subgrid` attribute (section 22) of the data variable with contracted axes can be used to indicate how the data values of the variable with uncontracted axes were aggregated to reduce the dimensions. The new `subgrid` information will be prefixed to the existing attribute, if any, indicating the name of the newly contracted dimension. Any existing references to the uncontracted dimension in the `subgrid` attribute should be modified to refer to the contracted dimension, since the uncontracted dimension will no longer be a dimension of the data variable.

As explained in section 22, this attribute will indicate that the data value is the mean, maximum, minimum, etc. The allowed `subgrid` *"methods" are listed in Appendix B, which will be expanded as need arises. As foreseen at the moment, the idea is limited to operations which give a single value representative of each contracted group of values, without reference to any external constants. For example, the number which exceeds 20% of the values in the group, or equivalently the 20th percentile, is a single number representing the group, but the procedure of finding it is not treated as a* `subgrid` *method because it requires the constant 0.2 to define it. Instead, the relationship of this new variable to the old should be shown by changing its* `long_name` *to indicate that it is a percentile value, and giving it a new singleton percentage axis with value 20, or cumulative probability with value 0.2. This kind of transformation is analogous to reducing a variable on three spatial dimensions (say) to two by extracting its values on a specified surface. The contraction or collapse is a special case, because, in general, the percentile axis need not have a size of unity; it might be a new multi-valued axis (in cumulative probability) replacing the old one (in some spatial dimension, for instance). This is like regridding a vertical axis of height onto pressure. Having said all this, however, we note that* `median` *is in fact a named instance of this operation—extraction of the 50th percentile—but we allow it on the grounds that it is a common method for choosing a single representative value.*

Singleton axes are not necessarily the result of collapsing an axis. In section 9, we recommend singleton axes as the means of attaching characteristic single physical values to a data variable, for instance the height or pressure of the surface on which a variable is supplied. If no `subgrid` *method is specified, the application knows only that the single value characterises the data in some way. All information in the* `subgrid` *attribute is entirely optional. For instance, a time-mean quantity should generally have a singleton time dimension to indicate the range of times to which it applies, but it is not mandatory to indicate in the* `subgrid` *attribute that it is a mean over time.*

NEW: On the coordinate variable of a contracted axis, the optional `interval` attribute specifies the typical spacing between two adjacent coordinates of the uncontracted axis, where "typical" is not well defined. Further information may be given by the the optional `spacing` attribute, which may have value `uniform`, indicating that the coordinates were evenly spaced with the `interval` specified (if any) and the cells contiguous, or `variable`, if they were not evenly spaced but still contiguous, or `disjoint`, which means there may have been gaps between them. The coordinates of the uncontracted axis may be explicitly recorded in separate variables; if so, the main uncontracted coordinate variable should be named by the attribute `expand` of the main contracted coordinate variable.

Area-averaging a longitude–latitude field to one of lower resolution: *The original resolution was 1 degree, and the field has been averaged into 10-degree boxes.*

```
dimensions:
```

```
  con_lat=18; // contracted dimension
  con_lon=36;
  lat=180; // original uncontracted dimension
  lon=360;
variables:
  float sst(con_lat,con_lon);
    sst:long_name="sea surface temperature";
    sst:subgrid="con_lat: mean con_lon: mean";
    sst:units="degC";
  float con_lat(con_lat); // contracted latitude axis
    con_lat:bounds="bounds_con_lat";
    con_lat:expand="lat";
    con_lat:interval=1.0f; // original resolution in latitude
    con_lat:long_name="latitude";
    con_lat:units="degree_north";
  float bounds_con_lat(2,con_lat);
  float lat(lat); // original uncontracted latitude axis
    lat:bounds="bounds_lat";
  float bounds_lat(2,lat);
data:
  con_lat=-85, -75, -65, ...;
  bounds_con_lat=-90, -80, -70, ..., 80, -80, -70, -60, ..., 90;
  lat=-89.5, -88.5, -87.5, ...;
  bounds_lat=-90, -89, -88, ..., 89, -89, -88, -87, ..., 90;
```

Instead of an area-average, the contracted field might instead have represented the subgrid spatial variation of SST. In that case, `subgrid="con_lat: con_lon: standard deviation"`.

