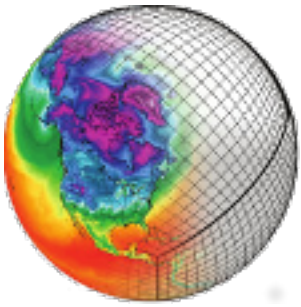


cdms

Release 3.1.0

Denis Nadeau

May 08, 2019

CONTENTS

Version : 3.1.0

CHAPTER

ONE

OVERVIEW

The Community Data Management System is an object-oriented data management system, specialized for organizing multidimensional, gridded data used in climate analysis and simulation.

CDMS is implemented as part of the Climate Data Analysis Tool (CDAT), which uses the Python language. The examples in this chapter assume some familiarity with the language and the Python Numpy module (http://www.numpy.org). A number of excellent tutorials on Python are available in books or on the Internet. For example, see the Python Foundation's homepage.

CHAPTER

TWO

VARIABLES

The basic unit of computation in CDMS is the variable. A variable is essentially a multidimensional data array, augmented with a domain, a set of attributes, and optionally a spatial and/or temporal coordinate system (see *Coordinate Axes*). As a data array, a variable can be sliced to obtain a portion of the data, and can be used in arithmetic computations. For example, if `u` and `v` are variables representing the eastward and northward components of wind speed, respectively, and both variables are functions of time, latitude, and longitude, then the velocity for time 0 (first index) can be calculated as:

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/clt.nc"
>>> f1=cdms2.open("clt.nc")
>>> u = f1('u')
>>> v = f1('v')
>>> from cdms2 import MV
>>> vel = MV.sqrt(u[0]**2 + v[0]**2)
```

This illustrates several points:

- Square brackets represent the slice operator. Indexing starts at 0, so `u[0]` selects from variable `u` for the first timepoint. The result of this slice operation is another variable. The slice operator can be multidimensional, and follows the syntax of Numpy Python arrays. In this example, `u[0:10,:,1]` would retrieve data for the first ten timepoints, at all latitudes, for the second longitude.
- Variables can be used in computation. `**` is the Python exponentiation operator.
- Arithmetic functions are defined in the `cdms2.MV2` module.
- Operations on variables carry along the corresponding metadata where applicable. In the above example, `vel` has the same latitude and longitude coordinates as `u` and `v`, and the time coordinate is the first time of `u` and `v`.

CHAPTER

THREE

FILE I/O

A variable can be obtained from a file or collection of files, or can be generated as the result of a computation.

Files can be in any of theself- describing formats:

netCDF, HDF, GrADS/GRIB (GRIB with a GrADS control file), or PCMDI DRS.

Note: (HDF and DRS support is optional, and is configured at the time CDAT is installed.)

For instance, to read datafrom file sample.nc into variable u:

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/clt.nc"
>>> f = cdms2.open('clt.nc')
>>> u = f('u')
```

Data can be read by index or by world coordinate values. The following reads the n-th timepoint of u (the syntax slice(i,j) refers to indices k such that i <= k < j):

```
>>> n = 0
>>> u0 = f('u',time=slice(n,n+1))
```

To read `u` at time 1.:

```
>>> u1 = f('u',time=1.)
```

A variable can be written to a file with the write function:

```
>>> g = cdms2.open('sample2.nc','w')
>>> g.write(u)
<cdms2.fvariable.FileVariable object at ...
>>> g.close()
```

CHAPTER

FOUR

COORDINATE AXES

A coordinate axis is a variable that represents coordinate information. Typically an axis is associated with one or more variables in a file or dataset, to represent the indexing and/or spatiotemporal coordinate system(s) of the variable(s).

Often in climate applications an axis is a one-dimensional variable whose values are floating-point and strictly monotonic. In some cases an axis can be multidimensional (see *Grids*). If an axis is associated with one of the canonical types latitude, longitude, level, or time, then the axis is called temporal .

The shape and physical ordering of a variable is represented by the variables domain , an ordered tuple of one-dimensional axes. In the previous example, the domain of the variable u is the tuple (time, latitude, longitude). This indicates the order of the dimensions, with the slowest-varying dimension listed first (time). The domain may be accessed with the `getAxisList()` method:

```
>>> u.getAxisList()
[   id: time1
Designated a time axis.
units:  months since 1978-12
Length: 1
First:  1.0
Last:   1.0
Other axis attributes:
   calendar: gregorian
   axis: T
Python id:  ...
,    id: plev
Designated a level axis.
units:  hPa
Length: 2
First:  200.0
Last:   850.0
Other axis attributes:
   axis: Z
   realtopology: linear
Python id:  ...
,    id: latitude1
Designated a latitude axis.
units:  degrees_north
Length: 80
First:  -88.2884
Last:   88.2884
Other axis attributes:
   axis: Y
   realtopology: linear
Python id:  ...
,    id: longitude1
```

(continues on next page)

(continued from previous page)

```
Designated a longitude axis.
units:  degrees_east
Length: 97
First:  -180.0
Last:   180.0
Other axis attributes:
   axis: X
   topology: circular
   modulo: 360.0
   realtopology: linear
Python id:  ...
]
```

In the above example, the domain elements are axes that are also spatiotemporal. In general it is not always the case that an element of a domain is spatiotemporal:

- An axis in the domain of a variable need not be spatiotemporal. For example, it may represent a range of indices, an index coordinate system.
- The latitude and/or longitude coordinate axes associated with a variable need not be elements of the domain. In particular this will be true if the variable is defined on a non-rectangular grid (see *Grids*).

As previously noted, a spatial and/or temporal coordinate system may be associated with a variable. The methods getLatitude, getLongitude, getLevel, and getTime return the associated coordinate axes. For example:

```
>>> t = u.getTime()
>>> print t[:]
[ 1.]
>>> print t.units
months since 1978-12
```

CHAPTER

FIVE

ATTRIBUTES

As mentioned above, variables can have associated attributes , name-value pairs. In fact, nearly all CDMS objects can have associated attributes, which are accessed using the Python dot notation:

```
>>> u.units='m/s'
>>> print u.units
m/s
```

Attribute values can be strings, scalars, or 1-D Numpy arrays.

When a variable is written to a file, not all the attributes are written. Some attributes, called internal attributes, are used for bookkeeping, and are not intended to be part of the external file representation of the variable. In contrast, external attributes are written to an output file along with the variable. By default, when an attribute is set, it is treated as external. Every variable has a field attributes, a Python dictionary that defines the external attributes:

```
>>> print u.attributes.keys()
'name', 'title', 'tileIndex', 'date', 'source', 'time', 'units', 'type']
```

The Python dir command lists the internal attribute names:

```
>>> dir(u)
['T', '_FillValue', '_TransientVariable__domain', ..., 'view']
```

In general internal attributes should not be modified directly. One exception is the id attribute, the name of the variable. It is used in plotting and I/O, and can be set directly.

CHAPTER

SIX

MASKED VALUES

Optionally, variables have a mask that represents where data are missing. If present, the mask is an array of ones and zeros having the shape of the data array. A mask value of one indicates that the corresponding data array element is missing or invalid.

Arithmetic operations in CDMS take missing data into account. The same is true of the functions defined in the cdms2.MV2 module. For example:

```
>>> a = MV2.array([1,2,3]) # Create array a, with no mask
>>> b = MV2.array([4,5,6]) # Same for b
>>> a+b # variable_... array([5,7,9,])
variable_...
masked_array(data = [5 7 9],
         mask = False,
    fill_value = 999999)
>>>
>>>
>>> a[1]=MV2.masked # Mask the second value of a a.mask()
>>> a.mask
array([False,  True, False], dtype=bool)
>>> a+b # The sum is masked also
variable_...
masked_array(data = [5 -- 9],
         mask = [False  True False],
    fill_value = 999999)
```

When data is read from a file, the result variable is masked if the file variable has a missing_value attribute. The mask is set to one for those elements equal to the missing value, zero elsewhere. If no such attribute is present in the file, the result variable is not masked.

When a variable with masked values is written to a file, data values with a corresponding mask value of one are set to the value of the variables `missing_value` attribute. The data and `missing_value` attribute are then written to the file.

Masking is covered in Section 2.9. See also the documentation of the Python Numpy and MA modules, on which `cdms.MV` is based, at

https://www.numpy.org/.

CHAPTER

SEVEN

FILE VARIABLES

A variable can be obtained either from a file, a collection of files, or as the result of computation. Correspondingly there are three types of variables in CDMS:

- *file variable* is a variable associated with a single data file. Setting or referencing a file variable generates I/O operations.
- *dataset variable* is a variable associated with a collection of files. Reference to a dataset variable reads data, possibly from multiple files. Dataset variables are read-only.
- *transient variable* is an in-memory object not associated with a file or dataset. Transient variables result from a computation or I/O operation.

Typical use of a file variables is to inquire information about the variable in a file without actually reading the data for the variables. A file variable is obtained by applying the slice operator `[]` to a file, passing the name of the variable, or by calling the getVariable function.

Note That obtaining a file variable does not actually read the data array:

```
>>> u = f.getVariable('u') # or u=f['u']
>>> u.shape
(1, 2, 80, 97)
```

File variables are also useful for fine-grained I/O. They behave like transient variables, but operations on them also affect the associated file. Specifically:

- slicing a file variable reads data,
- setting a slice writes data,
- referencing an attribute reads the attribute,
- setting an attribute writes the attribute,
- and calling a file variable like a function reads data associated with the variable:

```
>>> import os
>>> os.system("cp clt.nc /tmp")
0
>>> f = cdms2.open('/tmp/clt.nc','a') # Open read/write
>>> uvar = f['u'] # Note square brackets
>>> uvar.shape
(1, 2, 80, 97)
>>> u0 = uvar[0] # Reads data from sample.nc
>>> u0.shape
(2, 80, 97)
>>> uvar[1]=u0 # Writes data to sample.nc
>>> uvar.units # Reads the attribute 'm/s'
```

(continues on next page)

(continued from previous page)

```
'm/s'
>>> u24 = uvar(time=1.0) # Calling  a variable like a function reads data
>>> f.close() # Save changes to clt.nc (I/O may be buffered)
```

For transient variables, the data is printed only if the size of the array is less than the print limit . This value can be set with the function MV.set_print_limit to force the data to be printed:

```
>>> MV2.get_print_limit() # Current limit 1000
1000
>>> smallvar
small variable
masked_array(data =
  [[  0.   1.   2.   3.   4.]
  [  5.   6.   7.   8.   9.]
  [ 10.  11.  12.  13.  14.]
  [ 15.  16.  17.  18.  19.]],
   mask =
  False,
       fill_value = 999999.0)
>>> MV2.set_print_limit(100)
>>> largevar
large variable
masked_array(data =
[[   0.    1.    2. ...,   17.   18.   19.]
[  20.   21.   22. ...,   37.   38.   39.]
[  40.   41.   42. ...,   57.   58.   59.]
...,
[ 340.  341.  342. ...,  357.  358.  359.]
[ 360.  361.  362. ...,  377.  378.  379.]
[ 380.  381.  382. ...,  397.  398.  399.]],
mask = False,
fill_value = 999999.0)
```

The datatype of the variable is determined with the typecode function:

```
>>> u.typecode()
'f'
```

CHAPTER

EIGHT

DATASET VARIABLES

The third type of variable, a *dataset variable*, is associated with a *dataset*, a collection of files that is treated as a single file. A dataset is created with the `cdscan` utility. This generates an XML metafile that describes how the files are organized and what metadata are contained in the files. In a climate simulation application, a dataset typically represents the data generated by one run of a general circulation or coupled ocean-atmosphere model.

For example, suppose data for variables u and v are stored in six files:

1. u_2000.nc,
2. u_2001.nc,
3. u_2002.nc,
4. v_2000.nc,
5. v_2001.nc,
6. v_2002.nc.

A metafile can be generated with the command:

$ cdscan -x cdsample.xml [uv]*.nc

The metafile **cdsample.xml** is then used like an ordinary data file:

```
>>> import os
>>> os.system("cdscan -x cdsample.xml [uv]*.nc")
0
>>> f = cdms2.open('cdsample.xml')
>>> u = f('u')
>>> u.shape
(3, 16, 32)
```

CHAPTER

NINE

GRIDS

A latitude-longitude grid represents the coordinate information associated with a variable. A grid encapsulates:

- latitude, longitude coordinates
- grid cell boundaries
- area weights

CDMS defines a rich set of grid types to represent the variety of coordinate systems used in climate model applications. Grids can be categorized as rectangular or nonrectangular.

A rectangular grid has latitude and longitude axes that are one-dimensional, with strictly monotonic values. The grid is essentially the Cartesian product of the axes. If either criterion is not met, the grid is nonrectangular .

CDMS supports two types of nonrectangular grid:

- A curvilinear grid consists of a latitude and longitude axis, each of which is a two-dimensional coordinate axis. Curvilinear grids are often used in ocean model applications.
- A generic grid consists of a latitude and longitude axis, each of which is an auxiliary one-dimensional coordinate axis. An auxiliary axis has values that are not necessarily monotonic. As the name suggests, generic grids can represent virtually any type of grid. However, it is more difficult to determine adjacency relationships between grid points.

9.1 Example: A Curvilinear Grid

In this example, variable sample is defined on a 128x192 curvilinear grid. Note that:

- The domain of variable sample is (y , x) where y and x are index coordinate axes.
- The curvilinear grid associated with sample consists of axes (lat , lon), each a two-dimensional coordinate axis.
- lat and lon each have domain (y , x)

```
>>> f = cdms2.open('sampleCurveGrid4.nc')
>>> # lat and lon are coordinate axes, but are grouped with data variables
>>> f.variables.keys()
['lat', 'sample', 'bounds_lon', 'lon', 'bounds_lat']
>>> # y and x are index coordinate axes
>>> f.axes.keys()
['nvert', 'x', 'y']
>>> # Read data for variable sample
>>> sample = f('sample')
>>> # The associated grid g is curvilinear
>>> g = sample.getGrid()
```

(continues on next page)

(continued from previous page)

```
>>> # The domain of the variable consists of index axes
>>> sample.getAxisIds()
['y', 'x']
>>> # Get the coordinate axes associated with the grid
>>> lat = g.getLatitude() # or sample.getLatitude()
>>> lon = g.getLongitude() # or sample.getLongitude()
>>> # lat and lon have the same domain, a subset of the domain of 'sample'
>>> lat.getAxisIds()
['y', 'x']
>>> # lat and lon are variables ...
>>> lat.shape
(32, 48)
>>> lat
lat
masked_array(data =
[[-76.08465554 -76.08465554 -76.08465554 ..., -76.08465554 -76.08465554
-76.08465554]
[-73.92641847 -73.92641847 -73.92641847 ..., -73.92641847 -73.92641847
-73.92641847]
[-71.44420823 -71.44420823 -71.44420823 ..., -71.44420823 -71.44420823
-71.44420823]
...,
[ 42.32854943  42.6582209   43.31990211 ...,  43.3199019   42.65822088
42.32854943]
[ 42.70106429  43.05731498  43.76927818 ...,  43.76927796  43.05731495
42.70106429]
[ 43.0307341   43.41264383  44.17234165 ...,  44.17234141  43.41264379
43.0307341 ]],
mask = False, fill_value = 1e+20)
>>> lat_in_radians = lat*MV2.pi/180.0
```

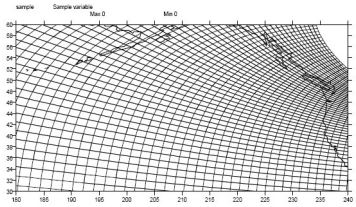


Fig. 1: Figure 1: Curvilinear Grid

9.2 Example: A Generic Grid

In this example variable zs is defined on a generic grid. Figure 2 illustrates the grid, in this case a geodesic grid:

```
>>> f.variables.keys()
['lat', 'sample', 'bounds_lon', 'lon', 'bounds_lat']
>>> f.axes.keys()
['nvert', 'x', 'y']
>>> zs = f('sample')
>>> g = zs.getGrid()
>>> g
<TransientCurveGrid, id: ..., shape: (32, 48)>
```

(continues on next page)

(continued from previous page)

```
>>> lat = g.getLatitude()
>>> lon = g.getLongitude()
>>> lat.shape
(32, 48)
>>> lon.shape # variable zs is defined in terms of a single index coordinate
(32, 48)
>>> # axis, 'cell'
>>> zs.shape
(32, 48)
>>> zs.getAxisIds()
['y', 'x']
>>> # lat and lon are also defined in terms of the cell axis
>>> lat.getAxisIds()
['y', 'x']
>>> # lat and lon are one-dimensional, 'auxiliary' coordinate
>>> # axes: values are not monotonic
>>> lat.__class__
<class 'cdms2.coord.TransientAxis2D'>
```

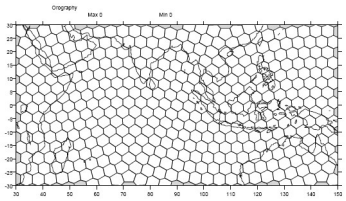


Fig. 2: Figure 2: Generic Grid

Generic grids can be used to represent any of the grid types. The method toGenericGrid can be applied to any grid to convert it to a generic representation. Similarly, a rectangular grid can be represented as curvilinear. The method toCurveGrid is used to convert a non-generic grid to curvilinear representation:

```
>>> f = cdms2.open(cdat_info.get_sampledata_path()+'/clt.nc')
>>> clt = f('clt')
>>> rectgrid = clt.getGrid()
>>> rectgrid.shape
(46, 72)
>>> curvegrid = rectgrid.toCurveGrid()
>>> curvegrid
<TransientCurveGrid, id: ..., shape: (46, 72)>
>>> genericgrid = curvegrid.toGenericGrid()
>>> genericgrid
<TransientGenericGrid, id: ..., shape: (3312,)>
```

CHAPTER

TEN

REGRIDDING

Regridding is the process of mapping variables from one grid to another. CDMS supports two forms of regridding. Which one you use depends on the class of grids being transformed:

- To interpolate from one rectangular grid to another, use the built-in CDMS regridder. CDMS also has built-in regridders to interpolate from one set of pressure levels to another, or from one vertical cross-section to another.
- To interpolate from any lat-lon grid, rectangular or non-rectangular, use the SCRIP regridder.

