NetCDF Climate and Forecast Aggregation (CFA) Conventions

Release 0.4

David Hassell and Jonathan Gregory

February 27, 2014

CONTENTS

CHAPTER ONE

AUTHORS

David Hassell and Jonathan Gregory

With thanks to Phil Bentley, John Caron, Stephen Pascoe and Ag Stephens.

CHAPTER

TWO

ABSTRACT

The proposed CF aggregation rules describe how multiple CF fields may be combined into one, larger field. These rules are fully general and are based purely on CF metadata. To make full use of such an algorithm, it is necessary that such an aggregated field may be stored within software memory (for field creation and manipulation) and on disk (for long term storage).

This document proposes the abstract **CFA framework** for the storage of CF-aggregated CF fields, and the **CFA-netCDF conventions** for their efficient storage in netCDF files.

As with other similar file storage conventions, on-disk storage is efficient because, wherever possible, the component fields from which an aggregated field is composed are stored as references to the files containing the original datasets.

Both the abstract framework and the ability to read and write NCA files are implemented in the cf-python software library.

CHAPTER

THREE

INTRODUCTION

The proposed CF aggregation rules offer a fully general methodology for combining two or more CF fields which logically form one, larger dataset.

To fully accommodate such fields, any in-memory or on-disk storage must allow for:

- Simultaneous aggregation across more than one dimension.
- Aggregation accounting for different but equivalent:
  - orders of dimensions
  - numbers of size 1 dimensions
  - senses (directions) in which dimensions run
  - units of the data
  - missing data values

A framework for in-memory storage must also allow for the aggregation of previously aggregated fields, since that is required for multidimensional aggregations. The framework presented here satisfies these requirements and is abstract in that it is "language neutral", i.e. it does not rely on structures particular to any one computer programming language, and should therefore be implementable in any sufficiently rich programming environment.

Having created such an abstract framework, it will also serve as the model for storing aggregated fields on disk. To be useful, a file encoding must follow a documented and standardized convention, since such files could form part of a maintained archive. The convention proposed here, the CFA-netCDF convention, proposes encodings for netCDF file storage. The key element of a CFA-netCDF file is that it is a CF-like netCDF file, which only differs from a normal CF-netCDF file in that it requires extra processing to realise any aggregated data arrays. In all other aspects, the metadata in a CFA-netCDF file is identical to CF metadata.

In addition to storing aggregated fields, the abstract framework and file storage convention proposed here also allows for:

- Storage of (not necessarily contiguous) subspaces of aggregated fields *without duplicating* any part of the original datasets which span the subspace.
- Storage of changes to (not necessarily contiguous) parts of aggregated fields.
- Aggregation and storage of datasets originating in a mixture of formats (in-memory, netCDF, PP, *etc.*), provided that they may be cast as CF fields.
- In-memory manipulation and on-disk storage of larger-than-memory fields.

It is important that the ideas presented are demonstrably viable from a software engineering point of view, so the cf-python software library has been modified to implement the full CF aggregation algorithm, the ability to read and write CFA-netCDF files and the additional benefits listed above.

CHAPTER

FOUR

THE CFA FRAMEWORK

4.1 A conceptual framework for the in-memory storage of aggregated arrays

A **master data array** is one which is constructed from the one or more **sub-arrays**.

The simplest master data array has just one sub-array which is equal to itself, but in general, a master data array may have up to as many, non-overlapping sub-arrays as is it has elements and each sub-array contains a contiguous subset of the master data array values. See *figure 1a* and *figure 2a* for examples of master data arrays and their constituent sub-arrays.

A master data array constructed from sub-arrays arises naturally from an aggregation process, such as by the CF aggregation rules.

Sub-arrays may be be combined to create hyperrectangular[1] master data arrays, which poses storage and access difficulties since there may be different numbers of sub-arrays across different sections of the same master data array dimension. For example, the master data array in *figure 2a* has three sub-arrays spanning the first two rows (values 0 to 6 and 7 to 13) but only two sub-arrays spanning third row (values 14 to 20). Such an irregular construction would, for example, complicate finding which sub-array contained a particular element or, more importantly, complicate the recombination of sub-arrays into an equivalent master data array (an operation required by the CF aggregation process).

A solution is to decompose the master data array into a hyperrectangular **partition matrix** such that its elements (called **partitions**) span all or part of exactly one sub-array. A given **partition's data array** is then a reference to a unique part of a unique sub-array. See *figure 1b*, *figure 1c* and *figure 2b* for examples.

The master data array may be therefore be considered as being constructed from a hyperrectangular matrix of partitions. See *example 1* and *example 2*.

A guaranteed hyperrectangular decomposition renders easy any operations requiring knowledge of the partition locations within the master data array (such as element location or partition matrix transposition) and allows the master data array to be aggregated with other master data arrays.

4.2 Larger-than-memory arrays

Sub-arrays larger than the available memory may be stored effectively in memory by simply ensuring that such a sub-array is kept in a file and partitioning the master data array so that each partition is small enough to be realized in memory, if required.

[1] The generalization of a rectangle for higher dimensions.