Mean over time and longitude: *Here, the time-mean zonal-mean humidity is given as a function of latitude and height. The means have been formed over the complete time and longitude intervals of the original data, so these dimensions are collapsed.*

```
dimensions:
  con_lon=1; // collapsed longitude dimension
  con_time=1; // collapsed time dimension
  lon=72;
  sigma=6;
variables:
  float humidity(con_time,sigma,lat,con_lon);
    humidity:long_name="specific humidity";
    humidity:subgrid="con_time: mean con_lon: mean";
  double con_time(con_time);
    con_time:bounds="bounds_con_time";
    con_time:interval=0.125; // original data was at intervals of 3 h
    con_time:units="days as %Y%m%d.%f";
  float bounds_con_time(2,con_time);
  float con_lon(con_lon);
    con_lon:bounds="bounds_con_lon";
```

```
    con_lon:long_name="longitude";
    con_lon:modulo=360f;
    con_lon:topology="circular";
    con_lon:units="degree_east";
  float bounds_con_lon(2,con_lon);
  float sigma(sigma);
    sigma:bounds="bounds_sigma";
    sigma:long_name="sigma";
  float bounds_sigma(2,sigma);
data:
  con_time=19960901.0;
  bounds_con_time=19960301.0, 19970301.0;
  con_lon=180;
  bounds_con_lon=0, 360;
  sigma=0.99, 0.96, 0.92, 0.8, 0.5, 0.1;
  bounds_sigma=1.00, 0.98, 0.94, 0.86, 0.65, 0.30, // decreasing, like
               0.98, 0.94, 0.86, 0.65, 0.30, 0.05; // main coordinate
```

This is a mean over the complete range of longitude from 1 March 1996 to 1 March 1997 (see section 27 concerning the time coordinate). The longitude axis indicates circular topology because this was the case before it was collapsed; after collapse, the topology is not really meaningful. If the humidity was subsequently meaned over the depth of the atmosphere as well, `subgrid` *would be prefixed with* `con_sigma: mean`*, and* `con_sigma` *would have bounds 1.00 and 0.05.*

NEW: If the same axis is contracted repeatedly, the methods may all be recorded in the `subgrid` attribute of the data variable, but only the most recent `interval` and `spacing` will be shown on the contracted coordinate variable. But if the axis before contraction is retained in the file (identified by an `expand` attribute), and was itself the result of a contraction, it can record the previous `interval` and `spacing`.

NEW: Repeated operations of some methods can be regarded as equivalent to a single operation. *For instance, meaning longitude cells of 1 degree width to 5 degrees, and then from 5 to 45 degrees, gives the same result as meaning in one step from 1 degree to 45 degrees (apart from complications with missing data). Similarly, meaning a time axis from days into months, then into seasons, and finally into years could be represented as a single operation of meaning from days to years.* In that case, the `subgrid`, `interval` and `spacing` attributes need not be modified for successive operations. The choice of whether to take this approach is left to the application.

24 Time variables and intervals

OGDT: A "time variable" is one which represents date and time, which we will refer to hereafter just as "time". An "interval of time" is the difference between two times.

OGDT: It would be possible to describe time in terms of six components (year, month, day, hour, minute, second) in a netCDF file, using six component variables of various data types. However, it is more efficient and for many purposes more convenient to represent a time as a single number, giving the elapsed interval since a certain reference time, which

may be either implicit or explicit. NEW: We refer to conversion from the components of a time into a single number as "encoding", and the reverse as "decoding". Encoding and decoding are complicated because year and month are units with lengths that depend on the date and the calendar in use, so special provisions are needed for time axes.

OGDT: 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 in units of fixed length (days, hours, minutes, seconds) between two times 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 one month, and equals 29 days in the standard 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. This standard permits the use of the standard calendar (below, section 28) and of other calendars (section 29). The `calendar` attribute, described in the following sections, indicates the calendar in use. If a time coordinate variable has no `calendar` attribute, the global `calendar` attribute (section 5), if present, applies to it.

NEW: This standard permits two different methods, distinguished by their `units`, of encoding a time into a number. These methods, referred to as "relative time" and "absolute time", are described in the following sections (26 and 27). Relative time is a more familiar method, but absolute time offers important advantages.

OGDT: Time variables may have an attribute `time_format`, to specify a format for printing the date and time, according to the conventions of the Unix (TM) `date` command.

25 Time dimension

COARDS: Time coordinate variables must always explicitly include the `units` attribute; there is no default value. OGDT: The `long_name` attribute is also mandatory. A time coordinate variable will be identifiable by its `units` and `long_name`.