10.1 CDMS Regridder

The built-in CDMS regridder is used to transform data from one rectangular grid to another. For example, to regrid variable `u` (from a rectangular grid) to a 96x192 rectangular Gaussian grid:

```
>>> f = cdms2.open('clt.nc')
>>> u = f('u')
>>> u.shape
(1, 2, 80, 97)
>>> t63_grid = cdms2.createGaussianGrid(96)
>>> u63 = u.regrid(t63_grid)
>>> u63.shape
(1, 2, 96, 192)
```

To regrid a variable `uold` to the same grid as variable `vnew`:

```
>>> f = cdms2.open('clt.nc')
>>> uold = f('u')
>>> unew = f2('uwnd')
>>> uold.shape
(1, 2, 80, 97)
>>> unew.shape
(1, 14, 181, 360)
>>> t63_grid = unew.getGrid() # Obtain the grid for vnew
>>> u63 = uold.regrid(t63_grid)
>>> u63.shape
(1, 2, 181, 360)
```

10.2 SCRIP Regridder

To interpolate between any lat-lon grid types, the SCRIP regridder may be used. The SCRIP package was developed at [Los Alamos National Laboratory] (http://oceans11.lanl.gov/drupal/Models/OtherSoftware). SCRIP is written in

Fortran 90, and must be built and installed separately from the CDAT/CDMS installation.

The steps to regrid a variable are:

(External to CDMS)

1. Obtain or generate the grids, in SCRIP netCDF format.
2. Run SCRIP to generate a *remapping* file.

(In CDMS)

1. Read the regridder from the SCRIP remapping file.
2. Call the regridder with the source data, returning data on the target grid.

Steps 1 and 2 need only be done once. The regridder can be used as often as necessary.

For example, suppose the source data on a T42 grid is to be mapped to a POP curvilinear grid. Assume that SCRIP generated a remapping file named rmp_T42_to_POP43_conserv.nc:

```
>>> # Import regrid package for regridder functions
>>> import regrid2, cdms2

>>> # Get the source variable
>>> f = cdms2.open('sampleT42Grid.nc')
>>> dat = f('src_array')
>>> f.close()

>>> # Read the regridder from the remapper file
>>> remapf = cdms2.open('rmp_T42_to_POP43_conserv.nc')
>>> regridf = regrid2.readRegridder(remapf)
>>> remapf.close()

>>> # Regrid the source variable
>>> popdat = regridf(dat)
```

Regridding is discussed in Chapter 4.

CHAPTER

ELEVEN

TIME TYPES

CDMS provides extensive support for time values in the cdtime module. cdtime also defines a set of calendars , specifying the number of days in a given month.

Two time types are available: relative time and component time . Relative time is time relative to a fixed base time. It consists of:

- a `units` string, of the form `"units since basetime"` , and
- a floating-point `value`

For example, the time "28.0 days since 1996-1-1" has value= 28.0 , and units=" days since 1996-1-1". To create a relative time type:

```
>>> import cdtime
>>> rt = cdtime.reltime(28.0, "days since 1996-1-1")
>>> rt
28.000000 days since 1996-1-1
>>> rt.value
28.0
>>> rt.units
'days since 1996-1-1'
```

A component time consists of the integer fields year, month, day, hour, minute , and the floating-point field second . For example:

```
>>> ct = cdtime.comptime(1996,2,28,12,10,30)
>>> ct
1996-2-28 12:10:30.0
>>> ct.year
1996
>>> ct.month
2
```

The conversion functions tocomp and torel convert between the two representations. For instance, suppose that the time axis of a variable is represented in units " days since 1979" . To find the coordinate value corresponding to January 1, 1990:

```
>>> ct = cdtime.comptime(1990,1)
>>> rt = ct.torel("days since 1979")
>>> rt.value
4018.0
```

Time values can be used to specify intervals of time to read. The syntax time=(c1,c2) specifies that data should be read for times t such that c1<=t<=c2:

```
>>> fh = cdms2.open(cdat_info.get_sampledata_path() + "/tas_6h.nc")
>>> c1 = cdtime.comptime(1980,1)
>>> c2 = cdtime.comptime(1980,2)
>>> tas = fh['tas']
>>> tas.shape
(484, 45, 72)
>>> x = tas.subRegion(time=(c1,c2))
>>> x.shape
(125, 45, 72)
```

or string representations can be used:

```
>>> fh = cdms2.open(cdat_info.get_sampledata_path() + "/tas_6h.nc")
>>> tas = fh['tas']
>>> x = tas.subRegion(time=('1980-1','1980-2'))
```

Time types are described in Chapter 3.

CHAPTER

TWELVE

PLOTTING DATA

Data read via the CDMS Python interface can be plotted using the vcs module. This module, part of the Climate Data Analysis Tool (CDAT) is documented in the VCS reference manual. The vcs module provides access to the functionality of the VCS visualization program.

To generate a plot:

- Initialize a canvas with the `vcs init` routine.
- Plot the data using the canvas `plot` routine.

For example:

```
>>> import cdms2, vcs, cdat_info
>>> fh=cdms2.open(cdat_info.get_sampledata_path() + "/tas_cru_1979.nc")
>>> fh['time'][:] # Print the time coordinates
array([ 1476.,  1477.,  1478.,  1479.,  1480.,  1481.,  1482.,  1483.,
        1484.,  1485.,  1486.,  1487.])
>>>
>>> tas = fh('tas', time=1479)
>>> tas.shape
(1, 36, 72)
>>> w = vcs.init() # Initialize a canvas
>>> w.plot(tas) # Generate a plot
<vcs.displayplot.Dp object at...
```

By default for rectangular grids, a boxfill plot of the lat-lon slice is produced. Since variable precip includes information on time, latitude, and longitude, the continental outlines and time information are also plotted. If the variable were on a non-rectangular grid, the plot would be a meshfill plot.

The plot routine has a number of options for producing different types of plots, such as isofill and x-y plots. See Chapter 5 for details.

CHAPTER

THIRTEEN

CDMS PYTHON APPLICATION PROGRAMMING INTERFACE

13.1 Overview

This chapter describes the CDMS Python application programming interface (API). Python is a popular public-domain, object-oriented language. Its features include support for object-oriented development, a rich set of programming constructs, and an extensible architecture. CDMS itself is implemented in a mixture of C and Python. In this chapter the assumption is made that the reader is familiar with the basic features of the Python language.

Python supports the notion of a module, which groups together associated classes and methods. The import command makes the module accessible to an application. This chapter documents the cdms, cdtime, and regrid modules.

The chapter sections correspond to the CDMS classes. Each section contains table s base. If no parent, the datapath is absolute.describing the class internal (non-persistent) attributes, constructors (functions for creating an object), and class methods (functions). A method can return an instance of a CDMS class, or one of the Python types:

13.1.1 PythonTypes used in CDMS

Type	Description
Array	Numpy or masked multidimensional data array. All elements of the array are of the same type. Defined in the Numpy and MV2 modules.
Comptime	Absolute time value, a time with representation (year, month, day, hour, minute, second). Defined in the cdtime module. cf. reltime
Dictionary	An unordered 2,3 collection of objects, indexed by key. All dictionaries in CDMS are indexed by strings, e.g.: `axes['time']`
Float	Floating-point value.
Integer	Integer value.
List	An ordered sequence of objects, which need not be of the same type. New members can be inserted or appended. Lists are denoted with square brackets, e.g., `[1, 2.0, 'x', 'y']`
None	No value returned.
Reltime	Relative time value, a time with representation (value, units since basetime). Defined in the cdtime module. cf. comptime
Tuple	An ordered sequence of objects, which need not be of the same type. Unlike lists, tuples elements cannot be inserted or appended. Tuples are denoted with parentheses, e.g., `(1, 2.0, 'x', 'y')`

13.2 A First Example

The following Python script reads January and July monthly temperature data from an input dataset, averages over time, and writes the results to an output file. The input temperature data is ordered (time, latitude, longitude).

```
>>> import cdms2, cdat_info
>>> from cdms2 import MV
>>> jones = cdms2.open(cdat_info.get_sampledata_path()+'/tas_mo.nc')
>>> tasvar = jones['tas']
>>> jans = tasvar[0::12]
>>> julys = tasvar[6::12]
>>> janavg = MV.average(jans)
>>> janavg.id = "tas_jan"
>>> janavg.long_name = "mean January surface temperature"
>>> julyavg = MV.average(julys)
>>> julyavg.id = "tas_jul"
>>> julyavg.long_name = "mean July surface temperature"
>>> out = cdms2.open('janjuly.nc','w')
>>> out.write(janavg)
<cdms2.fvariable.FileVariable object...
>>> out.write(julyavg)
<cdms2.fvariable.FileVariable object...
>>> out.comment = "Average January/July from Jones dataset"
>>> jones.close()
>>> out.close()
```

Line	Notes
2,3	Makes the CDMS and MV modules available. MV defines arithmetic functions.
4	Opens a netCDF file read-only. The result jones is a dataset object.
5	**Gets the surface air temperature variable. 'tas' is the name of the variable in the input dataset.** This does not actually read the data.
6	**Read all January monthly mean data into a variable jans.** • Variables can be sliced like arrays. • The slice operator [0::12] means take every 12th slice from dimension 0, starting at index 0 and ending at the last index. • If the stride 12 were omitted, it would default to 1. **Note: That the variable is actually 3-dimensional.** • Since no slice is specified for the second or third dimensions, all values of those 2,3 dimensions are retrieved. • The slice operation could also have been written [0::12, : , :]. **Also Note: That the same script works for multi-file datasets.** • CDMS opens the needed data files, extracts the appropriate slices, and concatenates them into the result array.
7	Reads all July data into a masked array julys.
8	Calculate the average January value for each grid zone. Any missing data is handled automatically.
9,10	**Set the variable id and long_name attributes.** • The id is used as the name of the variable when plotted or written to a file.
14	Create a new netCDF output file named 'janjuly.nc' to hold the results.
15	**Write the January average values to the output file.** • The variable will have id "tas_jan" in the file. `write` is a utility function which creates the variable in the file, then writes data to the variable. • A more general method of data output is first to create a variable, then set a slice of the variable. **Note: That janavg and julavg have the same latitude and longitude information as tasvar.** It is carried along with the computations.
17	Set the global attribute 'comment'.
18	Close the output file.

13.3 Cdms Module

The Cdms Module is the Python interface to CDMS. The objects and methods in this chapter are made accessible with the command:

```
>>> import cdms2
```

The functions described in this section are not associated with a class. Rather, they are called as module functions, e.g.,

```
>>> file = cdms2.open('sample.nc')
```

13.3.1 Cdms Module Functions

Type	Definition
Variable	**asVariable(s):** **Transform s into a transient variable.** • s is a masked array, Numpy array, or Variable. • If s is already a transient variable, s is returned. • See also: isVariable.
Axis	**createAxis(data, bounds=None):** **Create a one-dimensional coordinate Axis, which is not associated with a file or dataset.** **This is useful for creating a grid which is not contained in a file or dataset.** • data is a one-dimensional, monotonic Numpy array. • bounds is an array of shape (len(data),2), such that for all i • data[i] is in the range [bounds[i,0],bounds[i,1]]. **Note:** • If bounds is not specified, the default boundaries are generated at the midpoints between the consecutive data values, provided that the autobounds mode is 'on' (the default). • See setAutoBounds. • Also see: CdmsFile.createAxis
Axis	**createEqualAreaAxis(nlat):** **Create an equal-area latitude axis.** **The latitude values range from north to south, and for all axis values x[i], sin(x[i])sin(x[i+1]) is constant.** • nlat is the axis length. **Note:** The axis is not associated with a file or dataset.
Axis	**createGaussianAxis(nlat):** **Create a Gaussian latitude axis.** **Axis values range from north to south.** • nlat is the axis length. **Note:** The axis is not associated with a file or dataset.
RectGrid	**createGaussianGrid(nlats, xorigin=0.0, order='yx'):** **Create a Gaussian grid, with shape (nlats, 2*nlats).** • nlats is the number of latitudes. • xorigin is the origin of the longitude axis. • order is either 'yx' (lat-lon, default) or 'xy' (lon-lat)

13.3.2 Cdms Module Functions(cont'd)

Type	Definition
RectGrid	**createGenericGrid(latArray, lonArray, latBounds=None** lonBounds=None, order='yx', mask=None): Create a generic grid, that is, a grid which is not typed as Gaussian, uniform, or equal-area. **The grid is not associated with a file or dataset.** • latArray is a NumPy array of latitude values. • lonArray is a NumPy array of longitude values. • latBounds is a NumPy array having shape (len(latArray),2), of latitude boundaries. • lonBounds is a NumPy array having shape (len(lonArray),2), of longitude boundaries. • **order is a string specifying the order of the axes, either 'yx' for (latitude,** longitude), or 'xy' for the reverse. • mask (optional) is an integer-valued NumPy mask array, having the same shape and ordering as the grid.
RectGrid	**createGlobalMeanGrid(grid):** **Generate a grid for calculating the global mean via a regridding operation.** **The return grid is a single zone covering the range of he input grid.** • grid is a RectGrid.
RectGrid	**createRectGrid(lat, lon, order, type='generic', mask=None):** **Create a rectilinear grid, not associated with a file or dataset.** **This might be used as the target grid for a regridding operation.** • lat is a latitude axis, created by cdms.createAxis. • lon is a longitude axis, created by cdms.createAxis. • order is a string with value 'yx' (the first grid dimension is latitude) or 'xy' (the first grid dimension is longitude). • type is one of 'gaussian','uniform','equalarea',or 'generic'. **Note: If specified, mask is a two-dimensional, logical Numpy array (all values** are zero or one) with the same shape as the grid.
RectGrid	**createUniformGrid(startLat, nlat, deltaLat,** start-Lon, nlon, deltaLon, order='yx', mask=None) **Create a uniform rectilinear grid.** The grid is not associated with a file or dataset. The grid boundaries are at the midpoints of the axis values. • startLat is the starting latitude value. • nlat is the number of latitudes. • If nlat is 1, the grid latitude boundaries will be startLat +/- deltaLat/2. • deltaLat is the increment between latitudes. • startLon is the starting longitude value. • nlon is the number of longitudes. • If nlon is 1, the grid longitude boundaries will be startLon +/- deltaLon/2. • deltaLon is the increment between longitudes. • order is a string with value 'yx. (the first grid dimension is latitude) or .xy. (the first grid dimension is longitude). **Note: If specified, mask is a two-dimensional, logical Numpy array (all values** are zero or one) with the same shape as the grid.

13.3.3 Cdms Module Functions(cont'd)

Type	Definition
Axis	**createUniformLatitudeAxis(startLat , nlat, deltaLat):** **Create a uniform latitude axis.** The axis boundaries are at the midpoints of the axis values. The axis is designated as a circular latitude axis. • `startLat` is the starting latitude value. • `nlat` is the number of latitudes. • `deltaLat` is the increment between latitudes.
RectGrid	**createZonalGrid(grid):** **Create a zonal grid.** The output grid has the same latitude as the input grid, and a single longitude. This may be used to calculate zonal averages via a regridding operation. • `grid` is a RectGrid.
Axis	**createUniformLongitudeAxis(startLon, nlon, delta-Lon):** **Create a uniform longitude axis.** The axis boundaries are at the midpoints of the axis values. The axis is designated as a circular longitude axis. • `startLon` is the starting longitude value. • `nlon` is the number of longitudes. • `deltaLon` is the increment between longitudes.
Variable	`createVariable(array, typecode=None, copy=0, savespace=0, mask=None, fill_value=None, grid=None, axes=None , attributes=None, id=None):`
Integer	**getAutoBounds():** **Get the current autobounds mode.** • Returns 0, 1, or 2. • See `setAutoBounds`.
Integer	**isVariable(s):** • Return `1` if `s` is a variable, `0` otherwise. • See also: `asVariable`.