4.3 Subspaces

A subspace of a master data array may be created easily by discarding partitions which do not overlap the subspace and for each remaining partition, adjusting the definition of the part of the sub-array which comprises the partition's data array (see the *part* parameter).

4.4 Partition matrix properties

- The partition matrix's dimensions are always a subset of the master data array's dimensions. I.e. each dimension of partition matrix index space corresponds to a dimension of the master data array.
- The partition matrix may be a scalar, if it contains exactly one partition whose data array spans the entire master data array.
- The partition matrix may span fewer dimensions than the master data array (see *figure 1b*, in which the partition matrix is 1-dimensional but the master data array is 2-dimensional), but never more.
- The order of the partition matrix's dimensions may be different to the relative ordering of the equivalent dimensions of the master data array.
- The creation of a subspace of the master data array may result in one or more partitions which reference a non-contiguous part of a sub-array (see the *part* parameter).

4.5 Examples

4.5.1 Example 1

The 2-dimensional 2 x 7 master data array in **figure 1a** is composed from 3 sub-arrays. The most efficient way of partitioning the master data array into a hyperrectangular partition matrix such that each partition contains all or part of exactly one sub-array is shown in **figure 1b**. Note that, in this case:

- The partitions' data arrays are not all of the same size or shape
- Each partition's data array spans an entire sub-array
- The partition matrix has fewer dimensions than the master data array.

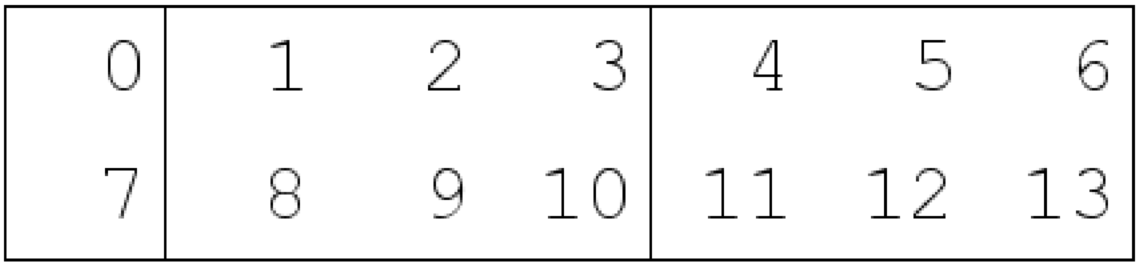

Fig. 4.1: **Figure 1a**. The 3 sub-arrays of the master data array.

Another, equally valid partitioning of the master data array is shown in **figure 1c**. Note that, in this case:

- No partitions' data arrays span an entire sub-array.
- The partition matrix has the same number of dimensions as the master data array.

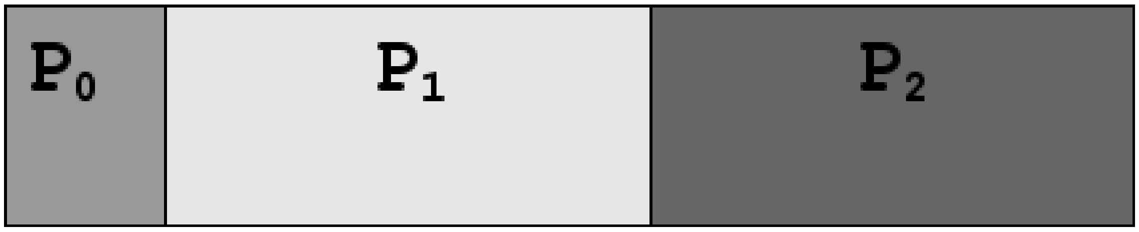

Fig. 4.2: **Figure 1b**. A 1-dimensional, 3 element partition matrix of the master data array. Each block of colour represents one of the 3 sub-arrays and each partition of the partition matrix is labelled P_x. For example, the partition data array of P_0 contains values 0 and 7.

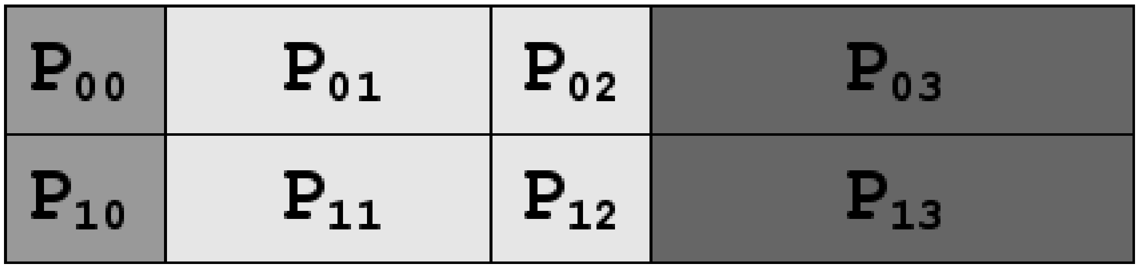

Fig. 4.3: **Figure 1c**. A 2-dimensional, 2 x 4 element partition matrix of the master data array. Each block of colour represents one of the 3 sub-arrays and each partition of the partition matrix is is labelled P_{yx}. For example, the partition data array of P_{11} contains values 8 and 9 and the partition data array of P_{12} contains value 10.