26 Relative time

NEW: A time encoded as a relative time gives the elapsed interval since a specified reference time; `units` takes the form "time-unit `since` reference-time", as per the recommendations of the Unidata udunits package (but see below concerning the time-unit) *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).* In order to decode the values on a relative time axis, the application will in general need to know the calendar; the encoded time values are meaningless without this knowledge. Furthermore, a given date may result in different time values when encoded in two different calendars with the same `units`. *For instance,* `1996-2-1 15:00:00` *is* `62.625 days since 1995-12-1 0:0:0` *in the standard calendar, and* `60.625 days since 1995-12-1 0:0:0` *in the 360-day calendar.*

OGDT: The file `udunits.dat` defines second, minute, hour and day as units of time. Units of months and years are disallowed by Appendix C of this standard, because they are not well-defined; since udunits defines a year as a "tropical year" of 31556925.97 s (674.03 s less than 365 days) and a month as exactly a twelfth of a year, use of these units will probably not give the expected results. *For example,* `1 month since 1995-4-1 0:0:0` *is treated by udunits as* `30.4368 days since 1995-4-1 0:0:0`*, which is approximately* `1995-5-1 10:29`*, not* `1995-5-1 0:0:0`*. Also,* `1 year since 1995-4-1 0:0:0` *is about* `1996-3-31 5:49`*, not* `1996-4-1 0:0:0`*.* NEW: The udunits unit `common_year` (exactly 365 days) is permitted, but not recommended.

A relative time axis for instantaneous measurements of a quantity: *Measurements are made at noon on 2nd–5th June 1996.*

```
dimensions:
  time=4;
variables:
  double time(time);
    time:long_name="time";
    time:units="days since 1996-1-1 0:0:0";
data:
  time=1.5, 2.5, 3.5, 4.5;
```

A relative time axis for monthly means: *Means are calculated for February, March and April of 1990.*

```
dimensions:
  time=3;
variables:
  double time(time);
    time:bounds="bounds_time";
    time:long_name="time";
    time:units="days since 1990-1-1 0:0:0";
  double bounds_time(2,time);
data:
  time=45.0, 74.5, 105.0;
  bounds_time=31.0, 59.0, 90.0,
              59.0, 90.0, 120.0;
```

In this example, the main time coordinates are merely representative values, being the mid-points of their respective months. Decoded, they are `1990-2-15 0:0:0`*,* `1990-3-16 12:0:0` *and* `1990-4-16 0:0:0`*.*

27 Absolute time

NEW: This method of encoding time refers to the separate components of time, rather than to a single unit of fixed length. It offers two advantages. Firstly, the encoded times are meaningful and can be decoded into components of time without knowledge of the

calendar, although to calculate intervals between them this knowledge is still required. Secondly, "partial" times can be encoded, which omit the year, or the "seasonal phase" (time of year, time within the seasonal cycle), or the "diurnal phase" (time of day, time within the diurnal cycle). By contrast, relative times can only be "complete" times, which include information about all three of these.

NEW: The `units` attribute of absolute time takes the form "time-unit `as` time-string", The possibilities with the recommended data types and their meanings are as follows:

Format	Data type	Interpretation
`second as %S.%f`	`float`	Diurnal phase
`minute as %M.%f`	`float`	Diurnal phase
`hour as %H.%f`	`float`	Diurnal phase
`day as %Y%m%d.%f`	`double`	Time
`day as %Y%m%d`	`int`	Year and seasonal phase
`day as %m%d.%f`	`double`	Seasonal phase and diurnal phase
`day as %m%d`	`int`	Seasonal phase
`day as .%f`	`float`	Diurnal phase
`calendar_month as %Y%m.%f`	`double`	Year and seasonal phase
`calendar_month as %Y%m`	`int`	Year and seasonal phase
`calendar_month as %m.%f`	`float`	Seasonal phase
`calendar_month as %m`	`int`	Seasonal phase
`calendar_year as %Y.%f`	`double`	Year and seasonal phase
`calendar_year as %Y`	`int`	Year
`calendar_year as .%f`	`float`	Seasonal phase

Standard abbreviations and plural forms of the unit names are acceptable, as usual. The time-units `calendar_year` and `calendar_month` are units of time defined by this standard (Appendix C).

NEW: The time-string codes show how the year, month, day within month and time within day are encoded into a single number, after the fashion of the Unix (TM) `date` and `printf` commands, thus:

Format letter	Interpretation
`%Y`	Year (including century)
`%m`	Two-digit month (01=January)
`%d`	Two-digit day within month
`%H`	Hours since midnight
`%M`	Minutes since midnight
`%S`	Seconds since midnight
`%f`	Floating-point fraction of the specified time-unit
`.`	Position of decimal point

Since an encoded time is an ordinary number, leading zeroes in the integer part may be omitted.