13.3.4 Cdms Module Functions(cont'd)

Type	Definition
Dataset	**`open(url,mode='r'):`** **Open or create a `Dataset` or `CdmsFile`.** • `url` is a Uniform Resource Locator, referring to a cdunif or XML file. • If the URL has the extension '.xml' or '.cdml', a `Dataset` is returned, otherwise a `CdmsFile` is returned. • If the URL protocol is 'http', the file must be a '.xml' or '.cdml' file, and the mode must be 'r'. • If the protocol is 'file' or is omitted, a local file or dataset is opened. • `mode` is the open mode. See *Open Modes* **Example:** Open an existing dataset: `f = cdms.open('sampleset.xml')` **Example:** Create a netCDF file: `f = cdms.open('newfile.nc','w')`
List	**`order2index (axes, orderstring):`** **Find the index permutation of axes to match order.** • Return a list of indices. • `axes` is a list of axis objects. • `orderstring` is defined as in `orderparse`.
List	**`orderparse(orderstring):`** **Parse an order string.** • Returns a list of axes specifiers. **`orderstring` consists of:** • Letters t, x, y, z meaning time, longitude, latitude, level • Numbers 0-9 representing position in axes • Dash (-) meaning insert the next available axis here. • The ellipsis . . . meaning fill these positions with any remaining axes. • (name) meaning an axis whose id is name

13.3.5 Cdms Module Functions(cont'd)

Type	Definition
None	**setAutoBounds(mode):** **Set autobounds mode.** In some circumstances CDMS can generate boundaries for 1-D axes and rectilinear grids, when the bounds are not explicitly defined. The autobounds mode determines how this is done: • If `mode` is `'grid'` or `2` (the default), the `getBounds` method will automatically generate boundary information for an axis or grid if the axis is designated as a latitude or longitude axis, and the boundaries are not explicitly defined. • If `mode` is `'on'` or `1`, the `getBounds` method will automatically generate boundary information for an axis or grid, if the boundaries are not explicitly defined. • If `mode` is `'off'` or `0`, and no boundary data is explicitly defined, the bounds will NOT be generated; the `getBounds` method will return `None` for the boundaries. **Note:** In versions of CDMS prior to V4.0, the default `mode` was `'on'`.
None	**setClassifyGrids(mode):** **Set the grid classification mode.** This affects how grid type is determined, for the purpose of generating grid boundaries. • If `mode` is `'on'` (the default), grid type is determined by a grid classification method, regardless of the value of `grid.get-Type()`. • If `mode` is `'off'`, the value of `grid.getType()` determines the grid type.
None	**writeScripGrid(path, grid, gridTitle=None):** **Write a grid to a SCRIP grid file.** • `path` is a string, the path of the SCRIP file to be created. • `grid` is a CDMS grid object. It may be rectangular. `gridTitle` is a string ID for the grid.

13.3.6 Class Tags

Tag	Class
'axis'	Axis
'database'	Database
'dataset'	Dataset, CdmsFile
'grid'	RectGrid
'variable'	Variable
'xlink'	Xlink

13.4 CdmsObj

A CdmsObj is the base class for all CDMS database objects. At the application level, CdmsObj objects are never created and used directly. Rather the subclasses of CdmsObj (Dataset, Variable, Axis, etc.) are the basis of user application programming.

All objects derived from CdmsObj have a special attribute .attributes. This is a Python dictionary, which contains all the external (persistent) attributes associated with the object. This is in contrast to the internal, non-persistent attributes of an object, which are built-in and predefined. When a CDMS object is written to a file, the external attributes are written, but not the internal attributes.

Example: Get a list of all external attributes of obj.

13.4.1 Attributes Common to All CDMS Objects

Type	Name	Definition
Dictionary	attributes	External attribute dictionary for this object.

13.4.2 Getting and Setting Attributes

Type	Definition
various	**`value = obj.attname`** **Get an internal or external attribute value.** • If the attribute is external, it is read from the database. • If the attribute is not already in the database, it is created as an external attribute. **Note:** Internal attributes cannot be created, only referenced.
various	**`obj.attname = value`** **Set an internal or external attribute value.** • If the attribute is external, it is written to the database.

13.5 CoordinateAxis

A CoordinateAxis is a variable that represents coordinate information. It may be contained in a file or dataset, or may be transient (memoryresident). Setting a slice of a file CoordinateAxis writes to the file, and referencing a file CoordinateAxis slice reads data from the file. Axis objects are also used to define the domain of a Variable.

CDMS defines several different types of CoordinateAxis objects. See *MV module* documents methods that are common to all CoordinateAxis types. See *HorizontalGrid* specifies methods that are unique to 1D Axis objects.

13.5.1 CoordinateAxis Types

Type	Definition
CoordinateAxis	**A variable that represents coordinate information.** • Has subtypes `Axis2D` and `AuxAxis1D`.
Axis	**A one-dimensional coordinate axis whose values are strictly monotonic.** • Has subtypes `DatasetAxis`, `FileAxis`, and `TransientAxis`. • May be an index axis, mapping a range of integers to the equivalent floating point value. • If a latitude or longitude axis, may be associated with a `RectGrid`.
Axis2D	**A two-dimensional coordinate axis, typically a latitude or longitude axis related** to a `CurvilinearGrid`. • Has subtypes `DatasetAxis2D`, `FileAxis2D`, and `TransientAxis2D`.
AuxAxis1D	**A one-dimensional coordinate axis whose values need not be monotonic.** • Typically a latitude or longitude axis associated with a `GenericGrid`. • Has subtypes `DatasetAuxAxis1D`, `FileAuxAxis1D`, and `TransientAuxAxis1D`. • An axis in a `CdmsFile` may be designated the unlimited axis, meaning that it can be extended in length after the initial definition. • There can be at most one unlimited axis associated with a `CdmsFile`.

13.5.2 CoordinateAxis Internal Attributes

Type	Name	Definition
Dictionary	`attributes`	External attribute dictionary.
String	`id`	CoordinateAxis identifier.
Dataset	`parent`	The dataset which contains the variable.
Tuple	`shape`	The length of each axis.

13.5.3 Axis Constructors

Constructor	Description
`cdms.createAxis(data, bounds=None)`	**Create an axis which is not associated with a dataset or file.** • See *A First Example*.
`Dataset.createAxis(name,ar)`	Create an `Axis` in a `Dataset`. (This function is not yet implemented.)
`CdmsFile.createAxis(name,ar, unlimited=0)`	**Create an Axis in a `CdmsFile`.** • `name` is the string `name` of the `Axis`. • `ar` is a 1-D data array which defines the `Axis` values. • It may have the value `None` if an unlimited axis is being defined. • At most one `Axis` in a `CdmsFile` may be designated as being unlimited, meaning that it may be extended in length. To define an axis as unlimited, either: – A) set `ar` to `None`, and leave `unlimited` undefined, or – B) set `ar` to the initial 1-D array, and set `unlimited` to `cdms.Unlitmited` ∗ `cdms.createEqualAreaAxis(nlat)` ∗ See *A First Example*. ∗ `cdms.createGaussianAxis(nlat)` ∗ See *A First Example*. ∗ `cdms.createUniformLatitudeAxis(startlat, nlat, deltalat)` ∗ See *A First Example*. ∗ `cdms.createUniformLongitudeAxis(startlon, nlon, deltalon)` ∗ See *A First Example* .

13.5.4 CoordinateAxis Methods

Type	Method	Definition
Array	`array = axis[i:j]"`	**Read a slice of data from the external file or dataset.** • Data is returned in the physical ordering defined in the dataset. • See *Variable Slice Operators* for a description of slice operators.
None	`axis[i:j] = array`	**Write a slice of data to the external file.** • Dataset axes are read-only.
None	`assignValue(array)`	**Set the entire value of the axis.** • `array` is a Numpy array, of the same dimensionality as the axis.
Axis	`clone(copyData=1)`	**Return a copy of the axis, as a transient axis.** • If copyData is 1 (the default) the data itself is copied.
None	`designateLatitude (persistent=0)`	**Designate the axis to be a latitude axis.** • If persistent is true, the external file or dataset (if any) is modified. • By default, the designation is temporary.
None	`designateLevel (persistent=0)`	**Designate the axis to be a vertical level axis.** • If persistent is true, the external file or dataset (if any) is modified. • By default, the designation is temporary.
None	`designateLongitude (persistent=0, modulo=360.0)`	**Designate the axis to be a longitude axis.** • `modulo` is the modulus value. • Any given axis value `x` is treated as equivalent to `x + modulus`. • If `persistent` is true, the external file or dataset (if any) is modified. • By default, the designation is temporary.
None	`designateTime (persistent=0, calendar = cdtime. MixedCalendar)`	**Designate the axis to be a time axis.** • If `persistent` is true, the external file or dataset (if any) is modified. • By default, the designation is temporary. • `calendar` is defined as in `getCalendar()`.

13.5.5 CoordinateAxis Methods(cont'd)

Type	Method	Definition
Array	`getBounds()`	**Get the associated boundary array.** **The shape of the return array depends on the type of axis:** • `Axis`: `(n,2)` • `Axis2D`: `(i,j,4)` • `AuxAxis1D`: `(ncell, nvert)` where nvert is the maximum number of vertices of a cell. **If the boundary array of a latitude or longitude** • `Axis` is not explicitly defined, and `autoBounds` mode is on, a default array is generated by calling `genGenericBounds`. • Otherwise if auto-Bounds mode is off, the return value is `None`. • See `setAutoBounds`.
Integer	`getCalendar()`	**Returns the calendar associated with the `(time)` axis.** **Possible return values, as defined in the `cdtime` module, are:** • `cdtime.GregorianCalendar`: the standard Gregorian calendar • `cdtime.MixedCalendar`: mixed Julian/Gregorian calendar • `cdtime.JulianCalendar`: years divisible by 4 are leap years • `cdtime.NoLeapCalendar`: a year is 365 days • `cdtime.Calendar360`: a year is 360 days • `None`: no calendar can be identified **Note:** If the axis is not a time axis, the global, file-related calendar is returned.
Array	`getValue()`	Get the entire axis vector.
Integer	`isLatitude()`	Returns true iff the axis is a latitude axis.
Integer	`isLevel()`	Returns true iff the axis is a level axis.
Integer	`isLongitude()`	Returns true iff the axis is a longitude axis.
Integer	`isTime()`	Returns true iff the axis is a time axis.
Integer	`len(axis)`	**The length of the axis if one-dimensional.** • If multidimensional, the length of the first dimension.
Integer	`size()`	The number of elements in the axis.
String	`typecode()`	The `Numpy` datatype identifier.

13.5.6 Axis Methods, Additional to CoordinateAxis

Type	Method	Definition
List of component times	`asComponentTime (calendar=None)`	**`Array` version of `cdtime tocomp`.** • Returns a `List` of component times.
List of relative times	`asRelativeTime()`	**`Array` version of `cdtime torel`.** • Returns a `List` of relative times.
None	`designate Circular( modulo, persistent=0)`	**Designate the axis to be circular.** • `modulo` is the modulus value. • Any given axis value `x` is treated as equivalent to `x + modulus`. • If `persistent` is `True`, the external file or dataset (if any) is modified. • By default, the designation is temporary.
Integer	`isCircular()`	**Returns `True` if the axis has circular topology.** **An axis is defined as circular if:** • `axis.topology == 'circular'`, or • `axis.topology` is undefined, and the axis is a longitude. • The default cycle for circular axes is 360.0
Integer	`isLinear()`	Returns `True` if the axis has a linear representation.
Tuple	`mapInterval (interval)`	Same as `mapIntervalExt`, but returns only the tuple `(i,j)`, and `wraparound` is restricted to one cycle.

13.5.7 Axis Methods, Additional to CoordinateAxis(cont'd)

Type	Method	Definition
(i,j,k)	`mapIntervalExt (interval)`	Map a coordinate interval to an index `interval`. `interval` is a tuple having one of the forms: • `(x,y)` • `(x,y,indicator)` • `(x,y,indicator,cycle)` • `None or ':'` Where `x` and `y` are coordinates indicating the interval `[x, y]`, and: • `indicator` is a two or three-character string, where the first character is `'c'` if the interval is closed on the left, `'o'` if open, and the second character has the same meaning for the right-hand point. If present, the third character specifies how the interval should be intersected with the axis • `'n'` - select node values which are contained in the interval • `'b'` -select axis elements for which the corresponding cell boundary intersects the interval • `'e'` - same as n, but include an extra node on either side • `'s'` - select axis elements for which the cell boundary is a subset of the interval • The default indicator is 'ccn', that is, the interval is closed, and nodes in the interval are selected. • If `cycle` is specified, the axis is treated as circular with the given cycle value. • By default, if `axis.isCircular()` is true, the axis is treated as circular with a default modulus of `360.0`. • An interval of `None` or `':'` returns the full index interval of the axis. • The method returns the corresponding index interval as a 3tuple `(i,j,k)`, where `k` is the integer stride, and `[i.j)` is the half-open index interval `i <= k < j (i >= k > j if k < 0)`, or `none` if the intersection is empty. • If `0 <= i < n` and `0 <= j <= n`, the interval does not wrap around the axis endpoint. • Otherwise the interval wraps around the axis endpoint. • See also: `mapinterval`, `variable.subregion()`
transient axis	`subaxis(i,j,k=1)`	Create an axis associated with the integer range `[i:j:k]`. • The stride `k` can be positive or negative. • Wraparound is supported for longitude dimensions or those with a modulus attribute.

13.5.8 Axis Slice Operators

Slice	Definition
`[i]`	The `ith` element, starting with index `0`
`[i:j]`	The `ith` element through, but not including, element `j`
`[i:]`	The `ith` element through and including the end
`[:j]`	The beginning element through, but not including, element `j`
`[:]`	The entire array
`[i:j:k]`	Every `kth` element, starting at `i`, through but not including `j`
`[-i]`	**The `ith` element from the end. `-1` is the last element.** **Example: a longitude axis has value** • `[0.0, 2.0, ..., 358.0]` • of length `180` **Map the coordinate interval:** • `-5.0 <= x < 5.0` to index interval(s), with wraparound. the result index interval • `-2 <= n < 3` wraps around, since • `-2 < 0`, and has a stride of `1` • This is equivalent to the two contiguous index intervals • `2 <= n < 0` and `0 <= n < 3`

Example 1

```
>>> axis.isCircular()
>>> 1
>>> axis.mapIntervalExt((-5.0,5.0,'co'))
>>> (-2,3,1)
```

13.6 CdmsFile

A `CdmsFile` is a physical file, accessible via the `cdunif` interface. netCDF files are accessible in read-write mode. All other formats (DRS, HDF, GrADS/GRIB, POP, QL) are accessible read-only.

As of CDMS V3, the legacy cuDataset interface is also supported by Cdms-Files. See "cu Module".

13.6.1 CdmsFile Internal Attributes

Type	Name	Definition
Dictionary	`attributes`	Global, external file attributes
Dictionary	`axes`	Axis objects contained in the file.
Dictionary	`grids`	Grids contained in the file.
String	`id`	File pathname.
Dictionary	`variables`	Variables contained in the file.

13.6.2 CdmsFile Constructors

Constructor	Description
Constructor	Description
`fileobj = cdms.open(path, mode)`	**Open the file specified by path returning a CdmsFile object.** • `path` is the file pathname, a string. • `mode` is the open mode indicator, as listed in *Open Modes*.
`fileobj = cdms.createDataset(path)`	Create the file specified by path, a string.

13.6.3 CdmsFile Methods

Type	Method	Definition

Transient-Variable	`fileobj(varname, selector)`	Calling a `CdmsFile` object as a function reads the region of data specified by the `selector`. The result is a transient variable, unless `raw = 1` is specified. See *Selectors*. **Example:** The following reads data for variable 'prc', year 1980:
		`>>> f = cdms.open('test.nc')`
		`>>> x = f('prc', time=('1980-1', '1981-1'))`

Variable, Axis, or Grid	`fileobj['id']`	**Get the persistent variable, axis or grid object having the string identifier.** This does not read the data for a variable. **Example: The following gets the persistent variable**
		`>>> v, equivalent to`
		`>>> v = f.variables['prc']`
		`>>> f = cdms.open('sample.nc')`
		`>>> v = f['prc']`
		Example: The following gets the axis named time, equivalent to
		`>>> t = f.axes['time']`
		`>>> t = f['time']`

None	`close()`	Close the file.
Axis	`copyAxis(axis, newname=None)`	**Copy `axis` values and attributes to a new axis in the file.** The returned object is persistent: it can be used to write axis data to or read axis data from the file. • If an axis already exists in the file, having the same name and coordinate values, it is returned. • It is an error if an axis of the same name exists, but with different coordinate values. • `axis` is the axis object to be copied. • `newname`, if specified, is the string identifier of the new axis object. • If not specified, the identifier of the input axis is used.

13.6.4 CdmsFile Methods(cont'd)

Type	Method	Definition
Grid	`copyGrid(grid, newname=None)`	**Copy grid values and attributes to a new grid in the file. The returned grid is persistent.** • If a grid already exists in the file, having the same name and axes, it is returned. • An error is raised if a grid of the same name exists, having different axes. • `grid` is the grid object to be copied. • `newname`, if specified is the string identifier of the new grid object. • If unspecified, the identifier of the input grid is used.
Axis	`createAxis(id, ar, unlimited=0)`	**Create a new `Axis`.** This is a persistent object which can be used to read or write axis data to the file. • `id` is an alphanumeric string identifier, containing no blanks. • `ar` is the one-dimensional axis array. • Set `unlimited` to `cdms.Unlimited` to indicate that the axis is extensible.
RectGrid	`createRectGrid (id,lat, lon,order, type='generic', mask=None)`	Create a `RectGrid` in the file. This is not a persistent object: the order, type, and mask are not written to the file. However, the grid may be used for regridding operations. • `lat` is a latitude axis in the file. • `lon` is a longitude axis in the file. • `order` is a string with value `'yx'` (the latitude) or `'xy'` (the first grid dimension is longitude). • `type` is one of `'gaussian'`,`'uniform'`,`'equalarea'` , or `'generic'`. • If specified, `mask` is a two-dimensional, logical Numpy array (all values are zero or one) with the same shape as the grid.