4.5.2 Example 2

The 2-dimensional 8 x 7 master data array in **figure 2a** is composed from 10 sub-arrays. The most efficient way of partitioning the master data array into a hyperrectangular partition matrix such that each partition contains all or part of exactly one sub-array is shown in **figure 2b**. Note that, in this case:

- The partitions' data arrays are not all of the same size or shape
- Some partitions' data arrays span an entire sub-array (P_{00} and P_{33}), but the rest do not.

4.6 Accounting for arbitrary partition data array properties

There are properties of a partition's data array which are arbitrary in the sense that, whilst these properties may differ to their equivalents in the master data array, the partition's data array may always be altered to conform with the master data array with no loss of information.

A partition's data array inherits these properties, unchanged, from the sub-array which contains it.

The properties for which a partition's data array may differ from its master data array are:

- The order of dimensions.
- The number of size 1 dimensions.
- The sense in which dimensions run.
- The units of the data values.
- The missing data value.

When a partition's data array is required by the master data array, it needs to be **conformed** by doing any or none of:

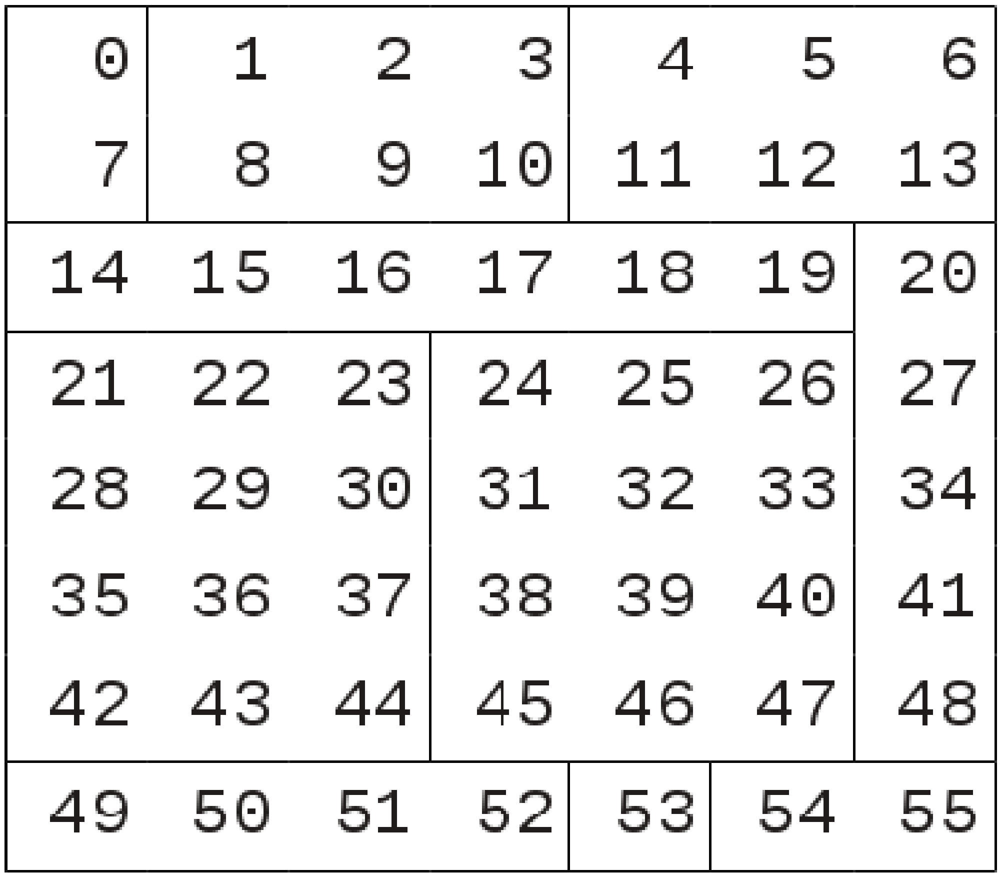

Fig. 4.4: **Figure 2a**. The 10 sub-arrays of the master data array.