In absolute time, 3 p.m. on 5th April 1998 is encoded with value 19980405.625 and `units="day as %Y%m%d.%f"`*. The advantage of this method of encoding a complete time is that it can be done without knowledge of the calendar, whereas if we encoded in relative time units of* `days since 1900-1-1`*, the value would be 35888.625 in the standard*

calendar, and 35374.625 in the 360-day calendar. We also know, without reference to the calendar, that the value 19980605.625, with the same units, is a time exactly two calendar months later, and 19970405.625 is exactly one calendar year earlier. But to calculate these intervals in other time units—days, hours, etc.—we still need to know the calendar.

NEW: The only complete form of absolute time is `"day as %Y%m%d.%f"`. Note in particular that the forms `"calendar_month as %Y%m.%f"` and `"calendar_year as %Y.%f"` are partial times which imply no information about the diurnal phase. *This is a very important point. For instance,* `1998.25 calendar_year as %Y.%f` *means no more than "a quarter of the way through 1998 as regards the seasonal cycle". This meaning is the same in the standard and 360-day calendars. Because this representation carries no information about diurnal phase, it is not permitted to decode it to* `1998-4-2 3:0:0` *(i.e. 91.25 days from the start of the year) in the standard calendar or* `1998-4-1 0:0:0` *in the 360-day calendar. Similarly,* `199804.3 calendar_month as %Y%m.%f` *means "30% of the way through April 1998 as regards the seasonal cycle". Examples below show the use of such partial times.* Note also that the only form of partial time which is composed of the seasonal and diurnal phases is `"day as %m%d.%f"`; there is no method of encoding the seasonal phase as a fraction of a calendar year or month in combination with the diurnal phase. Should this be required, the application could construct it as a two-component time variable. *This exclusion seems reasonable because data which resolves both the seasonal and diurnal cycles must belong to a known calendar (it will exhibit a certain number of days in a year, for instance) and so its seasonal cycle can be labelled by month and day. The calendar-independent representations of the seasonal cycle, shown in examples below, are more useful when portions of the seasonal cycle have been averaged, in which case the seasonal and diurnal cycles, if both present, will be on separate axes.* The forms `"calendar_month as %Y%m"` and `"calendar_month as %m"` are shorthands, respectively, for `"calendar_month as %Y%m.%f"` and `"calendar_month as %m.%f"`, with the fraction assumed to be exactly 0.5 (i.e. the middle of the month) in a main variable, and exactly 0.0 (i.e. the beginning of a month) in a boundary coordinate variable.

An absolute time axis for instantaneous measurements of a quantity: *Measurements are made at noon on 2nd–5th June 1996.*

```
dimensions:
  time=4;
variables:
  double time(time);
    time:long_name="time";
    time:units="days as %Y%m%d.%f";
data:
  time=19960602.5, 19960603.5, 19960604.5, 19960605.5;
```

An absolute time axis for monthly means, encoded in days:

```
dimensions:
  time=3;
variables:
  double time(time);
    time:bounds="bounds_time";
```

```
    time:long_name="time";
    time:units="days as %Y%m%d.%f";
  double bounds_time(2,time);
data:
  time=19900215.0, 19900316.5, 19900416.0;
  bounds_time=19900201.0, 19900301.0, 19900401.0,
              19900301.0, 19900401.0, 19900501.0;
```

As in the relative time version of this example, the main time coordinates are the mid-points of their respective months. Although they are encoded straightforwardly, their values depend on the calendar. If one was comparing means over these months from data sources which used different calendars, that might be inconvenient, and could be avoided as in the next example.

An absolute time axis for monthly means, encoded in months:

```
dimensions:
  time=3;
variables:
  double time(time);
    time:bounds="bounds_time";
    time:long_name="year and seasonal phase";
    time:units="calendar_months as %Y%m.%f";
  double bounds_time(2,time);
data:
  time=199002.5, 199003.5, 199004.5;
  bounds_time=199002.0, 199003.0, 199004.0,
              199003.0, 199004.0, 199005.0;
```

This method shows directly that the main coordinates are half-way through their months. The units could alternatively be given as `calendar_months as %Y%m`*, which is a short-hand. The coordinates would then be*

```
data:
  time=199002, 199003, 199004;
  bounds_time=199002, 199003, 199004,
              199003, 199004, 199005;
```

These have the same interpretation as the above, by convention.