13.6.5 CdmsFile Methods(cont'd)

Type	Method	Definition
Variable	`createVariable (Stringid,String datatype,Listaxes, fill_value=None)`	**Create a new Variable.** This is a persistent object which can be used to read or write variable data to the file. • `id` is a String name which is unique with respect to all other objects in the file. • `datatype` is an `MV2` typecode, e.g., `MV2.Float`, `MV2.Int`. • `axes` is a list of Axis and/or Grid objects. • `fill_value` is the missing value (optional).
Variable	`createVariableCopy (var, newname=None)`	**Create a new `Variable`, with the same name, axes, and** attributes as the input variable. An error is raised if a variable of the same name exists in the file. • `var` is the `Variable` to be copied. • `newname`, if specified is the name of the new variable. • If unspecified, the returned variable has the same name as `var`. **Note:** Unlike copyAxis, the actual data is not copied to the new variable.
CurveGrid or Generic-Grid	`readScripGrid (self,whichGrid= 'destination', check-Grid=1)`	**Read a curvilinear or generic grid from a SCRIP netCDF file.** The file can be a SCRIP grid file or remapping file. • If a mapping file, `whichGrid` chooses the grid to read, either `'source'` or `'destination'`. • If `checkGrid` is `1` (default), the grid cells are checked for convexity, and 'repaired' if necessary. • Grid cells may appear to be nonconvex if they cross a `0 / 2pi` boundary. • The repair consists of shifting the cell vertices to the same side modulo 360 degrees.

13.6.6 CdmsFile Methods(cont'd)

Type	Method	Definition
None	`sync()`	Writes any pending changes to the file.
Variable	`write(var, attributes=None, axes=None, extbounds=None, id=None, extend=None, fill_value=None, index=None, typecode=None)`	**Write a variable or array to the file.** **The return value is the associated file variable.** • If the variable does not exist in the file, it is first defined and all attributes written, then the data is written. • By default, the time dimension of the variable is defined as the unlimited dimension of the file. • If the data is already defined, then data is extended or overwritten depending on the value of keywords `extend` and `index`, and the unlimited dimension values associated with `var`. • `var` is a Variable, masked array, or Numpy array. • `attributes` is the attribute dictionary for the variable. The default is `var.attributes`. • `axes` is the list of file axes comprising the domain of the variable. • The default is to copy `var.getAxisList()`. • `extbounds` is the unlimited dimension bounds. Defaults to `var.getAxis(0).getBounds()`. • `id` is the variable name in the file. Default is `var.id`. • `extend = 1` causes the first dimension to be unlimited: iteratively writeable. • The default is `None`, in which case the first dimension is extensible if it is `time.Set` to `0` to turn off this behaviour. • `fill_value` is the missing value flag. • `index` is the extended dimension index to write to. The default index is determined by lookup relative to the existing extended dimension. **Note:** Data can also be written by setting a slice of a file variable, and attributes can be written by setting an attribute of a file variable.

13.6.7 CDMS Datatypes

CDMS Datatype	Definition
CdChar	character
CdDouble	double-precision floating-point
CdFloat	floating-point
CdInt	integer
CdLong	long integer
CdShort	short integer

13.7 Dataset

A Dataset is a virtual file. It consists of a metafile, in CDML/XML representation, and one or more data files.

As of CDMS V3, the legacy cuDataset interface is supported by Datasets. See "cu Module".

13.7.1 Dataset Internal Attributes

Type	Name	Description
Dictionary	`attributes`	Dataset external attributes.
Dictionary	`axes`	Axes contained in the dataset.
String	`datapath`	Path of data files, relative to the parent database. If no parent, the datapath is absolute.
Dictionary	`grids`	Grids contained in the dataset.
String	`mode`	Open mode.
Database	`parent`	Database which contains this dataset. If the dataset is not part of a database, the value is `None`.
String	`uri`	Uniform Resource Identifier of this dataset.
Dictionary	`variables`	Variables contained in the dataset.
Dictionary	`xlinks`	External links contained in the dataset.

13.7.2 Dataset Constructors

Constructor	Description
`datasetobj = cdms.open(String uri, String mode='r')`	Open the dataset specified by the Universal Resource Indicator, a CDML file. Returns a Dataset object. mode is one of the indicators listed in *Open Modes* . `openDataset` is a synonym for `open`

13.7.3 Open Modes

Mode	Definition
'r'	read-only
'r+'	read-write
'a'	read-write. Open the file if it exists, otherwise create a new file
'w'	Create a new file, read-write

13.7.4 Dataset Methods

Type	Definition	Description

Transient-Variable	`datasetobj(varname, selector)`	**Calling a Dataset object as a function** reads the region of data defined by the selector. The result is a transient variable, unless `raw = 1` is specified. See 'Selectors'.
		Example: The following reads data for variable 'prc', year 1980:
		`>>> f = cdms.open('test. xml ↪')`
		`>>> x = f('prc', time=('1980-1 ↪','1981-1'))`

Variable, Axis, or Grid	`datasetobj['id']`	**The square bracket operator applied to a dataset gets** the persistent variable, axis or grid object having the string identifier. This does not read the data for a variable. Returns `None` if not found. **Example:** `>>> f = cdms.open('sampl e. ↪xml')` `>>> v = f['prc']` `* gets the persistent␣ ↪variable v, equivalent␣ ↪to ``v =f.variab les['prc ↪']``.` **Example:** `>>> t = f['time'] gets the␣ ↪axis named 'time',␣ ↪equivalent to` `>>> t = f.axes['time']`
None	`close()`	Close the dataset.
RectGrid	`createRectGrid(id, lat, lon,order, type='generic', mask=None)`	**Create a RectGrid in the dataset.** This is not a persistent object: the order, type, and mask are not written to the dataset. However, the grid may be used for regridding operations. • `lat` is a latitude axis in the dataset. • `lon` is a longitude axis in the dataset. • `order` is a string with value 'yx' (the first grid dimension is latitude) or 'xy' (the first grid dimension is longitude). • `type` is one of 'gaussian','uniform','equalarea',or 'generic' • If specified, `mask` is a two-dimensional, logical Numpy array (all values are zero or one) with the same shape as the grid.

13.7.5 Dataset Methods(Cont'd)

Type	Definition	Description
Axis	`getAxis(id)`	**Get an axis object from the file or dataset.** • `id` is the string axis identifier.
Grid	`getGrid(id)`	**Get a grid object from a file or dataset.** • `id` is the string grid identifier.
List	`getPaths()`	Get a sorted list of pathnames of datafiles which comprise the dataset. This does not include the XML metafile path, which is stored in the .uri attribute.
Variable	`getVariable(id)`	**Get a variable object from a file or dataset.** • `id` is the string variable identifier.
CurveGrid or GenericGrid	`readScripGrid(self, whichGrid='destination', check-orGeneric-Grid=1)`	**Read a curvilinear orgeneric grid from a SCRIP dataset.** • The dataset can be a SCRIP grid file or remappingfile. • If a mapping file, `whichGrid` chooses the grid to read, either `'source'` or `'destination'`. • If `checkGrid` is 1 (default), the grid cells are checked for convexity, and 'repaired' if necessary. Grid cells may appear to be nonconvex if they cross a `0 / 2pi` boundary. The repair consists of shifting the cell vertices to the same side modulo 360 degrees.
None	`sync()`	Write any pending changes to the dataset.

13.8 MV Module

The fundamental CDMS data object is the variable. A variable is comprised of:

- a masked data array, as defined in the NumPy MV2 module.
- a domain: an ordered list of axes and/or grids.
- an attribute dictionary.

The MV module is a work-alike replacement for the MV2 module, that carries along the domain and attribute information where appropriate. MV provides the same set of functions as MV2. However, MV functions generate transient variables as results. Often this simplifies scripts that perform computation. MV2 is part of the Python Numpy package, documented at http://www.numpy.org.

MV can be imported with the command:

```
>>> import MV
```

The command

```
>>> from MV import *
```

Allows use of MV commands without any prefix.

Table *Variable Constructors in module MV*, lists the constructors in MV. All functions return a transient variable. In most cases the keywords axes, attributes, and id are available. Axes is a list of axis objects which specifies the domain of the variable. Attributes is a dictionary. id is a special attribute string that serves as the identifier of the variable, and should not contain blanks or non-printing characters. It is used when the variable is plotted or written to a file. Since the id is just an attribute, it can also be set like any attribute:

```
>>> var.id = 'temperature'
```

For completeness MV provides access to all the MV2 functions. The functions not listed in the following tables are identical to the corresponding MV2 function: `allclose`, `allequal`, `common_fill_value`, `compress`, `create_mask`, `dot`, `e`, `fill_value`, `filled`, `get_print_limit`, `getmask`, `getmaskarray`, `identity`, `indices`, `innerproduct`, `isMV2`, `isMaskedArray`, `is_mask`, `isarray`, `make_mask`, `make_mask_none`, `mask_or`, `masked`, `pi`, `put`, `putmask`, `rank`, `ravel`, `set_fill_value`, `set_print_limit`, `shape`, `size`. See the documentation at http://numpy.sourceforge.net for a description of these functions.

13.8.1 Variable Constructors in Module MV

Constructor	Description
`arrayrange(start, stop=None, step=1, typecode=None, axis=None, attributes=None, id=None)`	Just like `MV2.arange()` except it returns a variable whose type can be specfied by the keyword argument typecode. The axis, attribute dictionary, and string identifier of the result variable may be specified. **Synonym:** `arange`
`masked_array(a, mask=None, fill_value=None, axes=None, attributes=None, id=None)`	Same as MV2.masked_array but creates a variable instead. If no axes are specified, the result has default axes, otherwise axes is a list of axis objects matching a.shape.
`masked_object(data,value, copy=1,savespace=0, axes=None, attributes=None, id=None)`	Create variable masked where exactly data equal to value. Create the variable with the given list of axis objects, attribute dictionary, and string id.
`masked_values(data,value, rtol=1e-05, atol=1e-08, copy=1, savespace=0, axes=None, attributes=None, id=None)`	Constructs a variable with the given list of axes and attribute dictionary, whose mask is set at those places where `abs(data - value) > atol + rtol * abs(data)`. This is a careful way of saying that those elements of the data that have value = value (to within a tolerance) are to be treated as invalid. If data is not of a floating point type, calls masked_object instead.
`ones(shape, typecode='l', savespace=0,axes=none, attributes=none, id=none)`	Return an array of all ones of the given length or shape.
`reshape(a,newshape, axes=none, attributes=none, id=none)`	Copy of a with a new shape.
`resize(a,newshape, axes=none, attributes=none, id=none)`	Return a new array with the specified shape. the original arrays total size can be any size.
`zeros(shape,typecode='l', savespace=0, axes=none, attributes=none, id=none)`	An array of all zeros of the given length or shape

The following table describes the MV non-constructor functions. with the exception of argsort, all functions return a transient variable.

13.8.2 MV Functions

Function	Description
`argsort(x, axis=-1, fill_value=None)`	Return a Numpy array of indices for sorting along a given axis.
`asarray(data, typecode=None)`	**Same as `cdms.createVariable (data, typecode, copy=0).`** • This is a short way of ensuring that something is an instance of a variable of a given type before proceeding, as in `data = asarray(data)`. • Also see the variable `astype()` function.
`average(a, axis=0, weights=None)`	**Computes the average value of the non-masked elements of x along the selected axis.** • If weights is given, it must match the size and shape of x, and the value returned is: `sum(a*weights)/sum(weights)` • In computing these sums, elements that correspond to those that are masked in x or weights are ignored.
`choose(condition, t)`	**Has a result shaped like array condition.** • `t` must be a tuple of two arrays `t1` and `t2`. • Each element of the result is the corresponding element of `t1`where `condition` is true, and the corresponding element of `t2` where `condition` is false. • The result is masked where `condition` is masked or where the selected element is masked.
`concatenate(arrays, axis=0, axisid=None, axisattributes=None)`	Concatenate the arrays along the given axis. Give the extended axis the id and attributes provided - by default, those of the first array.
`count(a, axis=None)`	Count of the non-masked elements in `a`, or along a certain axis.
`isMaskedVariable(x)`	Return true if `x` is an instance of a variable.
`masked_equal(x, value)`	`x` masked where `x` equals the scalar value. For floating point values consider `masked_values(x, value)` instead.
`masked_greater(x, value)`	`x` masked where `x > value`
`masked_greater_equal(x, value)`	`x` masked where `x >= value`
`masked_less(x, value)`	`x` masked where `x < value`
`masked_less_equal(x, value)`	`x` masked where `x &le; value`
`masked_not_equal(x, value)`	`x` masked where `x != value`
`masked_outside(x, v1, v2)`	`x` with mask of all values of `x` that are outside `[v1,v2]`
`masked_where(condition, x, copy=1)`	**Return `x` as a variable masked where condition is true.** • Also masked where `x` or `condition` masked. • `condition` is a masked array having the same shape as `x`.

13.8.3 MV Functions(cont'd)

Function	Description
`maximum(a, b=None)`	Compute the maximum valid values of `x` if `y` is `None`; with two arguments, return the element-wise larger of valid values, and mask the result where either `x` or `y` is masked.
`minimum(a, b=None)`	Compute the minimum valid values of `x` if `y` is None; with two arguments, return the element-wise smaller of valid values, and mask the result where either `x` or `y` is masked.
`outerproduct(a, b)`	Return a variable such that `result[i, j] = a[i] * b[j]`. The result will be masked where `a[i]` or `b[j]` is masked.
`power(a, b)`	`a**b`
`product(a, axis=0, fill_value=1)`	Product of elements along axis using `fill_value` for missing elements.
`repeat(ar, repeats, axis=0)`	Return `ar` repeated `repeats` times along `axis`. `repeats` is a sequence of length `ar.shape[axis]` telling how many times to repeat each element.
`set_default_fill_value (value_type, value)`	**Set the default fill value for `value_type` to `value`.** • `value_type` is a string: 'real','complex','character','integer',or 'object'. • `value` should be a scalar or single-element array.
`sort(ar, axis=-1)`	Sort array `ar` elementwise along the specified axis. The corresponding axis in the result has dummy values.
`sum(a, axis=0, fill_value=0)`	Sum of elements along a certain axis using `fill_value` for missing.
`take(a, indices, axis=0)`	Return a selection of items from `a`. See the documentation in the Numpy manual.
`transpose(ar, axes=None)`	Perform a reordering of the axes of array ar depending on the tuple of indices axes; the default is to reverse the order of the axes.
`where(condition, x, y)`	`x` where `condition` is true, `y` otherwise

13.9 HorizontalGrid

A HorizontalGrid represents a latitude-longitude coordinate system. In addition, it optionally describes how lat-lon space is partitioned into cells. Specifically, a HorizontalGrid:

- Consists of a latitude and longitude coordinate axis.
- May have associated boundary arrays describing the grid cell boundaries,
- May optionally have an associated logical mask.

CDMS supports several types of HorizontalGrids:

13.9.1 Grids

Grid Type	Definition
RectGrid	Associated latitude an longitude are 1-D axes, with strictly monotonic values.
GenericGrid	Latitude and longitude are 1-D auxiliary coordinate axis (AuxAxis1D)

13.9.2 HorizontalGrid Internal Attribute

Type	Name	Definition
Dictionary	`attributes`	External attribute dictionary.
String	`id`	The grid identifier.
Dataset or CdmsFile	`parent`	The dataset or file which contains the grid.
Tuple	`shape`	The shape of the grid, a 2-tuple

13.9.3 RectGrid Constructors

Constructor	Description
`cdms.createRectGrid(lat, lon, order, type='generic', mask=None)`	Create a grid not associated with a file or dataset. See *A First Example*
`CdmsFile.createRectGrid(id, lat, lon, order, type='generic', mask=None)`	Create a grid associated with a file. See *CdmsFile Constructors*
`Dataset.createRectGrid(id, lat, lon, order, type='generic', mask=None)`	Create a grid associated with a dataset. See *Dataset Constructors*
`cdms.createGaussianGrid(nlats, xorigin=0.0, order='yx')`	See *A First Example*
`cdms.createGenericGrid(latArray, lonArray, latBounds=None, lonBounds=None, order='yx', mask=None)`	See *A First Example*
`cdms.createGlobalMeanGrid(grid)`	See *A First Example*
`cdms.createRectGrid(lat, lon, order, type='generic', mask=None)`	See *A First Example*
`cdms.createUniformGrid(startLat, nlat, deltaLat, startLon, nlon, deltaLon, order='yx', mask=None)`	See *A First Example*
`cdms.createZonalGrid(grid)`	See *A First Example*

13.9.4 HorizontalGrid Methods

Type	Method	Description
Horizontal-Grid	`clone()`	Return a transient copy of the grid.
Axis	`getAxis(Integer n)`	Get the n-th axis.n is either 0 or 1.
Tuple	`getBounds()`	**Get the grid boundary arrays. Returns a tuple** `(latitudeArray, longitudeArray)`, where latitudeArray is a Numpy array of latitude bounds, and similarly for longitudeArray. The shape of latitudeArray and longitudeArray **depend on the type of grid:** • For rectangular grids with shape (nlat, nlon), the boundary arrays have shape (nlat,2) and (nlon,2). • For curvilinear grids with shape (nx, ny), the boundary arrays each have shape (nx, ny, 4). • For generic grids with shape (ncell,), the boundary arrays each have shape (ncell, nvert) where nvert is the maximum number of vertices per cell. • For rectilinear grids: If no boundary arrays are explicitly defined (in the file or dataset), the result depends on the autoBounds mode (see `cdms.setAutoBounds`) and the grid classification mode (see `cdms.setClassifyGrids`). By default, autoBounds mode is enabled, in which case the boundary arrays are generated based on the type of grid. • If disabled, the return value is (None,None).For rectilinear grids: • The grid classification mode specifies how the grid type is to be determined. • By default, the grid type (Gaussian, uniform, etc.) is determined by calling grid.classifyInFamily. • If the mode is 'off' grid.getType is used instead.
Axis	`getLatitude()`	Get the latitude axis of this grid.
Axis	`getLongitude()`	Get the latitude axis of this grid.