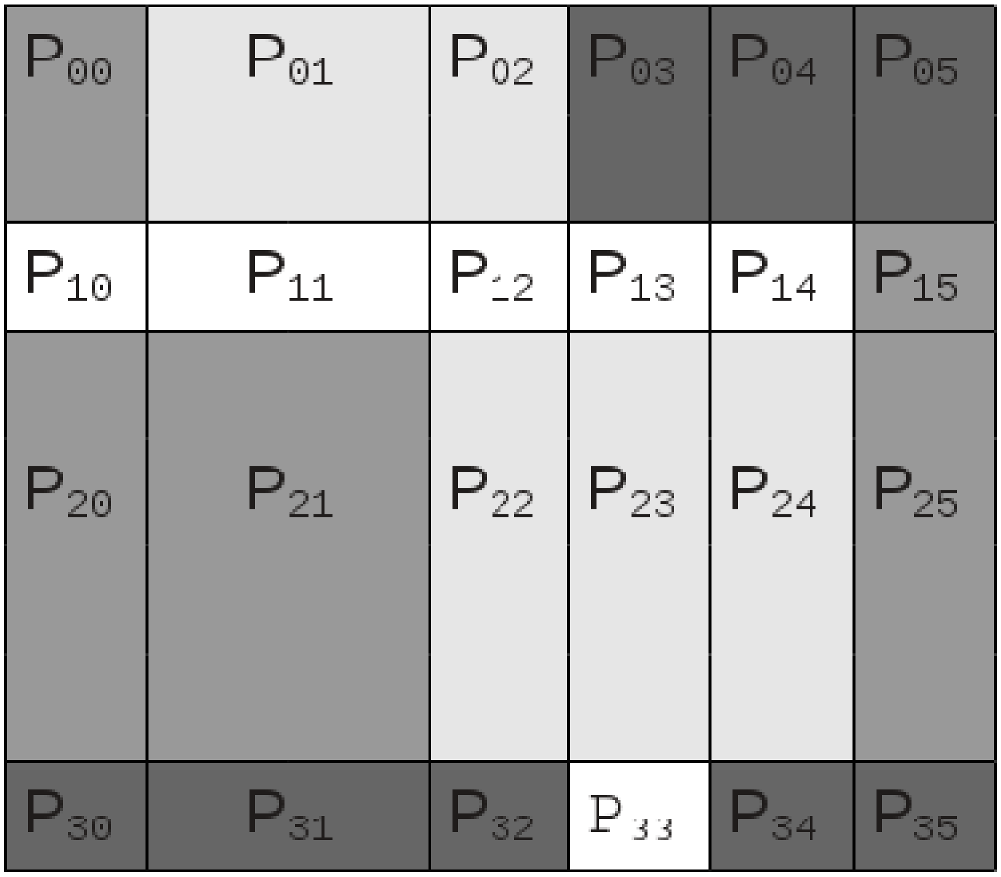

Fig. 4.5: **Figure 2b**. A 2-dimensional, 4 x 6 element partition matrix of the master data array. Each block of colour represents one of the 10 sub-arrays and each partition of the partition matrix is labelled P_{yx}. For example, the partition data array of P_{30} contains value 49; the partition data array of P_{31} contains values 50 and 51; and the partition data array of P_{32} contains value 52.

- Reordering its dimensions to the same order as the master data array.
- Removing size 1 dimensions which don't exist in the master data array.
- Adding size 1 dimensions which exist in the master data array but not in the partition's data array.
- Reversing dimensions which run in the opposite direction to the master data array.
- Converting the data values to have the same units as the master array.
- *Either* the missing data value is converted to that of the master array (accounting for conflicts with non-missing data values) *or*, if arrays are stored with ancillary missing data masks (as can be the case with python numpy arrays, for example), the partition's data array's missing data value may be ignored.

4.7 Parameters required for specifying a master data array

It follows that a master data array and its partitions may be completely specified by a small number of parameters.

Note: When a partition has a parameter value equal to the master array then there is some redundancy which will be exploited when storing the array by its parameters with the *CFA-netCDF conventions*.

4.7.1 Master data array parameters

The master data array comprises:

dtype The data type of the master data array.

units The units of the master data array.

calendar (*if required by units*) The calendar of the master data array.

dimensions An ordered list of the master data array's dimensions.

shape An ordered list of the master data array's dimension sizes. The sizes correspond to the *dimensions* list.

Partitions A matrix of the master data array's partitions. Each partition is described by its *partition parameters*.

4.7.2 Partition matrix parameters

pmdimensions An ordered list of the dimensions along which the master data array is partitioned. Each of these dimensions is one those specified by the *dimensions* parameter.

pmshape An ordered list containing the number of partitions along each partitioned dimension of the master data array. The sizes correspond to the *pmdimensions* list.

4.7.3 Partition parameters

Each partition of the partition matrix comprises:

pdimensions An ordered list of the partition's data array dimensions.

punits The units of the partition's data array.

pcalendar (*if required by punits*) The calendar of the partition's data array.

flip A collection of the partition's data array dimensions which run in the run in the opposite direction to those of the master data array.

location An ordered list of ranges of indices, one for each dimension of the *master data array*, which describe the contiguous section of the master data array spanned by this partition. The ranges correspond to the *dimensions* list.

part An ordered list of indices for each dimension of the partition's data array which describe the part of the sub-array which is spanned by this partition. The indices correspond to the *pdimensions* list.

subarray A reference to the sub-array which contains the partition's data array.

CHAPTER

FIVE

THE CFA-NETCDF CONVENTIONS

The CFA-netCDF conventions describe the efficient netCDF file storage of CF fields with master data arrays which have been aggregated within the *CFA framework*.

5.1 Storing the master data array parameters in a string

Since the *parameters* which completely describe a master data array may each be cast as one of a set of particular basic types, these parameters may be easily encoded in a JSON (JavaScript Object Notation) string for simple inclusion in a netCDF file.