A partial time defining just the year: *An axis of this kind could be used to record the number of occurrences of a particular kind of event:*

```
dimensions:
  year=3;
variables:
  int year(year);
    year:long_name="year";
    year:units="calendar_year as %Y";
```

```
  int count(year);
data:
  year=1991,1992,1993,1994,1995;
  count=0,2,1,0,1;
```

Year and seasonal phase defined in calendar years: *By contrast to the last example, if it was appropriate to indicate that each count applied to the whole of the continuous period of time of its respective year, this could be done thus:*

```
variables:
  double year(year);
    year:bounds="bounds_year";
    year:long_name="year and seasonal phase";
    year:units="calendar_year as %Y.%f";
  int count(year);
data:
  year=1991.5, 1992.5, 1993.5, 1994.5, 1995.5;
  bounds_year=1991.0, 1992.0, 1993.0, 1994.0, 1995.0,
              1992.0, 1993.0, 1994.0, 1995.0, 1996.0;
  count=0,2,1,0,1;
```

The use of floating-point years allows us conveniently to represent exactly the beginning and ending of a year and a point half-way through. In the standard calendar, of course, the interval from 1992.0 to 1993.0 is longer in relative time than all the other years. But for some purposes, it might be more useful to record that each interval is a calendar year. This could be especially helpful when comparing data from different calendars.

Seasonal phase as a function of year: *Here we show the date within the year of a particular event, such as the highest daily maximum temperature, or the onset of the monsoon, as a partial time within its year.*

```
dimensions:
  year=5;
variables:
  int year(year);
    year:long_name="year";
    year:units="calendar_year as %Y";
  int date(year);
    date:long_name="seasonal phase";
    date:units="day as %m%d";
data:
  year=2011, 2013, 2027, 2028, 2051;
  date=629, 627, 626, 703, 710;
```

The event concerned occurred on 29th June 2011, 27th June 2013, 26th June 2027, 3rd July 2028 and 10th July 2051. Clearly the date variable could have been encoded as a complete time, perhaps in relative time units, but this would have included redundant year information.

NEW: A time variable which indicates seasonal phase but not year has a modulo of one year. If it spans the entire seasonal cycle, it also has circular topology. Similarly, a time variable indicating diurnal phase but not seasonal phase has a modulo of one day, and has circular topology if it spans the entire diurnal cycle. *These kinds of time coordinate are particular useful for representing climatological time, in conjunction with other contracted time axes. See section 30.*

Average seasonal cycle expressed in months: *Data for solar radiation as 3-monthly averages.*

```
dimensions:
  time=4;
  lat=72;
  lon=96;
variables:
  float sol(time,lat,lon);
    sol:long_name="vertical component of "
      "solar radiative flux density";
    sol:units="W m-2";
  float time(time);
    time:bounds="bounds_time";
    time:long_name="seasonal phase";
    time:modulo=12.0f;
    time:topology="circular";
    time:units="calendar_month as %m.%f";
  float bounds_time(2,time);
data:
  time=10.5, 13.5, 16.5, 19.5;
  bounds_time= 9.0, 12.0, 15.0, 18.0,
               12.0, 15.0, 18.0, 21.0;
```

The first time point applies from the beginning of month 9 to the beginning of month 12, i.e. September to November inclusive. A representative main coordinate is given of half-way through October. The second point runs to the beginning of month 15, which is equivalent to 3 i.e. March under modulo 12, and thus covers December to February. The use of the modulo allows the main coordinate to be specified as monotonic, as is generally required. Because the axis is also circular, it would be permissible to rotate the values in order to begin with a different season.

Average seasonal cycle expressed in years: *The time coordinate above could equally well be given in calendar years, thus:*

```
dimensions:
  double time(time);
    time:bounds="bounds_time";
    time:long_name="seasonal phase";
    time:modulo=1.0;
    time:topology="circular";
    time:units="calendar_year as .%f";
```

```
    float bounds_time(2,time);
  data:
    time=0.7917, 1.0417, 1.2917, 1.5417;
    bounds_time= 0.6667, 0.9167, 1.1667, 1.4167,
                 0.9167, 1.1667, 1.4167, 1.6667;
```

Here, the periods have been constructed as exactly quarters of a year, beginning two-thirds of the way through the year. In the 360-day calendar, this is identical to the last example, of periods of three months starting at the beginning of September, but in the standard calendar it is slightly different, since a quarter of a year is not exactly three calendar months.

28 Gregorian calendar

NEW: This standard recommends that Gregorian times be given in units of `days as %Y%m%d.%f` with data type `double` (section 27), unless compatibility is essential with applications that cannot process absolute times. COARDS: In that case, Gregorian times may have units of time formatted as per the recommendations of the Unidata udunits package, which specify a unit and a reference time, i.e. a relative time (section 26). NEW: The recommended unit is `days`, with data type `double`.