13.9.5 HorizontalGrid Methods(cont'd)

Type	Method	Description
Axis	`getMask()`	**Get the mask array of this grid, if any.** • Returns a 2-D Numpy array, having the same shape as the grid. • If the mask is not explicitly defined, the return value is `None`.
Axis	`getMesh(self, transpose=None)`	**Generate a mesh array for the meshfill graphics method.** • If transpose is defined to a tuple, say (1,0), first transpose latbounds and lonbounds according to the tuple, in this case (1,0,2).
None	`setBounds (latBounds, lonBounds, persistent=0)`	**Set the grid boundaries.** • `latBounds` is a NumPy array of shape (n,2), such that the boundaries of the kth axis value are `[latBounds[k,0],latBou nds[k,1] ]`. • `lonBounds` is defined similarly for the longitude array. **Note:** By default, the boundaries are not written to the file or dataset containing the grid (if any). This allows bounds to be set on read-only files, for regridding. If the optional argument `persistent` is set to the boundary array is written to the file.
None	`setMask(mask, persistent=0)`	**Set the grid mask.** • If `persistent == 1`, the mask values are written to the associated file, if any. • Otherwise, the mask is associated with the grid, but no I/O is generated. • `mask` is a two-dimensional, Boolean-valued Numpy array, having the same shape as the grid.

13.9.6 HorizontalGrid Methods(cont'd)

Type	Method	Description
Horizontal-Grid	`subGridRegion (latInterval, lonInterval)`	**Create a new grid corresponding to the coordinate region defined by `latInterval, lonInterv al.`** • `latInterval` and `lonInterval` are the coordinate intervals for latitude and longitude, respectively. • Each interval is a tuple having one of the forms: • `(x,y)` • `(x,y,indicator)` • `(x,y,indicator,cycle)` • `None` Where `x` and `y` are coordinates indicating the interval `[x,y)`, and: • `indicator` is a two-character string, where the first character is 'c' if the interval is closed on the left, 'o' if open, and the second character has the same meaning for the right-hand point. (Default: 'co'). • If `cycle` is specified, the axis is treated as circular with the given cycle value. • By default, if `grid. isCircular()` is true, the axis is treated as circular with a default value of 360.0. • An interval of `None` returns the full index interval of the axis. • If a mask is defined, the subgrid also has a mask corresponding to the index ranges. **Note:** The result grid is not associated with any file or dataset.
Transient-CurveGrid	`toCurveGrid (gridid=None)`	**Convert to a curvilinear grid.** • If the grid is already curvilinear, a copy of the grid object is returned. • `gridid` is the string identifier of the resulting curvilinear grid object. • If unspecified, the grid ID is copied. **Note: This method does not apply to generic grids.** • Transient-GenericGrid `toGenericGrid(gridid=None)` Convert to a generic grid. • If the grid is already generic, a copy of the grid is returned. • `gridid` is the string identifier of the resulting curvilinear grid object. • If unspecified, the grid ID is copied.

13.9.7 RectGrid Methods, Additional to HorizontalGrid Methods

Type	Method	Description
String	`getOrder()`	**Get the grid ordering, either 'yx' if latitude is the first axis,** **or 'xy' if longitude is the first axis.** **String `getType()`** • Get the grid type, either 'gaussian', 'uniform', 'equalarea', or 'generic'. • (Array,Array) `getWeights()` • Get the normalized area weight arrays, as a tuple `(latWeights, lonWeights)`. • It is assumed that the latitude and longitude axes are defined in degrees. **The latitude weights are defined as:** • `latWeights[i] = 0.5 * abs(sin(latBounds[i+1]) - sin(latBounds[i]))` **The longitude weights are defined as:** • `lonWeights[i] = abs(lonBounds[i+1] - lonBounds [i])/360.0` For a global grid, the weight arrays are normalized such that the sum of the weights is 1.0 **Example:** • Generate the 2-D weights array, such that `weights[i.j]` is the fractional area of grid zone `[i,j]`. • From cdms import MV • latwts, lonwts = gri d.getWeights() • weights = MV.outerproduct(latwts, lonwts) • Also see the function `area_weights` in module `pcmdi.weighting`.
None	`setType (gridtype)`	**Set the grid type.** • `gridtype` is one of 'gaussian', 'uniform', 'equalarea', or 'generic'.

13.9.8 RectGrid Methods, Additional to HorizontalGrid Methods(cont'd)

Type	Method	Description
RectGrid	`subGrid ((latStart, latStop), (lonStart, lonStop))`	**Create a new grid, with latitude index range '' [latStart** [latStop] and longitude index range [lonStart][lonStop]. Either index range can also be specified as None, indicating that the entire range of the latitude or longitude is used.] **Example:** • This creates newgrid corresponding to all latitudes and index range [lonStart:lonStop] of oldgrid. • `newgrid = oldgrid.subGrid(None, (lonStart, lon Stop))` • If a mask is defined, the subgrid also has a mask corresponding to the index ranges. **Note:** The result grid is not associated with any file or dataset.
RectGrid	`transpose()`	**Create a new grid, with axis order reversed. The grid mask is also transposed. Note:** The result grid is not associated with any file or dataset.

13.10 Variable

A Variable is a multidimensional data object, consisting of:

- A multidimensional data array, possibly masked,
- A collection of attributes
- A domain, an ordered tuple of CoordinateAxis objects.

A Variable which is contained in a Dataset or CdmsFile is called a persistent variable. Setting a slice of a persistent Variable writes data to the Dataset or file, and referencing a Variable slice reads data from the Dataset. Variables may also be transient, not associated with a Dataset or CdmsFile.

Variables support arithmetic operations, including the basic Python operators (+,,*,/,**, abs, and sqrt), as well as the operations defined in the MV module. The result of an arithmetic operation is a transient variable, that is, the axis information is transferred to the result.

The methods subRegion and subSlice return transient variables. In addition, a transient variable may be created with the cdms.createVariable method. The vcs and regrid module methods take advantage of the attribute, domain, and mask information in a transient variable.

13.10.1 Variable Internal Attributes

Type	Name	Definition
Dictionary	`attributes`	External attribute dictionary.
String	`id`	Variable identifier.
String	`name_in_file`	The name of the variable in the file or files which represent the dataset. If different from id, the variable is aliased.
Dataset or CdmsFile	`parent`	The dataset or file which contains the variable.
Tuple	`shape`	The length of each axis of the variable

13.10.2 Variable Constructors

Constructor	Description
`Dataset.createVariable(String id, String datatype, List axes)`	Create a Variable in a Dataset. This function is not yet implemented.
`CdmsFile.createVariable(String id, String datatype, List axes or Grids)`	**Create a Variable in a CdmsFile.** • `id` is the name of the variable. • `datatype` is the MV2 or Numpy	typecode, for example, MV2.Float. • `axesOrGrids` is a list of Axis and/or Grid objects, on which the variable is defined. Specifying a rectilinear grid is equivalent to listing the grid latitude and longitude axes, in the order defined for the grid. **Note:** This argument can either be a list or a tuple. If the tuple form is used, and there is only one element, it must have a following comma, e.g.: `(axisobj,)`.
`cdms.createVariable(array, typecode=None, copy=0, savespace=0, mask=None, fill_value=None, grid=None, axes=None, attributes=None, id=None)`	**Create a transient variable, not associated with a file or dataset.** • `array` is the data values: a Variable, masked array, or Numpy array. • `typecode` is the MV2 typecode of the array. Defaults to the typecode of array. • `copy` is an integer flag: if 1, the variable is created with a copy of the array, if 0 the variable data is shared with array. • `savespace` is an integer flag: if set to 1, internal Numpy operations will attempt to avoid silent upcasting. • `mask` is an array of integers with value 0 or 1, having the same shape as array. array elements with a corresponding mask value of 1 are considered invalid, and are not used for subsequent Numpy operations. The default mask is obtained from array if present, otherwise is None. • `fill_value` is the missing value flag. The default is obtained from array if possible, otherwise is set to 1.0e20 for floating point variables, 0 for integer-valued variables. • `grid` is a rectilinear grid object. • `axes` is a tuple of axis objects. By default the axes are obtained from array if present. Otherwise for a dimension of length n, the default axis has values [0., 1., . . . , double(n)]. • `attributes` is a dictionary of attribute values. The dictionary keys must be strings. By default the dictionary is obtained from array if present, otherwise is empty. • `id` is the string identifier of the variable. By default the id is obtained from array if possible, otherwise is set to 'variable_n' for some integer.

13.10.3 Variable Methods

Type	Method	Definition
Variable	`tvar = var[ i:j, m:n]`	**Read a slice of data from the file or dataset, resulting in a transient variable.** • Singleton dimensions are 'squeezed' out. • Data is returned in the physical ordering defined in the dataset. • The forms of the slice operator are listed in *Variable Slice Operators*
None	`var[ i:j, m:n] = array`	**Write a slice of data to the external dataset.** • The forms of the slice operator are listed in *Result Entry Methods* . (Variables in CdmsFiles only)
Variable	`tvar = var(selector)`	**Calling a variable as a function reads the region of data defined by the selector.** • The result is a transient variable, unless raw=1 keyword is specified. • See *Selectors'* .
None	`assignValue(Array ar)`	Write the entire data array. Equivalent to `var[:] = ar`. (Variables in CdmsFiles only).
Variable	`astype(typecode)`	**Cast the variable to a new datatype.** • Typecodes are as for MV, MV2, and Numpy modules.
Variable	`clone(copyData=1)`	**Return a copy of a transient variable.** • If copyData is 1 (the default) the variable data is copied as well. • If copyData is 0, the result transient variable shares the original transient variables data array.

13.10.4 Variable Methods(cont'd)

Type	Method	Definition
Transient Variable	`crossSectionRegrid (newLevel, newLatitude, method='log', missing=None, order=None)`	**Return a lat/level vertical** cross-section regridded to a new set of latitudes newLatitude and levels newLevel. • The variable should be a function of latitude, level, and (optionally) time. • `newLevel` is an axis of the result pressure levels. • `newLatitude` is an axis of the result latitudes. • `method` is optional, either 'log' to interpolate in the log of pressure (default), or 'linear' for linear interpolation. • `missing` is a missing data value. The default is `var.getMissing()` • `order` is an order string such as 'tzy' or 'zy'. The default is `var.getOrder()`. • See also: `regrid`, `pressureRegrid`.
Axis	`getAxis(n)`	**Get the n-th axis.** • `n` is an integer.
List	`getAxisIds()`	Get a list of axis identifiers.
Integer	`getAxisIndex (axis_spec)`	**Return the index of the axis specificed by axis_spec. Return -1 if no match.** • `axis_spec` is a specification as defined for getAxisList
List	`getAxisList (axes=None, omit=None, order=None)`	**Get an ordered list of axis objects in the domain of the variable.** • If `axes` is not `None`, include only certain axes. Otherwise axes is a list of specifications as described below. Axes are returned in the order specified unless the order keyword is given. • If `omit` is not `None`, omit those specified by an integer dimension number. Otherwise omit is a list of specifications as described below. • `order` is an optional string determining the output order.

13.10.5 Variable Methods(cont'd)

Type	Method	Definition
List(cont'd)	`getAxisList (axes=None, omit=None, order=None)`	Specifications for the axes or omit keywords are a list, each element having one of the following forms: • An integer dimension index, starting at 0. • A string representing an axis id or one of the strings 'time', 'latitude', 'lat', 'longitude', 'lon', 'lev' or 'level'. • A function that takes an axis as an argument and returns a value. If the value returned is true, the axis matches. • an axis object; will match if it is the same object as axis. • `order` can be a string containing the characters t,x,y,z, or * . • If a dash ('-') is given, any elements of the result not chosen otherwise are filled in from left to right with remaining candidates.
List	`getAxisListIndex (axes=None, omit=None, order=None)`	Return a list of indices of axis objects. Arguments are as for getAxisList.
List	`getDomain()`	**Get the domain.** Each element of the list is itself a tuple of the form `(axis,start,length,true_length)` • Where axis is an axis object, • Start is the start index of the domain relative to the axis object, • Length is the length of the axis, and true_length is the actual number of (defined) points in the domain. • See also: `getAxisList`.
Horizontal-Grid	`getGrid()`	Return the associated grid, or `None` if the variable is not gridded.
Axis	`getLatitude()`	Get the latitude axis, or `None` if not found.
Axis	`getLevel()`	Get the vertical level axis, or `None` if not found.
Axis	`getLongitude()`	Get the longitude axis, or `None` if not found.

13.10.6 Variable Methods(cont'd)

Type	Method	Definition
Various	`getMissing()`	Get the missing data value, or `None` if not found.
String	`getOrder()`	**Get the order string of a spatio-temporal variable.** • The order string specifies the physical ordering of the data. • It is a string of characters with length equal to the rank of the variable, indicating the order of the variable's time, level, latitude, and/or longitude axes. **Each character is one of:** • 't': time • 'z': vertical level • 'y': latitude • 'x': longitude • '-': the axis is not spatio-temporal. **Example:** A variable with ordering 'tzyx' is 4-dimensional, where the ordering of axes is (time, level, latitude, longitude). **Note:** The order string is of the form required for the order argument of a regridder function. • `intervals` is a list of scalars, 2-tuples representing [i,j), slices, and/or Ellipses. • If no `argument(s)` are present, all file paths associated with the variable are returned. • Returns a list of tuples of the form (path,slicetuple), where path is the path of a file, and slicetuple is itself a tuple of slices, of the same length as the rank of the variable, representing the portion of the variable in the file corresponding to intervals. **Note:** This function is not defined for transient various.
Axis	`getTime()`	Get the time axis, or `None` if not found.
List	`getPaths(*intervals)`	Get the file paths associated with the index region specified by intervals.

13.10.7 Variable Methods(cont'd)

Type	Method	Definition
Integer	`len(var)`	**The length of the first dimension of the variable.** If the variable is zero-dimensional (scalar), a length of 0 is returned. **Note:** `size()` returns the total number of elements.
Transient Variable	`pressureRegrid (newLevel, method='log', missin=None, order=None)`	**Return the variable regridded to a new set of** pressure levels newLevel. The variable must be a function of latitude, longitude, pressure level, and (optionally) time. • `newLevel` is an axis of the result pressure levels. • `method` is optional, either 'log' to interpolate in the log of pressure (default), or 'linear' for linear interpolation. • `missing` is a missing data value. The default is `var.getMissing()` • `order` is an order string such as 'tzyx' or 'zyx'. The default is `var.getOrder()` • See also: `regrid`, `crossSectionRegrid`.
Integer	`rank()`	The number of dimensions of the variable.
Transient	`regrid (togrid, missing=None, order=None, Variable mask=None)`	**Return the variable regridded to the horizontal grid togrid.** • `missing` is a Float specifying the missing data value. The default is 1.0e20. • `order` is a string indicating the order of dimensions of the array. It has the form returned from `variable.getOrder()`. • For example, the string 'tzyx' indicates that the dimension order of array is (time, level, latitude, longitude). • If unspecified, the function assumes that the last two dimensions of array match the input grid. • `mask` is a Numpy array, of datatype Integer or Float, consisting of ones and zeros. A value of 0 or 0.0 indicates that the corresponding data value is to be ignored for purposes of regridding. • If mask is two-dimensional of the same shape as the input grid, it overrides the mask of the input grid.

13.10.8 Variable Methods(cont'd)

Type	Method	Definition
Transient(cont'd)	`regrid (togrid, missing=None, order=None, Variable mask=None)`	**Return the variable regridded to the horizontal grid togrid.** • If the mask has more than two dimensions, it must have the same shape as array. In this case, the missing data value is also ignored. • Such an n-dimensional mask is useful if the pattern of missing data varies with level (e.g., ocean data) or time. **Note:** If neither missing or mask is set, the default mask is obtained from the mask of the array if any. See also: `crossSectionRegrid`, `pressureRegrid`.
None	`setAxis(n, axis)`	Set the n-th axis (0-origin index) of to a copy of axis.
None	`setAxisList (axislist)`	Set all axes of the variable. axislist is a list of axis objects.
None	`setMissing(value)`	Set the missing value. Integer `size()` Number of elements of the variable.
Variable	`subRegion (*region, time=None, level=None, latitude=None, longitude=None, squeeze=0, raw=0)`	**Read a coordinate region of data, returning a** transient variable. A region is a hyperrectangle in coordinate space. • `region` is an argument list, each item of which specifies an interval of a coordinate axis. The intervals are listed in the order of the variable axes. • If trailing dimensions are omitted, all values of those dimensions are retrieved. • If an axis is circular (axis.isCircular() is true) or cycle is specified (see below), then data will be read with wraparound in that dimension. • Only one axis may be read with wraparound. • A coordinate interval has one of the forms listed in *Index and Coordinate Intervals* . • Also see `axis.mapIntervalExt`.

13.10.9 Variable Methods(cont'd)

Type	Method	Definition
Variable(cont'd)	`subRegion (*region, time=None, level=None, latitude=None, longitude=None, squeeze=0, raw=0)`	Read a coordinate region of data, returning a transient variable. **A region is a hyperrectangle in coordinate space.** • The optional keyword arguments `time`, `level`, `latitude`, and `longitude` may also be used to specify the dimension for which the interval applies. This is particularly useful if the order of dimensions is not known in advance. • An exception is raised if a keyword argument conflicts with a positional region argument. • The optional keyword argument `squeeze` determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 0, and such dimensions are not 'squeezed out'. • The optional keyword argument `raw` specifies whether the return object is a variable or a masked array. • By default, a transient variable is returned, having the axes and attributes corresponding to2,3 the region read. If raw=1, an MV2 masked array is returned, equivalent to the transient variable without the axis and attribute information.