JSON is a lightweight data-interchange format which is easy for humans to read and write and easy for machines to parse and generate. There are JSON encoders and decoders for every reasonable language. See the JSON Wikipedia article for some good examples of JSON strings.

The basic types recognised by JSON are (with JSON encoded examples):

- string (double-quoted Unicode, with backslash escaping: `"foo"`, `""`)
- number (`3.14159`, `0`)
- boolean (true or false: `true`, `false`)
- null (empty: `null`)
- Array (an ordered, comma-separated sequence of values enclosed in square brackets; the values do not need to be of the same type: `[1, "a", [2, null, {"x": false}]]`)
- Object (an unordered, comma-separated collection of key:value pairs enclosed in curly braces, with the ':' character separating the key and the value; the keys must be strings and should be distinct from each other: `{"x": [4, [true, null]], "y": 2.71828}`)

It clear that, with the exception of the partition's *subarray* parameter, each of the values of the parameters which describe a master data array is one or a combination of these basic types. For example, the *pmshape* parameter is a list of numbers.

The same is actually true for the partition's *subarray* parameter, although it is not so immediately obvious. The issue is that it may be a reference to section of a file or to a multidimensional array in memory. However, in the latter case the sub-array must be written to a file in order to be stored, so this reduces to the former case. A file reference is easily encapsulated by a collection of the basic types. For example, if the reference is to a netCDF file variable then all that is required are the file name (a string), the name of the netCDF variable containing the sub-array (a string) and its shape (a list of numbers)[1].

Thus, for storage purposes, all of the master data array parameters may be encoded in a single JSON string.

[1] The shape is required since the sub-array may omit (contain) size one dimensions which are (are not) present in the master data array. Also, a multi-character string array in memory may be different to the shape of the array stored in a netCDF file, which may be stored as a character array with an extra trailing dimension.

See the complete description of *CFA-netCDF attributes* for details on how each master data array parameter is encoded.

5.2 Creating a netCDF variable for the master data array

A multidimensional master data array may be stored in a **scalar netCDF variable** with the same **data type** as its master array, whose attributes include the JSON encoded strings of the master data array parameters. When read, this scalar array variable may then be converted to a multidimensional array variable after the parameters have been decoded. Note that the datum of this scalar netCDF variable is ignored and need not be set. Such a variable is called a **CFA variable** and a file storing CFA variables is called a **CFA-netCDF file** and should include both 'CF' and 'CFA' in its global `conventions` attribute. A CFA-netCDF file may contain a mixture of CFA variables and normal netCDF variables, or even no CFA variables at all. In the latter case, the CFA-netCDF file is also a CF-netCDF file.

For partitions whose *subarray* parameter refers to a sub-array in memory, that sub-array is written to the CFA-netCDF file itself as a **CFA private variable**. This is a normal netCDF variable marked as containing the sub-array for a partition of one or more CFA variables and should not be interpreted otherwise according to the CF conventions. As this sub-array now exists in a netCDF file, the relevant CFA variables may refer to this sub-array as for any netCDF variable, i.e. by specifying the netCDF file name[2], the netCDF variable name and its shape. See *example 4*.

5.3 Advantages of netCDF

File storage of an aggregated field stored does not need to use netCDF format (for example, plain text XML or CDL format files would be sufficient) but there some reasons why it is desirable:

- Nearly all of the discovery metadata in a CFA-netCDF file is easily retrievable with standard, unmodified netCDF libraries. The only exception is the ordered list netCDF dimension names of a CFA variable, and to ameliorate any difficulties associated with this, these are stored in their own netCDF property for easy access (see the *cfa_dimensions* property).
- A plain text CDL representation of a CFA-netCDF file is trivial to produce with the ncdump utility.
- An important part of the CFA conventions is the ability to mix, within the same CFA-netCDF file, normal CF-netCDF variables (possibly with very large data arrays) with CFA variables (which may contain references to CFA private variables, possibly with very large data arrays, in the same file), so a binary format is desirable to keep the file size to a minimum.
- A CFA variable may contain any CF property such as cell methods, flags, ancillary variables, coordinates, formula_terms, *etc*. By using a netCDF file, there is no need to invent new encodings for complex field properties which have already been standardized for netCDF variables by the CF conventions.

5.4 Recommended usage

CFA-netCDF files should have the file name extension ".nca".

It is recommended, though not necessary to write the following types of variable as normal netCDF variables:

- One dimensional coordinates and their bounds (to facilitate discovery).
- Master data arrays with only one partition whose data array would otherwise be written to a CFA-netCDF file as a CFA private variable (to avoid unnecessary obfuscation).

[2] In this case, though, the file name may be omitted, in which case the name of the CFA-netCDF file is assumed. See the *file* attribute.

5.5 Examples

The following example files are described by their CDL representation.