COARDS: Intervals between two times in the standard Gregorian calendar can be calculated by the Unidata udunits package. *Udunits implements the mixed Gregorian/Julian calendar system, as followed in England, in which dates prior to 1582-10-15 are assumed to use the Julian calendar. Other software cannot be relied upon to handle the change of calendar in the same way, so for robustness it is recommended that the reference date be later than 1582. If earlier dates must be used, it should be noted that udunits treats 0 AD as identical to 1 AD.*

Data type `double` *gives a precision of about 16 decimal digits, which means that it can resolve tenths of a second for years of up to O(1 million) in relative times. The precision of absolute times is an order of magnitude worse, since a year looks like 10 000 days, rather than 365. The larger the year, the worse the absolute precision. If very large years are needed and the precision is not sufficient, the reference year will have to be modified to keep the interval small enough.*

OGDT: If there is no `calendar` attribute applying to a time variable, the values are assumed to be in the normal Gregorian calendar. This can be made explicit by setting `calendar` to `standard` or `gregorian`.

29 Non-Gregorian calendars

NEW: It is recommended that times in other calendars should be encoded in units of `days as %Y%m%d.%f` with data type `double` (section 27). Relative times are permitted, the recommended `units` being `days since 1-1-1` (midnight on 1 January of year 1), with data type `double`. Since the Unidata udunits package can process only the standard calendar, an extension will be required to process relative times for other calendars.

OGDT: 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 relative times or calculate intervals in the calendar concerned.

30 Multiple time axes and climatological time

OGDT: 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 use of this is to decompose time into multiple partial time dimensions (section 27), of which one or may be collapsed (section 23). NEW: This gives a method of indicating disjoint intervals of time belonging to corresponding parts of the seasonal or diurnal cycles. When a variable has two or three time axes, the first interval of time which they cover is assumed to begin at the earliest boundary values of all the axes. If there is an uncollapsed axis in combination with collapsed axes, it is a "climatological time" axis. *There may be more than one—see below for an example.*

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).

A mean of a corresponding months in a number of years: *A longitude–latitude precipitation field with time axes to indicate the mean over the months of January in 1961 to 1990 inclusive:*

```
dimensions:
  con_year=1;
  year=30;
  month=1;
variables:
  float precipitation(con_year,month,lat,lon);
    precipitation:axis="-TYX";
    precipitation:subgrid="con_year: mean month: mean";
  int con_year(con_year);
    con_year:bounds="bounds_con_year";
    con_year:expand="year";
    con_year:interval=1;
    con_year:long_name="year";
    con_year:units="calendar_year as %Y";
  int bounds_con_year(2,con_year);
  int year(year);
  float month(month);
    month:bounds="bounds_month";
    month:long_name="seasonal phase";
    month:units="calendar_month as %m.%f";
```

```
    float bounds_month(2,month);
  data:
    con_year=1975;
    bounds_con_year=1961, 1990;
    year=1961, 1962, 1963, ..., 1990;
    month=1.5;
    bounds_month=1.0, 2.0;
```

The representative year is not likely to be particularly useful in this case; the important information is the boundaries, which indicate the range of years used to form the climatological mean. These years are also given explicitly, and optionally, for reference.

Climatological seasonal means for several decades: *This is an extension of the previous case, and of the example of an average seasonal cycle in section 27. Here, the axes are set up to indicate climatological means for two of the seasons in three successive decades.*

```
  dimensions:
    decade=3;
    season=2;
  variables:
    float precipitation(decade,season,lat,lon);
      precipitation:axis="-TYX";
      precipitation:subgrid="decade: mean season: mean";
    int decade(decade);
      decade:bounds="bounds_decade";
      decade:interval=1;
      decade:units="calendar_year as %Y";
    int bounds_decade(2,decade);
    int season(season);
      season:bounds="bounds_season";
      season:calendar="standard";
      season:modulo=1200;
      season:units="day as %m%d";
    int bounds_season(2,season);
  data:
    decade=1966, 1976, 1986;
    bounds_decade=1961, 1971, 1981, 1970, 1980, 1990;
    season=115, 415;
    bounds_season=1, 301,
                  228, 531;
```

Here, `precipitation[0][0][*][*]` *is the data for December–February (i.e. 1 December to 28 February inclusive) of the decade 1960–1970 (first December in 1960, last February in 1970), while* `[2][1][*][*]` *is March–May 1981–1990. The choice has been made to give the seasonal phase in months and days, rather than months alone; hence the modulo is 1200 rather than 12. Under modulo 1200, midnight on 1 December can be expressed equivalently as 1 or 1201. If 1201 were specified, it would mean that the first interval of time began on 1 December 1961 (rather than 1960), taking the combination of the lower*

boundaries of both time axes; the value 1 is a year earlier. The drawback of this `%m%d` *scheme is that it is awkward or impossible to give accurate representative dates for the middle of the periods, especially since February has variable length. The absolute time format* `%m.%f` *for seasonal phase is better from this point of view. The season axis is not shown as having circular topology because no information is implied about the other two seasons.*