13.10.10 Variable Methods(cont'd)

Type	Method	Definition
Variable	`subSlice (*specs, time=None, level=None, latitude=None, longitude=None, squeeze=0, raw=0)`	**Read a slice of data, returning a transient variable.** This is a functional form of the slice operator [] with the squeeze option turned off. • `specs` is an argument list, each element of which specifies a slice of the corresponding dimension. There can be zero or more positional arguments, each of the form: • A single integer n, meaning `slice(n, n+1)` • An instance of the slice class • A tuple, which will be used as arguments to create a slice • ':', which means a slice covering that entire dimension • Ellipsis (...), which means to fill the slice list with ':' leaving only enough room at the end for the remaining positional arguments • A Python slice object, of the form `slice(i, j, k)` • If there are fewer slices than corresponding dimensions, all values of the trailing dimensions are read. • The keyword arguments are defined as in subRegion. • There must be no conflict between the positional arguments and the keywords. • In `(a)-(c)` and (f), negative numbers are treated as offsets from the end of that dimension, as in normal Python indexing. • String `typecode()` The Numpy datatype identifier.

13.10.11 Example Get a Region of Data

Variable ta is a function of (time, latitude, longitude). Read data corresponding to all times, latitudes -45.0 up to but not including+45.0, longitudes 0.0 through and including longitude 180.0:

```
>>> data = ta.subRegion(':', (-45.0,45.0,'co'), (0.0, 180.0))
```

or equivalently:

```
>>> data = ta.subRegion(latitude=(-45.0,45.0,'co'), longitude=(0.0,
180.0)
```

Read all data for March, 1980:

```
>>> data = ta.subRegion(time=('1980-3','1980-4','co'))
```

13.10.12 Variable Slice Operators

Operator	Description
`[i]`	The ith element, zero-origin indexing.
`[i:j]`	The ith element through, but not including, element j
`[i:]`	The ith element through the end
`[:j]`	The beginning element through, but not including, element j
`[:]`	The entire array
`[i:j:k]`	Every kth element
`[i:j, m:n]`	Multidimensional slice
`[i, ..., m]`	(Ellipsis) All dimensions between those specified.
`[-1]`	Negative indices 'wrap around'. -1 is the last element

13.10.13 Index and Coordinate Intervals

Interval Definition	Example Interval Definition	Example
`x`	Single point, such that axis[i]==x In general x is a scalar. If the axis is a time axis, x may also be a cdtime relative time type, component time type, or string of the form 'yyyy-mm-dd hh:mi:ss' (where trailing fields of the string may be omitted.	**`180.0`** `cdtime.reltime(48,'hour s since 1980-1') '1980-1-3'`
`(x,y)`	Indices i such that x axis[i] y	`(-180,180)`
`(x,y,'co')`	`x axis[i] < y`. The third item is defined as in mapInterval.	`(-90,90,'cc')`
`(x,y,'co', cycle)`	`x axis[i]< y`, with wraparound	**`( 180, 180, 'co', 360.0)`** **Note:** It is not necesary to specify the cycle of a circular longitude axis, that is, for which `axis.isCircular()` is true.
`slice(i,j, k)`	Slice object, equivalent to i:j:k in a slice operator. Refers to the indices i, i+k, i+2k, ... up to but not including index j. If i is not specified or is None it defaults to 0. If j is not specified or is None it defaults to the length of the axis. The stride k defaults to 1. k may be negative.	**`slice(1,10)`** `slice(,,-1)` reverses the direction of the axis.
`':'`	All axis values of one dimension	
`Ellipsis`	All values of all intermediate axes	

13.11 Selectors

A selector is a specification of a region of data to be selected from a variable. For example, the statement:

```
>>> x = v(time='1979-1-1', level=(1000.0,100.0))
```

Means 'select the values of variable v for time '1979-1-1' and levels 1000.0 to 100.0 inclusive, setting x to the result.' Selectors are generally used to represent regions of space and time.

The form for using a selector is:

```
>>> result = v(s)
```

Where v is a variable and s is the selector. An equivalent form is:

```
>>> result = f('varid', s)
```

Where f is a file or dataset, and 'varid' is the string ID of a variable.

A selector consists of a list of selector components. For example, the selector:

```
>>> time='1979-1-1', level=(1000.0,100.0)
```

Has two components: time='1979-1-1', and level=(1000.0,100.0). This illustrates that selector components can be defined with keywords, using the form:

```
>>> keyword=value
```

Note that for the keywords time, level, latitude, and longitude, the selector can be used with any variable. If the corresponding axis is not found, the selector component is ignored. This is very useful for writing general purpose scripts. The required keyword overrides this behavior. These keywords take values that are coordinate ranges or index ranges as defined in See *Index and Coordinate Intervals*.

The following keywords are available: Another form of selector components is the positional form, where the component order corresponds to the axis order of a variable. For example:

13.11.1 Selector Keywords

Keyword	Description	Value
axisid	Restrict the axis with ID axisid to a value or range of values.	See *Index and Coordinate Intervals*
grid	Regrid the result to the grid.	Grid object
latitude	Restrict latitude values to a value or range. Short form: lat	See *Index and Coordinate Intervals*
level	Restrict vertical levels to a value or range. Short form: lev	See *Index and Coordinate Intervals*
longitude	Restrict longitude values to a value or range. Short form: lon	See *Index and Coordinate Intervals*
order	Reorder the result.	Order string, e.g., 'tzyx'
raw	Return a masked array (MV2.array) rather than a transient variable.	0: return a transient variable (default); =1: return a masked array.
required	Require that the axis IDs be present.	List of axis identifiers.
squeeze	Remove singleton dimensions from the result.	0: leave singleton dimensions (default); 1: remove singleton dimensions.
time	Restrict time values to a value or range.	See *Index and Coordinate Intervals*

Another form of selector components is the positional form, where the component order corresponds to the axis order of a variable. For example:

```
>>> x9 = hus(('1979-1-1','1979-2-1'),1000.0)
```

Reads data for the range ('1979-1-1','1979-2-1') of the first axis, and coordinate value 1000.0 of the second axis. Non-keyword arguments of the form(s) listed in *Index and Coordinate Intervals* are treated as positional. Such selectors are more concise, but not as general or flexible as the other types described in this section.

Selectors are objects in their own right. This means that a selector can be defined and reused, independent of a particular variable. Selectors are constructed using the cdms.selectors.Selector class. The constructor takes an argument list of selector components. For example:

```
>>> from cdms.selectors import Selector
>>> sel = Selector(time=('1979-1-1','1979-2-1'), level=1000.)
>>> x1 = v1(sel)
>>> x2 = v2(sel)

>>> from cdms.selectors import Selector
>>> sel = Selector(time=('1979-1-1','1979-2-1'), level=1000.)
>>> x1 = v1(sel)
>>> x2 = v2(sel)
```

For convenience CDMS provides several predefined selectors, which can be used directly or can be combined into more complex selectors. The selectors time, level, latitude, longitude, and required are equivalent to their keyword counterparts. For example:

```
>>> from cdms import time, level
>>> x = hus(time('1979-1-1','1979-2-1'), level(1000.))
```

and

Are equivalent. Additionally, the predefined selectors `latitudeslice`, `longitudeslice`, `levelslice`, and `timeslice` take arguments `(startindex, stopindex[, stride])`:

```
>>> from cdms import timeslice, levelslice
>>> x = v(timeslice(0,2), levelslice(16,17))
```

Finally, a collection of selectors is defined in module cdutil.region:

```
>>> from cdutil.region import *
>>> NH=NorthernHemisphere=domain(latitude=(0.,90.)
>>> SH=SouthernHemisphere=domain(latitude=(-90.,0.))
>>> Tropics=domain(latitude=(-23.4,23.4))
>>> NPZ=AZ=ArcticZone=domain(latitude=(66.6,90.))
>>> SPZ=AAZ=AntarcticZone=domain(latitude=(-90.,-66.6))
```

Selectors can be combined using the & operator, or by refining them in the call:

```
>>> from cdms.selectors import Selector
>>> from cdms import level
>>> sel2 = Selector(time=('1979-1-1','1979-2-1'))
>>> sel3 = sel2 & level(1000.0)
>>> x1 = hus(sel3)
>>> x2 = hus(sel2, level=1000.0)
```

13.11.2 Selector Examples

CDMS provides a variety of ways to select or slice data. In the following examples, variable hus is contained in file sample.nc, and is a function of (time, level, latitude, longitude). Time values are monthly starting at 1979-1-1. There

are 17 levels, the last level being 1000.0. The name of the vertical level axis is 'plev'. All the examples select the first two times and the last level. The last two examples remove the singleton level dimension from the result array.

```
>>> import cdms
>>> f = cdms.open('sample.nc')
>>> hus = f.variables['hus']
>>>
>>> # Keyword selection
>>> x = hus(time=('1979-1-1','1979-2-1'), level=1000.)
>>>
>>> # Interval indicator (see mapIntervalExt)
>>> x = hus(time=('1979-1-1','1979-3-1','co'), level=1000.)
>>>
>>> # Axis ID (plev) as a keyword
>>> x = hus(time=('1979-1-1','1979-2-1'), plev=1000.)
>>>
>>> # Positional
>>> x9 = hus(('1979-1-1','1979-2-1'),1000.0)
>>>
>>> # Predefined selectors
>>> from cdms import time, level
>>> x = hus(time('1979-1-1','1979-2-1'), level(1000.))
>>>
>>> from cdms import timeslice, levelslice
>>> x = hus(timeslice(0,2), levelslice(16,17))
>>>
>>> # Call file as a function
>>> x = f('hus', time=('1979-1-1','1979-2-1'), level=1000.)
>>>
>>> # Python slices
>>> x = hus(time=slice(0,2), level=slice(16,17))
>>>
>>> # Selector objects
>>> from cdms.selectors import Selector
>>> sel = Selector(time=('1979-1-1','1979-2-1'), level=1000.)
>>> x = hus(sel)
>>>
>>> sel2 = Selector(time=('1979-1-1','1979-2-1'))
>>> sel3 = sel2 & level(1000.0)
>>> x = hus(sel3)
>>> x = hus(sel2, level=1000.0)
>>>
>>> # Squeeze singleton dimension (level)
>>> x = hus[0:2,16]
>>> x = hus(time=('1979-1-1','1979-2-1'), level=1000., squeeze=1)
>>>
>>> f.close()
```

13.12 Examples

13.12.1 Example 1

In this example, two datasets are opened, containing surface air temperature ('tas') and upper-air temperature ('ta') respectively. Surface air temperature is a function of (time, latitude, longitude). Upper-air temperature is a function of (time, level, latitude, longitude). Time is assumed to have a relative representation in the datasets (e.g., with units

'months since basetime').

Data is extracted from both datasets for January of the first input year through December of the second input year. For each time and level, three quantities are calculated: slope, variance, and correlation. The results are written to a netCDF file. For brevity, the functions `corrCoefSlope` and `removeSeasonalCycle` are omitted.

```
>>> 1.  import cdms
>>>    import MV
>>>
>>>    # Calculate variance, slope, and correlation of
>>>    # surface air temperature with upper air temperature
>>>    # by level, and save to a netCDF file. 'pathTa' is the location of
>>>    # the file containing 'ta', 'pathTas' is the file with contains 'tas'.
>>>    # Data is extracted from January of year1 through December of year2.
>>>    def ccSlopeVarianceBySeasonFiltNet(pathTa,pathTas,month1,month2):
>>>
>>>        # Open the files for ta and tas
>>>        fta = cdms.open(pathTa)
>>>        ftas = cdms.open(pathTas)
>>>
>>> 2.      #Get upper air temperature
>>>        taObj = fta['ta']
>>>        levs = taObj.getLevel()
>>>
>>>        #Get the surface temperature for the closed interval [time1,time2]
>>>        tas = ftas('tas', time=(month1,month2,'cc'))
>>>
>>>        # Allocate result arrays
>>>        newaxes = taObj.getAxisList(omit='time')
>>>        newshape = tuple([len(a) for a in newaxes])
>>>        cc = MV.zeros(newshape, typecode=MV.Float, axes=newaxes, id='correlation')
>>>        b = MV.zeros(newshape, typecode=MV.Float, axes=newaxes, id='slope')
>>>        v = MV.zeros(newshape, typecode=MV.Float, axes=newaxes, id='variance')
>>>
>>>        # Remove seasonal cycle from surface air temperature
>>>        tas = removeSeasonalCycle(tas)
>>>
>>>        # For each level of air temperature, remove seasonal cycle
>>>        # from upper air temperature, and calculate statistics
>>> 5.      for ilev in range(len(levs)):
>>>
>>>            ta = taObj(time=(month1,month2,'cc'), \
>>>                       level=slice(ilev, ilev+1), squeeze=1)
>>>            ta = removeSeasonalCycle(ta)
>>>            cc[ilev], b[ilev] = corrCoefSlope(tas ,ta)
>>>            v[ilev] = MV.sum( ta**2 )/(1.0*ta.shape[0])
>>>
>>>        # Write slope, correlation, and variance variables
>>> 6.      f = cdms.open('CC_B_V_ALL.nc','w')
>>>        f.title = filtered
>>>        f.write(b)
>>>        f.write(cc)
>>>        f.write(v)
>>>        f.close()
>>>
>>> 7.  if __name__=='__main__':
>>>        pathTa = '/pcmdi/cdms/sample/ccmSample_ta.xml'
>>>        pathTas = '/pcmdi/cdms/sample/ccmSample_tas.xml'
```

(continues on next page)

(continued from previous page)

```
>>>         # Process Jan80 through Dec81
>>>         ccSlopeVarianceBySeasonFiltNet(pathTa,pathTas,'80-1','81-12')
```

Notes:

```
1.   Two modules are imported, "cdms", and "MV". "MV" implements
     arithmetic functions.
15.  "taObj" is a file (persistent) variable. At this point, no data has
     actually been read. This happens when the file variable is sliced, or
     when the subRegion function is called. levs is an axis.
20.  Calling the file like a function reads data for the given variable
     and time range. Note that month1 and month2 are time strings.
25.  In contrast to "taObj", the variables "cc:" b" and "v" are
     transient variables, not associated with a file. The assigned names
     are used when the variables are written.
34.  Another way to read data is to call the variable as a function. The
     squeeze option removes singleton axes, in this case the level axis.
43.  Write the data. Axis information is written automatically.
50.  This is the main routine of the script. "pathTa" and "pathTas"
     pathnames. Data is processed from January 1980 through December 1981.
```

13.12.2 Example 2

In the next example, the pointwise variance of a variable over time is calculated, for all times in a dataset. The name of the dataset and variable are entered, then the variance is calculated and plotted via the vcs module.

```
>>> #!/usr/bin/env python
>>> #
>>> # Calculates gridpoint total variance
>>> # from an array of interest
>>> #
>>>
>>> import cdms
>>> from MV import *
>>>
>>> # Wait for return in an interactive window
>>>
>>> def pause():
>>>     print Hit return to continue: ,
>>>     line = sys.stdin.readline()
>>>
>>> 1.      # Calculate pointwise variance of variable over time
>>> # Returns the variance and the number of points
>>> # for which the data is defined, for each grid point
>>> def calcVar(x):
>>>     # Check that the first axis is a time axis
>>>     firstaxis = x.getAxis(0)
>>>     if not firstaxis.isTime():
>>>         raise 'First axis is not time, variable:', x.id
>>>
>>>     n = count(x,0)
>>>     sumxx = sum(x*x)
>>>     sumx = sum(x)
>>>     variance = (n*sumxx -(sumx * sumx))/(n * (n-1.))
>>>
```

(continues on next page)

(continued from previous page)

```
>>>     return variance, n
>>>
>>>   if __name__=='__main__':
>>>     import vcs, sys
>>>
>>>     print 'Enter dataset path [/pcmdi/cdms/obs/erbs_mo.xml]: ',
>>>     path = string.strip(sys.stdin.readline())
>>>     if path=='': path='/pcmdi/cdms/obs/erbs_mo.xml'
>>>
>>> 2.  # Open the dataset
>>>     dataset = cdms.open(path)
>>>
>>>     # Select a variable from the dataset
>>>     print 'Variables in file:',path
>>>     varnames = dataset.variables.keys()
>>>     varnames.sort()
>>>     for varname in varnames:
>>>
>>>         var = dataset.variables[varname]
>>>         if hasattr(var,'long_name'):
>>>             long_name = var.long_name
>>>         elif hasattr(var,'title'):
>>>             long_name = var.title
>>>         else:
>>>             long_name = '?'
>>>
>>>     print '%-10s: %s'%(varname,long_name)
>>>     print 'Select a variable: ',
>>> 3.  varname = string.strip(sys.stdin.readline())
>>>     var = dataset(varname)
>>>     dataset.close()
>>>
>>>     # Calculate variance, count, and set attributes
>>>     variance,n = calcVar(var)
>>>     variance.id = 'variance_%s'%var.id
>>>     n.id = 'count_%s'%var.id
>>>     if hasattr(var,'units'):
>>>         variance.units = '(%s)^2'%var.units
>>>
>>>     # Plot variance
>>>     w=vcs.init()
>>> 4.  w.plot(variance)
>>>     pause()
>>>     w.clear()
>>>     w.plot(n)
>>>     pause()
>>>     w.clear()
```

The result of running this script is as follows:

```
>>> % calcVar.py
>>> Enter dataset path [/pcmdi/cdms/sample/obs/erbs_mo.xml]:
>>>
>>> Variables in file: /pcmdi/cdms/sample/obs/erbs_mo.xml
>>> albt    : Albedo TOA [%]
>>> albtcs : Albedo TOA clear sky [%]
>>> rlcrft  : LW Cloud Radiation Forcing TOA [W/m^2]
```

(continues on next page)

(continued from previous page)

```
>>> rlut    : LW radiation TOA (OLR) [W/m^2]
>>> rlutcs : LW radiation upward TOA clear sky [W/m^2]
>>> rscrft : SW Cloud Radiation Forcing TOA [W/m^2]
>>> rsdt    : SW radiation downward TOA [W/m^2]
>>> rsut    : SW radiation upward TOA [W/m^2]
>>> rsutcs : SW radiation upward TOA clear sky [W/m^2]
>>> Select a variable: albt
>>>
>>> <The variance is plotted>
>>>
>>> Hit return to continue:
>>>
>>> <The number of points is plotted>
```

Notes:

1. n = count(x, 0) returns the pointwise number of valid values, summing across axis 0, the first axis. count is an MV function.
2. dataset is a Dataset or CdmsFile object, depending on whether a .xml or .nc pathname is entered. dataset.variables is a dictionary mapping variable name to file variable.
3. var is a transient variable.
4. Plot the variance and count variables. Spatial longitude and latitude information are carried with the computations, so the continents are plotted correctly.