5.5.1 Example 3

A simple CFA-netCDF file:

```
netcdf temperature.nca {
dimensions:
  time = 48 ;
  lat = 64 ;
  lon = 128 ;
variables:
  double time(time) ;
    time:long_name = "time" ;
    time:units = "days since 0000-1-1" ;
  double lat(lat) ;
    lat:units = "degrees_north" ;
    lat:standard_name = "latitude" ;
  double lon(lon) ;
    lon:units = "degrees_east" ;
    lon:standard_name = "longitude" ;
  float tas ;
    tas:standard_name = "air_temperature" ;
    tas:units = "K" ;
    tas:cf_role = "cfa_variable" ;
    tas:cfa_dimensions = "time lat lon" ;
    tas:cfa_array = '{"pmshape": [2],
                      "pmdimensions": ["time"],
                      "Partitions": [{"index": [0],
                                      "data": {"file": "/home/david/test1.nc",
                                               "shape": [12, 64, 128],
                                               "ncvar": "tas"
                                              },
                                      "location": [[0, 12], [0, 64], [0, 128]],
                                      "format": "netCDF"
                                     },
                                     {"index": [1],
                                      "data": {"file": "/home/david/test2.nc",
                                               "shape": [36, 64, 128],
                                               "ncvar": "tas2"
                                              },
                                      "location": [[12, 48], [0, 64], [0, 128]],
                                      "format": "netCDF"
                                     }
                                    ]
                     }' ;

// global attributes:
    :Conventions = "CF-1.5 CFA" ;

data:

 time = 164569, 164599.5, 164630.5, 164660, 164689.5, 164720, 164750.5,
       // etcetera.
```

```
lat = -87.8638000488281, -85.0965270996094, -82.3129119873047,
      // etcetera.

lon = 0, 2.8125, 5.625, 8.4375, 11.25, 14.0625, 16.875, 19.6875, 22.5,
      // etcetera.
```

Points to note:

- The file specifies two conventions.
- The file contains one CFA variable (`tas`) and three normal variables (`time`, `lat` and `lon`).
- The CFA variable stores the master data array's dimensions in a separate attribute (`cfa_dimensions`) to facilitate reconstruction of a multidimensional variable without having to decode the `cfa_array` string.
- The `cfa_array` string has been split over many lines for enhanced readability. New lines are permitted in JSON strings.
- The CFA variable defines its data type and units in the normal manner, so that these parameters of the master array may be omitted from the `cfa_array` attribute.
- The CFA variable may have any CF-netCDF attributes, with no restrictions.
- Partition parameters which are the same as their master array may be omitted.

5.5.2 Example 4

Storing a master data array with an in-memory partition data array:

```
netcdf temperature2.nca {
dimensions:
  time = 48 ;
  lat = 64 ;
  lon = 128 ;
  cfa12 = 12 ;
  cfa64 = 64 ;
  cfa128 = 128 ;
variables:
  double time(time) ;
    time:long_name = "time" ;
    time:units = "days since 0000-1-1" ;
  double lat(lat) ;
    lat:units = "degrees_north" ;
    lat:standard_name = "latitude" ;
  double lon(lon) ;
    lon:units = "degrees_east" ;
    lon:standard_name = "longitude" ;
  float tas ;
    tas:standard_name = "air_temperature" ;
    tas:units = "K" ;
    tas:cf_role = "cfa_variable" ;
    tas:cfa_dimensions = "time lat lon" ;
    tas:cfa_array = '{"pmshape": [2],
                      "pmdimensions": ["time"],
                      "Partitions": [{"index": [0],
                                      "punits" : "K @ 273.15",
                                      "pdimensions": ["lon", "time", lat"],
                                      "flip": ["time"],
                                      "data": {"shape": [128, 12, 64],
                                               "ncvar": "cfa_45sdf83745"
                                              },
                                      "location": [[0, 12], [0, 64], [0, 128]],
                                      "format": "netCDF"
                                     },
                                     {"index": [1],
                                      "data": {"file": "/home/david/test2.nc",
                                               "shape": [36, 64, 128],
                                               "ncvar": "tas2"
                                              },
                                      "location": [[12, 48], [0, 64], [0, 128]],
                                      "format": "netCDF"
                                     }
                                    ]
                     }' ;
  float cfa_45sdf83745(cfa128, cfa12, cfa64) ;
    cfa_45sdf83745:cf_role = "cfa_private" ;


// global attributes:
    :Conventions = "CF-1.5 CFA" ;

data:

 time = 164569, 164599.5, 164630.5, 164660, 164689.5, 164720, 164750.5,
       // etcetera.

 lat = -87.8638000488281, -85.0965270996094, -82.3129119873047,
       // etcetera.

 lon = 0, 2.8125, 5.625, 8.4375, 11.25, 14.0625, 16.875, 19.6875, 22.5,
       // etcetera.

 cfa_45sdf83745 = -4.5, 3.5, 23.6, -4.45, 13.5, 13.6,
       // etcetera.
```

Points to note:

- The in-memory partition data array has been written to the file with an automatically generated variable name (`cfa_45sdf83745`), which has an attribute `cfa_private` to mark it as a private variable according to the CFA convention.
- The in-memory array had different units and dimension order relative to the master array.
- The time dimension of the in-memory array runs in the opposite direction to the time dimension of the master data array, but the other dimensions run in the same sense as the master array.
- The CFA private variable has dimensions which are only used by it and other CFA private variables.