Average early June maximum temperatures for several years: *In this example, the dimensions indicate that maximum daily temperatures (between 9 a.m. on the day of record and 9 a.m. of the previous day) were recorded for 1–10 June, and an average maximum found for these ten days in each of the years 1980–1984.*

```
dimensions:
  year=5;
  con_season=1;
  con_day=1;
variables:
  float temperature(year,con_season,con_day);
    temperature:axis="T--";
    temperature:subgrid="con_season: mean con_day: maximum";
  int year(year);
    year:long_name="year";
    year:units="calendar_year as %Y";
  int con_season(con_season);
    con_season:bounds="bounds_con_season";
    con_season:interval=1;
    con_season:long_name="seasonal phase";
    con_season:units="day as %m%d";
  int bounds_con_season(2,con_season);
  float con_day(con_day);
    con_day:bounds="bounds_con_day";
    con_day:long_name="diurnal phase";
    con_day:units="hour as %H.%f";
  float bounds_con_day(2,con_day);
data:
  year=1980, 1981, 1982, 1983, 1984;
  con_season=605;
  bounds_con_season=601, 610;
  con_day=-3.0;
  bounds_con_day=-15.0, 9.0;
```

The diurnal phase of -15 h means 15 hours before the beginning of the day in question, i.e. 9 a.m. on the previous day. No bounds are given for the year, because it is a discrete quantity, and there is no further information which could be added. But if the five years were averaged together, this would collapse the year axis, and the extreme years of 1980 and 1984 would be recorded as the boundaries of the collapsed axis. If, say, 1981 were not used in forming the average, the collapsed axis would have attribute `spacing="disjoint"`*.*

Daily values as an average of subdaily values: *Instantaneous pressure measurements are made at intervals of 3 hours (first measurement at midnight) throughout the*

days 6 May to 9 June 1937, and daily means formed from midnight to midnight.

```
dimensions:
  con_subday=1;
  day=35;
variables:
  float pressure(day,con_subday);
    pressure:axis="T-";
    pressure:subgrid="con_subday: mean con_subday: point";
  float con_subday(con_subday);
    con_subday:bounds="bounds_con_subday";
    con_subday:interval=0.125f;
    con_subday:long_name="diurnal phase";
    con_subday:spacing="uniform";
    con_subday:units="days as .%f";
  float bounds_con_subday(2,con_subday);
  int day(day);
    day:long_name="year and seasonal phase";
    day:units="days as %Y%m%d";
data:
  con_subday=0.5;
  bounds_con_subday=0.0, 0.875;
  day=19370506, 19370507, ..., 19370608, 19370609;
```

Note that the `con_subday` *axis is shown with two subgrid methods, referring to subgrid variation before and after its collapse. The only point here in having separate axes for day and diurnal phase is to show when the first and last instantaneous measurements were made in each day. If this is not important to record, the two axes could be merged together thus:*

```
dimensions:
  day=35;
variables:
  float pressure(day);
    pressure:subgrid="day: mean day: point";
  float day(day);
    con_subday:bounds="bounds_day";
    con_subday:interval=0.125f;
    con_subday:long_name="time";
    con_subday:spacing="uniform";
    con_subday:units="days as %Y%m%d.%f";
  float bounds_day(2,day);
data:
  day=19370506.5, 19370507.5, ..., 19370608.5, 19370609.5;
  bounds_day=19370506.0, 19370507.0, ..., 19370608.0, 19370609.0,
             19370507.0, 19370508.0, ..., 19370609.0, 19370610.0;
```

If the 35 days were then averaged together, the date axis would collapse with bounds of 19370506.0 and 19370610.0. The `subgrid` *attribute would not need modification since it is already shown as a mean over the* `day` *axis.*