CHAPTER

FOURTEEN

MODULE: CDTIME

14.1 Time Types

The `cdtime` module implements the CDMS time types, methods, and calendars. These are made available with the command:

```
>>> import cdtime
```

Two time types are available: relative time and component time. Relative time is time relative to a fixed base time. It consists of:

- a units string, of the form 'units since basetime', and
- a floating-point value

For example, the time "28.0 days since 1996-1-1" has value=28.0, and units='days since 1996-1-1'

Component time consists of the integer fields year, month, day, hour, minute, and the floating-point field second. A sample component time is `1996-2-28 12:10:30.0`

The `cdtime` module contains functions for converting between these forms, based on the common calendars used in climate simulation. Basic arithmetic and comparison operators are also available.

14.2 Calendars

A calendar specifies the number of days in each month, for a given year. cdtime supports these calendars:

- `cdtime.GregorianCalendar`: years evenly divisible by four are leap years, except century years not evenly divisible by 400. This is sometimes called the proleptic Gregorian calendar, meaning that the algorithm for leap years applies for all years.
- `cdtime.MixedCalendar`: mixed Julian/Gregorian calendar. Dates before 158210-15 are encoded with the Julian calendar, otherwise are encoded with the Gregorian calendar. The day immediately following 1582-10-4 is 1582-10-15. This is the default calendar.
- `cdtime.JulianCalendar`: years evenly divisible by four are leap years,
- `cdtime.NoLeapCalendar`: all years have 365 days,
- `cdtime.Calendar360`: all months have 30 days.

Several `cdtime` functions have an optional calendar argument. The default calendar is the `MixedCalendar`. The default calendar may be changed with the command:

```
cdtime.DefaultCalendar = newCalendar
```

14.3 Time Constructors

The following table describes the methods for creating time types.

14.3.1 Time Constructors

A relative time type has two members, value and units. Both can be set.

14.4 Relative Time

14.4.1 Relative Time Members

14.5 Component Time

A component time type has six members, all of which are settable.

14.5.1 Component Time

Type	Name	Summary
Integer	year	Year value
Integer	month	Month, in the range 1..12
Integer	day	Day of month, in the range 1 .. 31
Integer	hour	Hour, in the range 0 .. 23
Integer	minute	Minute, in the range 0 .. 59
Float	second	Seconds, in the range 0.0 .. 60.0

14.6 Time Methods

The following methods apply both to relative and component times.

14.6.1 Time Methods

Type	Method	Definition
Comptime or Reltime	`t.add(value, intervalUnits, cdtime.DefaultCalendar)`	**Add an interval of time to a time type t.** **Returns the same type of time.** • `value` is the Float number of interval units. • `intervalUnits` is ‘‘cdtime. • [Second (s) | Minute(s) Hour(s) | Day(s) | Week(s) | Month(s) | Season(s) | Year(s)]‘‘ • `calendar` is the calendar type.
Integer	`t.cmp(t2, cdtime.DefaultCalendar)`	**Compare time values t and t2.** Returns -1, 0, 1 as t is less than, equal to, or greater than t2 respectively. • `t2` is the time to compare. • `calendar` is the calendar type.
Comptime or Reltime	`t.sub(value, intervalUnits, cdtime.DefaultCalendar)`	**Subtract an interval of time from a time type t.** **Returns the same type of time.** • `value` is the Float number of interval units. • `intervalUnits` is cdtime.[Second (s) | Minute(s) | Hour(s) | Day(s) | Week(s) | Month(s) | Season(s) | Year(s)] • `calendar` is the calendar type.
Comptime	`t.tocomp(cdtime.DefaultCalendar)`	**Convert to component time.** **Returns the equivalent component time.** • `calendar` is the calendar type.
Reltime	`t.torel(units, cdtime.DefaultCalendar)`	**Convert to relative time.** Returns the equivalent relative time.

14.7 Examples

```
>>> import cdtime
>>> c = cdtime.comptime(1996,2,28)
>>> r = cdtime.reltime(28,"days since 1996-1-1")
>>> print r.add(1, cdtime.Day)
29.000000 days since 1996-1-1
>>> print c.add(36, cdtime.Hours)
1996-2-29 12:0:0.0
```

Note: When adding or subtracting intervals of months or years, only the month and year of the result are significant. The reason is that intervals in months/years are not commensurate with intervals in days or fractional days. This leads to results that may be surprising.

```
>>> c = comptime(1979,8,31)
>>> c.add(1, cdtime.Month)
1979-9-1 0:0:0.0
```

In other words, the day component of c was ignored in the addition, and the day/hour/minute components of the results are just the defaults. If the interval is in years, the interval is converted internally to months:

```
>>> c = comptime(1979,8,31)
>>> c.add(2, cdtime.Years)
1981-8-1 0:0:0.0
```

Compare time values.

```
>>> import cdtime
>>> r = cdtime.reltime(28,"days since 1996-1-1")
>>> c = cdtime.comptime(1996,2,28)
>>> print c.cmp(r)
1
>>> print r.cmp(c)
-1
>>> print r.cmp(r)
 1
```

Subtract an interval of time.

```
>>> import cdtime
>>> r = cdtime.reltime(28, "days since 1996-1-1")
>>> c = cdtime.comptime(1996, 2, 28)
>>> print r.sub(10, cdtime.Days)
18.000000 days since 1996-1-1
>>> print c.sub(30, cditme.Days)
1996-1-29 0:0:0.0
```

For intervals of years or months, see the **note** under add() in the example above.

Convert to component time.

```
>>> r = cdtime.reltime(28,"days since 1996-1-1")
>>> r.tocomp()
1996-1-29 0:0:0.0
```

Convert to relative time.

```
>>> c = comptime(1996,2,28)
>>> print c.torel("days since 1996-1-1")
58.000000 days since 1996-1-1
>>> r = reltime(28,"days since 1996-1-1")
>>> print r.torel("days since 1995")
393.000000 days since 1995
>>> print r.torel("days since 1995").value
393.0
```

CHAPTER

FIFTEEN

REGRIDDING DATA

15.1 Overview

CDMS provides several methods for interpolating gridded data:

- From one rectangular, lat-lon grid to another (CDMS regridder)
- Between any two lat-lon grids (SCRIP regridder)
- From one set of pressure levels to another
- From one vertical (lat/level) cross-section to another vertical cross-section.

15.2 CDMS Horizontal Regrider

The simplest method to regrid a variable from one rectangular, lat/lon grid to another is to use the regrid function defined for variables. This function takes the target grid as an argument, and returns the variable regridded to the target grid:

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/clt.nc"
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/geos5-sample.nc"
>>> import cdms2
>>> import cdat_info
>>> f1=cdms2.open("clt.nc")
>>> f2=cdms2.open("geos5-sample.nc")
>>> clt=f1('clt')  # Read the data
>>> clt.shape
(120, 46, 72)
>>> ozone=f2['ozone']  # Get the file variable (no data read)
>>> outgrid = ozone.getGrid() # Get the target grid
>>> cltnew = clt.regrid(outgrid)
>>> cltnew.shape
(120, 181, 360)
>>> outgrid.shape
(181, 360)
```

A somewhat more efficient method is to create a regridder function. This has the advantage that the mapping is created only once and can be used for multiple arrays. Also, this method can be used with data in the form of an MV2.MaskedArray. The steps in this process are:

1. Given an input grid and output grid, generate a regridder function.
2. Call the regridder function on a Numpy array, resulting in an array defined on the output grid. The regridder function can be called with any array or variable defined on the input grid.

The following example illustrates this process. The regridder function is generated at line 9, and the regridding is performed at line 10:

```
1  >>> # wget "http://cdat.llnl.gov/cdat/sample_data/clt.nc"
2  >>> # wget "http://cdat.llnl.gov/cdat/sample_data/geos5-sample.nc"
3  >>> import cdms2
4  >>> from regrid2 import Regridder
5  >>> f = cdms2.open("clt.nc")
6  >>> cltf = f['clt']
7  >>> ingrid = cltf.getGrid()
8  >>> g = cdms2.open('geos5-sample.nc')
9  >>> outgrid = g['ozone'].getGrid()
10 >>> regridfunc = Regridder(ingrid, outgrid)
11 >>> cltnew = regridfunc(cltf)
12 >>> f.close()
13 >>> g.close()
```

15.2.1 Notes

Line #3 Makes the CDMS module available.

Line #4 Makes the Regridder class available from the regrid module.

Line #5 Opens the input dataset.

Line #6 Gets the variable object named 'clt'. No data is read.

Line #7 Gets the input grid.

Line #8 Opens a dataset to retrieve the output grid.

Line #9 The output grid is the grid associated with the variable named 'ozone' in dataset g. Just the grid is retrieved, not the data.

Line #10 Generates a regridder function regridfunc.

Line #11 Reads all data for variable cltf, and calls the regridder function on that data, resulting in a transient variable cltnew.

15.3 SCRIP Horizontal Regridder

To interpolate between grids where one or both grids is non-rectangular, CDMS provides an interface to the SCRIP regridder package developed at Los Alamos National Laboratory (https://oceans11.lanl.gov/trac/SCRIP).

Figure 3 illustrates the process:

1. Obtain or generate the source and target grids in SCRIP netCDF format. A CDMS grid can be written to a netCDF file, in SCRIP format, using the write-ScripGrid method.
2. Edit the input namelist file scrip_in to reference the grids and select the method of interpolation, either conservative, bilinear, bicubic, or distance-weighted. See the SCRIP documentation for detailed instructions.
3. Run the scrip executable to generate a remapping file containing the transformation coefficients.
4. CDMS, open the remapping file and create a regridder function with the readRegridder method.
5. Call the regridder function on the input variable, defined on the source grid. The return value is the variable interpolated to the new grid. Note that the variable may have more than two dimensions. Also note that the input

arguments to the regridder function depend on the type of regridder. For example, the bicubic interpolation has additional arguments for the gradients of the variable.

15.4 Regridding Data with SCRIP

Example:

Regrid data from a T42 to POP4/3 grid, using the first-order, conservative interpolator.

In this example:

- The input grid is defined in remap_grid_T42.nc.
- The output grid is defined in remap_grid_POP43.nc.
- The input data is variable src_array in file sampleT42Grid.nc.
- The file scrip_in has contents:

```
>>> &remap_inputs
>>> num_maps = 1
>>>
>>> grid1_file = 'remap_grid_T42.nc'
>>> grid2_file = 'remap_grid_POP43.nc'
>>> interp_file1 = 'rmp_T42_to_POP43_conserv.nc'
>>> interp_file2 = 'rmp_POP43_to_T42_conserv.nc'
>>> map1_name = 'T42 to POP43 Conservative Mapping'
>>> map2_name = 'POP43 to T42 Conservative Mapping'
>>> map_method = 'conservative'
>>> normalize_opt = 'frac'
>>> output_opt = 'scrip'
>>> restrict_type = 'latitude'
>>> num_srch_bins = 90
>>> luse_grid1_area = .false.
>>> luse_grid2_area = .false.
```

`num_maps` specifies the number of mappings generated, either 1 or 2. For a single mapping, `grid1_file` and `grid2_file` are the source and target grid definitions, respectively. The `map_method` specifies the type of interpolation, either 'conservative', 'bilinear', 'bicubic', or 'distwgt' (distanceweighted). The remaining parameters are described in the SCRIP documentation.

Once the grids and input file are defined, run the scrip executable to generate the remapping file 'rmp_T42_to_POP43_conserv.nc'

```
>>> % scrip
>>> Using latitude bins to restrict search.
>>>  Computing remappings between:
>>> T42 Gaussian Grid
>>>                                    and
>>> POP 4/3 Displaced-Pole T grid
>>> grid1 sweep
>>> grid2 sweep
>>> Total number of links = 63112
```

Next, run CDAT and create the regridder:

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/remap_grid_POP43.nc"
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/remap_grid_T42.nc"
```

(continues on next page)

(continued from previous page)

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/rmp_POP43_to_T42_conserv.nc"
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/rmp_T42_to_POP43_conserv.nc"
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/xieArkin-T42.nc"
>>> # Import regrid package for regridder functions
>>> import regrid2, cdms2
>>> # Read the regridder from the remapper file
>>> remapf = cdms2.open('rmp_T42_to_POP43_conserv.nc')
>>> regridf = regrid2.readRegridder(remapf)
>>> remapf.close()
```

Then read the input data and regrid:

```
>>> # Get the source variable
>>> f = cdms2.open('xieArkin-T42.nc')
>>> t42prc = f('prc')
>>> f.close()
>>> # Regrid the source variable
>>> popdat = regridf(t42prc)
```

Note that `t42dat` can have rank greater than 2. The trailing dimensions must match the input grid shape. For example, if `t42dat` has shape (12, 64, 128), then the input grid must have shape (64,128). Similarly if the variable had a generic grid with shape (8092,), the last dimension of the variable would have length 8092.

15.5 Pressure-Level Regridder

To regrid a variable which is a function of latitude, longitude, pressure level, and (optionally) time to a new set of pressure levels, use the `pressureRegrid` function defined for variables. This function takes an axis representing the target set of pressure levels, and returns a new variable `d` regridded to that dimension.

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/ta_ncep_87-6-88-4.nc"
>>> f=cdms2.open("ta_ncep_87-6-88-4.nc")
>>> ta=f('ta')
>>> ta.shape
(11, 17, 73, 144)
>>> ta.getAxisIds()
['time', 'level', 'latitude', 'longitude']
>>> result = ta.pressureRegrid(cdms2.createAxis([1000.0]))
>>> result.shape
(11, 1, 73, 144)
```

15.6 Cross-Section Regridder

To regrid a variable which is a function of latitude, height, and (optionally) time to a new latitude/height cross-section, use the `crossSectionRegridder` defined for variables. This function takes as arguments the new latitudes and heights, and returns the variable regridded to those axes.