CHAPTER

SIX

CFA-NETCDF PROPERTIES

netCDF properties	cfa_array attributes	Partitions attributes	subarray attributes
cf_role, *cfa_dimensions*, *cfa_array*	*Partitions*, *pmdimensions*, *pmdimensions*, *pmshape*, *base*,	*index*, *location*, *pdimensions*, *reverse*, *subarray*, *punits*, *pcalendar*, *part*,	*shape*, *file*, *dtype*, *ncvar*, *varid*, *file_offset*, *format*

6.1 cf_role

`cf_role` This standard CF property *must* be used to mark the netCDF variable as either an *CFA variable* or a *CFA private variable*.

It takes values of `"cfa_variable"` or `"cfa_private"` for a CFA variable or CFA private variable respectively.

Examples
`"cfa_variable"`
`"cfa_private"`

6.2 cfa_dimensions

`cfa_dimensions` A netCDF property whose value is a string containing the ordered, space delimited set of the master array's dimension names. The specified dimensions are all defined as netCDF dimensions in the CFA-netCDF file.

If missing, an empty sting or a string containing only spaces then the master array is assumed to be scalar.

Examples
`"time lat lon"`
`""`
`" "`

The master array's dimensions are stored outside of the *cfa_array* property in order to provide useful information about the master array without having to decode the `cfa_array` string (which may not be possible in all APIs).

6.3 cfa_array

`cfa_array` A netCDF property whose value is a JSON encoded associative array containing the attributes required for constructing the aggregated data array.

The **dimensions**, **shape**, **dtype**, **units** and **calendar** *master data array parameters* are specified elsewhere (by properties of the netCDF CFA variable or inferrable from the netCDF dimensions in CFA-netCDF file) and so are not required.

The encoded string may be relatively succinct, as many attributes have well defined default values.

Examples
`'{"Partitions":[{"index": [0], "subarray": {"file": "test1.nc", "shape": [64, 128], "ncvar": "tas"}, "location": [[0, 64], [0, 128]]}]}'`

The JSON *decoded* attributes are described here:

pmdimensions (*optional*) An ordered list of strings containing the names of the dimensions of the partition martrix. Each of these dimensions is one those specified by the *cfa_dimensions* property (and is therefore defined as a netCDF dimension in the CFA-netCDF file).

The value `[]` means that the partition matrix is a scalar, i.e. the master array comprises a single partition.

If missing then it is assumed that the partition matrix is a scalar.

Examples
`["lat", "time"]`
`["height"]`
`[]`

pmshape (*optional*) An ordered list of integers containing shape of the partition matrix. The integers sizes correspond to the dimensions of the partition matrix given by *pmdimensions*.

The value `[]` means that the partition matrix is a scalar, i.e. the master array comprises a single partition.

If missing then it is assumed that each dimension of the partition matrix has size `1`. For example, if *pmdimensions* is `[]` then the default shape is `[]` and if *pmdimensions* is `["time", "height"]` then the default shape is `[1, 1]`.

Examples
`[2, 3]`
`[87]`
`[1, 1]`
`[1]`
`[]`

base (*optional*) A string giving the base for relative file names given by *file*. May be an absolute or relative URL or location on the local system. If it is a relative location then it assumed to be relative to the local directory or URL base containing the CFA-netCDF file.

If set then all file names are assumed to have relative paths. If it is an empty string (`""`) then the base is taken as the local directory or URL base containing the CFA-netCDF file.

If missing then it is assumed that all file names are absolute paths (local files or URLs).

Examples
`"/data/archive"`
`"../archive/"`
`"http://archive/files"`
`""`

Partitions A list whose elements define each of the master array's partitions. The order of the list is arbitrary since each element contains its (possibly multidimensional) index in the partition matrix.

Examples
`[{"index": [0], "subarray": {"shape": [2, 3], "ncvar": "cfa_1bE8EBC2c3"}, "location": [[0, 1], [0, 2]]}]`

Each element of the list is an associative array which specifies a partition with the *following attributes*:

6.3.1 Partition attributes

index An ordered list of indices specifying the position of the partition in the partition matrix. The indices correspond to the *pmdimensions* list.

An index of `[]` means that the partition matrix is a scalar, i.e. the master array comprises a single partition.

If missing then it is assumed that the master array comprises a single partition.

Examples
`[0]`
`[2, 1]`
`[]`

Note: Indices count from zero.

location (*optional*) An ordered list of ranges of indices, one for each dimension of the master data array, which describe the contiguous section of the master data array spanned by the partition's data array. The ranges correspond to the *cfa_dimensions* list.

Each range is a two-element list giving a *start* and *stop* index for its dimension. For example, the range `[3, 5]` is equivalent to indices `3`, `4` and `5`; and the range `[6, 6]` is equivalent to index `6`.

If the master data array is a scalar then it may be an empty list.

If missing then it is assumed that the partition matrix contains exactly one partition whose data array spans the entire master array.

Examples
`[[2, 2], [3, 5], [2, 56]]`
`[[0, 0]]`
`[]`

Note: Indices count from zero.

pdimensions (*optional*) An ordered list of the partition's sub-array dimension names. The specified dimensions are all defined as netCDF dimensions in the CFA-netCDF file.