Average diurnal cycle: *The following axes are appropriate for the average diurnal cycle of precipitation rate in July 1970–1979 as a function of latitude:*

```
dimensions:
  con_year=1;
  con_month=1;
  hour=8;
  lat=45;
  con_lon=1;
variables:
  float ppnrate(con_year,con_month,hour,lat,con_lon);
    ppnrate:axis="--TYX";
    ppnrate:subgrid="con_year: mean con_month: mean "
      "con_lon: mean";
    ppnrate:units="kg m-2 s-1";
  int con_year(con_year);
    con_year:bounds="bounds_con_year";
    con_year:interval=1;
    con_year:units="calendar_year as %Y";
  int bounds_con_year(2,con_year);
  float con_month(con_month);
    con_month:bounds="bounds_con_month";
    con_month:units="calendar_month as %m.%f";
  float bounds_con_month(2,con_month);
  float hour(hour);
    hour:bounds="bounds_hour";
    hour:modulo=1.0f;
    hour:topology="circular";
    hour:units="hour as %H.%f";
  float bounds_hour(2,bounds_hour);
data:
  con_year=1975;
  bounds_con_year=1970, 1979;
  con_month=7.5;
  bounds_con_month=7.0, 8.0;
  hour=1.5, 4.5, 7.5, 10.5, 13.5, 16.5, 19.5, 22.5;
  bounds_hour=0.0, 3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0,
              3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0;
```

31 Invalid values in a data variable

OGDT: 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). Invalid values are not permitted in a coordinate variable, but the attributes which define the valid range may be used in boundary variables (section 21) to indicate unbounded cells.

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. OGDT: 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.

OGDT: 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

OGDT: 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

OGDT: 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 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.

OGDT: 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, any ancillary and boundary coordinate variables, and the compressed data variables with all the attributes of the uncompressed variables are written to the archived netCDF file. The uncompressed data variables can be reconstituted exactly as they were using this information, except that their original variable names are not known.

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:
  long landpoint(landpoint);
    landpoint:compress="lat lon";
  float landsoilt(depth,landpoint);
    landsoilt:axis="Z-";
    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]`*.*

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(oceanpoint);
    salinity:axis="-";
  long oceanpoint(oceanpoint);
    oceanpoint:compress="depth lat lon";
  float depth(depth);
  float lat(lat);
  float lon(lon);
```

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

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. OGDT: 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 variable (containing the packed data) then in files adhering to this standard the 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	5	Version number of these appendices
associate	19 20	Identifies variables containing alternative sets of coordinates
axis	9 16 19	Identifies special spatiotemporal dimensions
bounds	21 23 30	Identifies a variable containing boundary values
calendar	5 24 28 29	Calendar used for encoding time axes
comment	5	Additional information about the file
component	18	Identifies variables containing components of a variable
compress	33	Records the dimensions which have been compressed by gathering
Conventions	5	Identifies the netCDF standard
coordinates	19	Synonym for associate
expand	23 30	Records coordinates of an axis before contraction
_FillValue	31	Indicator of invalid data
FORTRAN_format	12	Format for printing the values of a variable
history	5	Evolution of the data in the file
institution	5	Who made or supplied the data
interval	23 30	The typical separation between points on an axis before contraction
long_name	12 14–16 25	Long description of a physical quantity
modulo	12 14 27	Arithmetic modulo of a variable
north_pole	10	Geographical location of rotated North Pole
positive	16	Direction of positive for a vertical axis
production	5	How the data was produced
scale_factor	31 34	Multiplicative factor for packing data
spacing	23 30	Indicates the spacing of points along an axis before contraction
subgrid	22 23 30	Records how the data values represent subgrid variation
topology	13 14 27	Topology of an axis (circular or not)
time_format	24	Format for printing a time and date
units	12 14 15 25–29	Units of a physical quantity
valid_max	31 21	Largest valid value of a variable
valid_min	31 21	Smallest valid value of a variable
valid_range	31	Smallest and largest valid values of a variable

B Methods of representing subgrid variation

Method	Units	Description
`cell`	u	Value is a property of the whole cell (e.g. an integral)
`maximum`	u	Maximum
`median`	u	Median
`mid-range`	u	Average of maximum and minimum
`minimum`	u	Minimum
`mean`	u	Mean (average)
`mode`	u	Mode (most common)
`point`	u	Value applies at gridpoint
`standard deviation`	u	Standard deviation
`variance`	u^2	Variance

Units: u means the units of the quantity whose subgrid variation is represented by this method.

C Modifications to `udunits.dat`

NEW: The unit `unity` is defined as a dimensionless constant equal to one.

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.

NEW: The units `calendar_month` and `calendar_year` are units of time, but cannot be converted into each other or any other units of time, except that multiples of 12 calendar months equal integral numbers of calendar years. The units `year` and `month` are not allowed, because they can cause confusion.

D Long names for 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	long_name	units
1.0	depth below the surface	m
1.0	height above the surface	m
1.0	latitude	degree_north
1.0	longitude	degree_east
1.0	pressure	Pa
1.0	soil temperature	K
1.0	specific humidity	unity
1.0	temperature	K
1.0	time	s

Version: The version of the appendices at which this quantity was introduced.
long_name: Case, spaces and punctuation are not significant in the long_name.