If there are any size 1 dimensions of the partition which are not spanned by the master array then the partition's dimensions *must* be specified.

If the partition's data array is a scalar then it may be an empty list.

If missing then it is assumed to be equal to dimensions of the master array.

Examples
`["lon", "time", "lat"]`
`[]`

Note: If a partition's data array dimensions are not specified and the sub-array is stored in another file then it is required *only* that the sub-array has the same number of dimensions, with the same physical meaning and in the same order as the master array. For example, if the sub-array were in a different netCDF file, its dimensions may have different names and sizes relative to the equivalent dimensions in the CFA-netCDF file.

reverse (*optional*) An arbitrarily ordered list of zero or more of the partition's data array dimension names. The specified dimensions are all defined as netCDF dimensions in the CFA-netCDF file.

Each specified dimension of the partition's sub-array is assumed to run in the opposite direction to the master data array and so will need to be reversed prior to use within the master data array.

If missing, or an empty list, then it is assumed that no partition sub-array dimensions need to be reversed.

Examples
`["time"]`
`["lon", "time", "lat"]`
`[]`

punits (*optional*) A string containing the units of the partition's data array.

If missing then it is assumed to be equal to units of the master array.

Examples
`"m s-1"`
`"days since 2000-12-1"`
`"1"`
`""`

pcalendar (*optional*) A string containing the calendar of the partition's units.

If missing then it is assumed to be equal to calendar of the master array.

Examples
`"noleap"`

part (*optional*) A string defining the part of the partition's sub-array which comprises the partition's data array.

For each of the partition's dimensions, the string describes the indices of the sub-array which define the partition's data array. The indices correspond to to the *pdimensions* list.

Indices are contained within round or square brackets. Round brackets specify a sequence of indices along that dimension. Square brackets provide a succinct method of describing a strictly monotonic and regularly spaced sequence of indices for the dimension via *start*, *stop* and *step* values. For example, `[0, 3, 1]` is equivalent to `(0, 1, 2, 3)` and `[10, 4, -2]` is equivalent to `(10, 8, 6, 4)`.

If missing, or the string `"[]"`, then it is assumed that the partition's data array spans the entire sub-array.

Examples
`"[[2, 5, 1], (1, 3, 4, 7), [0, 11, 2]]"`
`"[(4, 2, 1), [5, 1, -4], [0, 0, 1]]"`
`"[]"`

Note: Indices count from zero.

subarray An associative array giving the attributes required to define the sub-array containing the partition's data array. Only a subset of these will be required, depending on the storage format of the sub-array.

Examples
`{"shape": [4, 7, 3], "ncvar": "cfa_345eA9001D", "varid": 10}`
`{"file": "temp.nc", "ncvar": "tas", "shape": [1200], "format": "netCDF"}`
`{"dtype": "float", "file": "temp.nc", "shape": [1200], "varid": 6}`
`{"shape": [73, 96], "file": "../data.pp", "file_offset": 7348, "format": "PP"}`
`{"shape": [73, 96], "file": "data.pp", "file_offset": 0, "format": "PP"}`

The keys specify a sub-array with the *following attributes*:

6.3.2 Sub-array attributes

format (*optional*) A string naming the format of the file containing the partition's sub-array.

If missing then the format is assumed to the same as the CFA-netCDF file.

Examples
`"netCDF"`
`"PP"`

shape An ordered list of the partition's sub-array dimension sizes. The sizes correspond to to the *pdimensions* list.

Examples
`[4, 7, 3]`
`[1, 1]`
`[]`

file (*optional*) A string naming the file which contains the partition's sub-array. May be a local file or a URL.

If *base* has been set then the file name is assumed to be relative to it. If *base* is an empty string (`""`) then the base is taken as the local directory or URL base containing the CFA-netCDF file.

If *base* has not been set then the file name is assumed to be an absolute path (local file or URL).

If missing or an empty string (`""`) then it is assumed to be the CFA-netCDF file itself.

Examples
`"/home/me/file.nc"`
`"../file2.pp"`
`"file3.nc"`
`"http://archive/data/file.nc"`
`"data/file.nc"`
`""`

dtype (*optional*) The data type of the partition's sub-array. Any of the netCDF data type strings are allowed.

If missing then the data type of the master array is assumed.

Examples
`"double"`
`"byte"`
`"char"`

ncvar (*for netCDF files only, optional*) A string containing the name of the netCDF variable containing the partition's sub-array.

Examples
`"tas"`
`"latitude"`
`"cfa_678df4g745"`

If missing then the *varid* attribute must be set.

If both the *ncvar* and *varid* attributes are set then *varid* is ignored.

varid (*for netCDF files only, optional*) The integer UNIDATA netCDF interface ID of the variable containing the partition's sub-array.

Examples
`24`
`0`

If missing then the *ncvar* attribute must be set.

If both the *ncvar* and *varid* attributes are set then *varid* is ignored.

file_offset (*not required for netCDF files*) The non-negative integer byte address of the file where the metadata of the partition's sub-array begins. The start of the file is addressed by zero.

Examples
`0`
`8460364`