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/ta_ncep_87-6-88-4.nc"
>>> f=cdms2.open("ta_ncep_87-6-88-4.nc")
>>> ta=f('ta')
>>> ta.shape
(11, 17, 73, 144)
```

(continues on next page)

(continued from previous page)

```
>>> levOut=cdms2.createAxis([1000.0,950.])
>>> levOut.designateLevel()
>>> latOut=cdms2.createAxis(ta.getLatitude()[10:20])
>>> latOut.designateLatitude()
>>> ta0 = ta[0,:]
>>> ta0.getAxisIds()
['level', 'latitude', 'longitude']
>>> taout = ta0.crossSectionRegrid(levOut, latOut)
>>> taout.shape
(2, 10, 144)
```

15.7 Regrid Module

The `regrid` module implements the CDMS regridding functionality as well as the SCRIP interface. Although this module is not strictly a part of CDMS, it is designed to work with CDMS objects.

15.8 CDMS Horizontal Regridder

```
from regrid2 import Regridder
```

Makes the CDMS Regridder class available within a Python program. An instance of Regridder is a function which regrids data from rectangular input to output grids.

15.8.1 CDMS Regridder Constructor

Constructor	Description
`regridFunction = Regridder(inputGrid, outputGrid)`	**Create a regridder function which interpolates a data array from input to output grid.** • *CDMS regridder functions* describes the calling sequence of this function. • `inputGrid` and `outputGrid` are CDMS grid objects. **Note:** To set the mask associated with inputGrid or outputGrid, use the grid setMask function.

15.9 SCRIP Regridder

SCRIP regridder functions are created with the `regrid.readRegridder` function:

15.9.1 SCRIP Regridder Constructor

Constructor	Description
`regridFunction = regrid.readRegridder(fileobj, mapMethod=None, checkGrid=1)`	**Read a regridder from an open CDMS file object.** • `fileobj` is a CDMS file object, as returned from `cdms.open`. • **`mapMethod` is one of:** – `'conservative'`: conservative remapper, suitable where area-integrated fields such as water or heat fluxes must be conserved. – `'bilinear'`: bilinear interpolation – `'bicubic'`: bicubic interpolation – `'distwgt'`: distance-weighted interpolation. • It is only necessary to specify the map method if it is not defined in the file. • If `checkGrid` is 1 (default), the grid cells are checked for convexity, and 'repaired' if necessary. • Grid cells may appear to be nonconvex if they cross a `0 / 2pi` boundary. • The repair consists of shifting the cell vertices to the same side modulo 360 degrees.

15.10 Regridder Functions

It is only necessary to specify the map method if it is not defined in the file.

If `checkGrid` is 1 (default), the grid cells are checked for convexity, and 'repaired' if necessary. Grid cells may appear to be nonconvex if they cross a `0 / 2pi` boundary. The repair consists of shifting the cell vertices to the same side modulo 360 degrees.

15.11 CDMS Regridder Functions

A CDMS regridder function is an instance of the CDMS `Regridder` class. The function is associated with rectangular input and output grids. Typically its use is straightforward:

- The function is passed an input array and returns the regridded array. However, when the array has missing data, or the input and/or output grids are masked, the logic becomes more complicated.

15.11.1 Step 1

The regridder function first forms an input mask. This mask is either two-dimensional or n-dimensional, depending on the rank of the user-supplied mask. If no mask or missing value is specified, the mask is obtained from the data array mask if present.

Two-dimensional case:

- Let mask_1 be the two-dimensional user mask supplied via the mask argument, or the mask of the input grid if no user mask is specified.
- If a missing-data value is specified via the missing argument, let the implicit_mask be the two-dimensional mask defined as 0 where the first horizontal slice of the input array is missing, 1 elsewhere.
- The input mask is the logical AND(mask_1, implicit_mask)

N-dimensional case:

- If the user mask is 3 or 4-dimensional with the same shape as the input array, it is used as the input mask.

15.11.2 Step 2

The data is then regridded. In the two-dimensional case, the input mask is 'broadcast' across the other dimensions of the array. In other words, it assumes that all horizontal slices of the array have the same mask. The result is a new array, defined on the output grid. Optionally, the regridder function can also return an array having the same shape as the output array, defining the fractional area of the output array which overlaps a non-missing input grid cell. This is useful for calculating area-weighted means of masked data.

15.11.3 Step 3

Finally, if the output grid has a mask, it is applied to the result array. Where the output mask is 0, data values are set to the missing data value, or 1.0e20 if undefined. The result array or transient variable will have a mask value of 1 (invalid value) for those output grid cells which completely overlap input grid cells with missing values

15.11.4 CDMS Regridder Function

Type	Function	Description
Array or TransientVariable	`regridFunction(array, missing=None, order=None, mask=None)`	**Interpolate a gridded data array to a new grid.** The interpolation preserves the area-weighted mean on each horizontal slice. If array is a Variable, a TransientVariable of the same rank as the input array is returned, otherwise a masked array is returned. • `array` is a Variable, masked array, or Numpy array of rank 2, 3, or 4. • For example, the string 'tzyx' indicates that the dimension order of `array` is (time, level, latitude, longitude). • If unspecified, the function assumes that the last two dimensions of `array` match the input grid. • `missing` is a Float specifying the missing data value. The default is 1.0e20. • `order` is a string indicating the order of dimensions of the array. It has the form returned from `variable.getOrder()`. • `mask` is a Numpy array, of datatype Integer or Float, consisting of a fractional number between 0 and 1. • A value of 1 or 1.0 indicates that the corresponding data value is to be ignored for purposes of regridding. • A value of 0 or 0.0 indicates that the corresponding data value is valid. This is consistent with the convention for masks used by the MV2 module. • A fractional value between 0.0 and 1.0 indicates the fraction of the data value (e.g., the corresponding cell) to be ignored when regridding. This is useful if a variable is regridded first to grid A and then to another grid B; the mask when regridding from A to B would be (1.0 - f) where f is the maskArray returned from the initial grid operation using the `returnTuple` argument.

15.11.5 DMS Regridder Function(cont'd)

Type	Function	Description
Array or TransientVariable	`regridFunction (array, missing=None, order=None, mask=None)`	**Interpolate a gridded data array to a new grid.** The interpolation preserves the area-weighted mean on each horizontal slice. If array is a Variable, a TransientVariable of the same rank as the input array is returned, otherwise a masked array is returned. • If `mask` is two-dimensional of the same shape as the input grid, it overrides the mask of the input grid. • If the `mask` has more than two dimensions, it must have the same shape as `array`. In this case, the `missing` data value is also ignored. Such an ndimensional mask is useful if the pattern of missing data varies with level (e.g., ocean data) or time. **Note:** If neither `missing` or `mask` is set, the default mask is obtained from the mask of the array if any.
Array, Array	`regridFunction (ar, missing=None, order=None, mask=None, returnTuple=1)`	If called with the optional `returnTuple` argument equal to 1, the function returns a tuple `dataArray`, `maskArray`). • `dataArray` is the result data array. • `maskArray` is a Float32 array of the same shape as `dataArray`, such that `maskArray[i,j]` is fraction of the output grid cell [i,j] overlapping a non-missing cell of the grid.

15.12 SCRIP Regridder Functions

A SCRIP regridder function is an instance of the ScripRegridder class. Such a function is created by calling the regrid.readRegridder method. Typical usage is straightforward:

```
>>> import cdms2
>>> import regrid2
>>> remapf = cdms2.open('rmp_T42_to_POP43_conserv.nc')
>>> regridf = regrid2.readRegridder(remapf)
>>> f = cdms2.open('xieArkin-T42.nc')
>>> t42prc = f('prc')
>>> f.close()
>>> # Regrid the source variable
>>> popdat = regridf(t42prc)
```

The bicubic regridder takes four arguments:

```
>>> # outdat = regridf(t42prc, gradlat, gradlon, gradlatlon)
```

A regridder function also has associated methods to retrieve the following fields:

- Input grid
- Output grid
- Source fraction: the fraction of each source (input) grid cell participating in the interpolation.
- Destination fraction: the fraction of each destination (output) grid cell participating in the interpolation.

In addition, a conservative regridder has the associated grid cell areas for source and target grids.

15.12.1 SCRIP Regridder Functions

Return Type	Method	Description
Array or Transient-Variable	[conservative, bilinear, and distance-weighted regridders] `regridFunction(array)`	**Interpolate a gridded data array to a new grid. The return value is the regridded data variable.** • `array` is a Variable, MaskedArray, or Numpy array. • The rank of the array may be greater than the rank of the input grid, in which case the input grid shape must match a trailing portion of the array shape. • For example, if the input grid is curvilinear with shape (64,128), the last two dimensions of the array must match. • Similarly, if the input grid is generic with shape (2560,), the last dimension of the array must have that length.
Array or Transient-Variable	[bicubic regridders] `regridFunction(array, gradientLat, gradientLon, gradientLatLon)`	**Interpolate a gridded data array to a new grid,** using a bicubic regridder. The return value is the regridded data variable. • `array` is a Variable, MaskedArray, or Numpy array. • The rank of the array may be greater than the rank of the input grid, in which case the input grid shape must match a trailing portion of the array shape. • For example, if the input grid is curvilinear with shape (64,128), the last two dimensions of the array must match. • Simiarly, if the input grid is generic with shape (2560,), the last dimension of the array must have that length. • `gradientLat`: df/di (see the SCRIP documentation). Same shape as `array`. • `gradientLon`: df/dj. Same shape as `array`. • `gradientLatLon`: d(df)/(di)(dj). Same shape as array.

15.12.2 SCRIP Regridder Functions(con'td)

Return Type	Method	Description
Numpy array	`getDestinationArea()` [conservative regridders only]	**Return the area of the destination (output) grid cell.** • The array is 1-D, with length equal to the number of cells in the output grid.
Numpy array	`getDestination Fraction()`	**Return the area fraction of the destination (output) grid cell that participates in the regridding.** • The array is 1-D, with length equal to the number of cells in the output grid.
CurveGrid or Generic-Grid	`getInputGrid()`	Return the input grid, or None if no input grid is associated with the regridder.
CurveGrid or Generic-Grid	`getOutputGrid()`	Return the output grid.
Numpy array	`getSourceFraction()`	**Return the area fraction of the source (input) grid cell that participates in the regridding.** • The array is 1-D, with length equal to the number of cells in the input grid

15.13 Examples

15.13.1 CDMS Regridder

Example:

Regrid data to a uniform output grid.

```
1 >>> import cdms2
2 >>> from regrid2 import Regridder
3 >>> f = cdms2.open('clt.nc')
4 >>> cltf = f.variables['clt']
5 >>> ingrid = cltf.getGrid()
6 >>> outgrid = cdms2.createUniformGrid(90.0, 46, -4.0, 0.0, 72, 5.0)
7 >>> regridFunc = Regridder(ingrid, outgrid)
8 >>> newrls = regridFunc(cltf)
9 >>> f.close()
```

15.13.2 Regridder Constructure

Line	Notes
3	Open a netCDF file for input.
6	Create a 4 x 5 degree output grid. Note that this grid is not associated with a file or dataset.
7	Create the regridder function.
8	Read all data and regrid. The missing data value is obtained from variable rlsf

Return the area fraction of the source (input) grid cell that participates in the regridding. The array is 1-D, with length equal to the number of cells in the input grid.

Example:

Get a mask from a separate file, and set as the input grid mask.

```
1  >>> # wget http://cdat.llnl.gov/cdat/sample_data/clt.nc
2  >>> # wget http://cdat.llnl.gov/cdat/sample_data/geos5-sample.nc
3  >>> import cdms2
4  >>> from regrid2 import Regridder
5  >>> #
6  >>> f = cdms2.open('clt.nc')
7  >>> cltf = f.variables['clt']
8  >>> outgrid = cltf.getGrid()
9  >>> g = cdms2.open('geos5-sample.nc')
10 >>> ozoneg = g.variables['ozone']
11 >>> ingrid = ozoneg.getGrid()
12 >>> regridFunc = Regridder(ingrid,outgrid)
13 >>> uwmaskvar = g.variables['uwnd']
14 >>> uwmask = uwmaskvar[:]<0
15 >>> outArray = regridFunc(ozoneg.subSlice(time=0),mask=uwmask)
16 >>> f.close()
17 >>> g.close()
```

Line	Notes
7	Get the input grid.
10	Get the output grid.
11	Create the regridder function.
14	Get the mask.
15	Regrid with a user mask. The subslice call returns a transient variable corresponding to variables of at time 0.

Note: Although it cannot be determined from the code, both mask and the input arrays of are four-dimensional. This is the n-dimensional case.

Example:

Generate an array of zonal mean values.

```
1 >>> f = cdms.open('rls_ccc_per.nc')
2 >>> rlsf = f.variables['rls']
3 >>> ingrid = rlsf.getGrid()
4 >>> outgrid = cdms.createZonalGrid(ingrid)
5 >>> regridFunc = Regridder(ingrid,outgrid)
6 >>> mean = regridFunc(rlsf)
7 >>> f.close()
```

Line	Notes
3	Open a netCDF file for inputGet the input grid. Return the area fraction of the source (input) grid cell that participates in the regridding. The array is 1-D, with length equal to the number of cells in the input grid.
4	Create a zonal grid. outgrid has the same latitudes as ingrid, and a singleton longitude dimension. createGlobalMeanGrid could be used here to generate a global mean array.
5	Generate the regridder function.
6	Generate the zonal mean array.

Example:

Regrid an array with missing data, and calculate the area-weighted mean of the result.

```
>>> import cdms2
>>> from cdms2.MV2 import *
>>> from regrid2 import Regridder
>>> f = cdms2.open("ta_ncep_87-6-88-4.nc")
>>> var = f('ta')
>>> outgrid = cdms2.createUniformGrid(90.0, 46, -4.0, 0.0, 72, 5.0)
>>> outlatw, outlonw = outgrid.getWeights()
>>> outweights = outerproduct(outlatw, outlonw)
>>> grid = var.getGrid()
>>> sample = var[0,0]
>>> latw, lonw = grid.getWeights()
>>> weights = outerproduct(latw, lonw)
>>> inmask = where(greater(absolute(sample),1.e15),0,1)
>>> mean = add.reduce(ravel(inmask*weights*sample))/add.reduce(ravel(inmask*weights))
>>> regridFunc = Regridder(grid, outgrid)
>>> outsample, outmask = regridFunc(sample, mask=inmask, returnTuple=1)
>>> outmean = add.reduce(ravel(outmask*outweights*outsample)) / add.
↪reduce(ravel(outmask*outweights))
```

Line	Notes
2	Create a uniform target grid.
3	Get the latitude and longitude weights.
4	Generate a 2-D weights array.
5	Get the input grid. `var` is a 4-D variable.
6	Get the first horizontal slice from `var`.
7-8	Get the input weights, and generate a 2-D weights array.
9	Set the 2-D input mask.
10	Calculate the input array area-weighted mean.
11	Create the regridder function.
12	Regrid. Because returnTuple is set to 1, the result is a tuple (dataArray, maskArray).
13	Calculate the area-weighted mean of the regridded data. mean and outmean should be approximately equal.

15.13.3 SCRIP Regridder

Example:

Regrid from a curvilinear to a generic grid, using a conservative remapping. Compute the area-weighted means on input and output for comparison.

```
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/remap_grid_T42.nc"
>>> # wget http://cdat.llnl.gov/cdat/sample_data/rmp_T42_to_C02562_conserv.nc
>>> # wget "http://cdat.llnl.gov/cdat/sample_data/xieArkin-T42.nc"
>>> import cdms2, regrid2, MV2
>>> # Open the SCRIP remapping file and data file
>>> fremap = cdms2.open('rmp_T42_to_C02562_conserv.nc')
>>> fdat = cdms2.open('xieArkin-T42.nc')
>>> # Input data array
>>> dat = fdat('prc')[0,:]
>>> # Read the SCRIP regridder
>>> regridf = regrid2.readRegridder(fremap)
```

(continues on next page)

(continued from previous page)

```
>>> # Regrid the variable
>>> outdat = regridf(dat)
>>> # Get the cell area and fraction arrays. Areas are computed only
>>> # for conservative regridding.
>>> srcfrac = regridf.getSourceFraction()
>>> srcarea = regridf.getSourceArea()
>>> dstfrac = regridf.getDestinationFraction()
>>> dstarea = regridf.getDestinationArea()
>>> # calculate area-weighted means
>>> inmean = MV2.sum(srcfrac*srcarea*MV2.ravel(dat)) / MV2.sum(srcfrac*srcarea)
>>> outmean = MV2.sum(dstfrac*dstarea*MV2.ravel(outdat)) / MV2.sum(dstfrac*dstarea)
>>> print 'Input mean:', inmean
Input mean: 2.60376502339
>>> print 'Output mean:', outmean
Output mean: 2.60376502339
>>> fremap.close()
>>> fdat.close()
```

CHAPTER

SIXTEEN

PLOTTING CDMS DATA IN PYTHON

16.1 Overview

Data read via the CDMS Python interface can be plotted using the `vcs` module. This module, part of the Climate Data Analysis Tool (CDAT) is documented in the CDAT reference manual. The `vcs` module provides access to the functionality of the VCS visualization program.

Examples of plotting data accessed from CDMS are given below, as well as documentation for the plot routine keywords.

16.2 Examples

In the following examples, it is assumed that variable `psl` is dimensioned (time, latitude, longitude). `psl` is contained in the dataset named `'sample.xml'`.

16.2.1 Plotting a Gridded Variable

Example: Plotting a gridded variable

```
1 >>> import cdms2, vcs
2 >>> f = cdms2.open("clt.nc")
3 >>> clt = f.variables['clt']
4 >>> sample = clt[0,:]
5 >>> w=vcs.init()
6 >>> w.plot(sample)
7 <vcs.displayplot.Dp object ...>
8 >>> f.close()
```

Notes:

Line	Notes
3	Get a horizontal slice, for the first time point.
4	Create a VCS Canvas `w`.
5	Plot the data. Because sample is a transient variable, it encapsulates all the time, latitude, longitude, and attribute information.
7	Close the file. This must be done after the reference to the persistent variable `ps l`.

Thats it! The axis coordinates, variable name, description, units, etc. are obtained from variable sample.

What if the units are not explicitly defined for `clt`, or a different description is desired? `plot` has a number of other keywords which fill in the extra plot information.

16.2.2 Using A Plot Keywords

```
>>> import cdms2, vcs
>>> f = cdms2.open("clt.nc")
>>> clt = f.variables['clt']
>>> sample = clt[0,:]
>>> w=vcs.init()
>>> w.plot(sample, units='percent', file_comment='', long_name="Total Cloud",␣
↪comment1="Example plot", hms="00:00:00", ymd="1979/01/01")
<vcs.displayplot.Dp object ...>
>>> f.close()
```

Note: Keyword arguments can be listed in any order.

16.2.3 Plotting a Time-Latitude Slice

Assuming that variable `clt` has domain `(time,latitude,longitude)`, this example selects and plots a time-latitude slice:

```
>>> import cdms2, vcs
>>> f = cdms2.open("clt.nc")
>>> clt = f.variables['clt']
>>> samp = clt[:,:,0]
>>> w = vcs.init()
>>> w.plot(samp, name='Total Cloudiness')
<vcs.displayplot.Dp object ...>
```

Line	Notes
4	`samp` is a slice of `clt`, at index `0` of the last dimension. Since `samp` was obtained from the slice operator, it is a transient variable, which includes the latitude and time information.
6	The `name` keyword defines the identifier, default is the name found in the file.

16.2.4 Plotting Subsetted Data

Calling the variable `clt` as a function reads a subset of the variable. The result variable `samp` can be plotted directly:

```
>>> import cdms2, vcs
>>> f = cdms2.open("clt.nc")
>>> clt = f.variables['clt']
>>> samp = clt(time = (0.0,100.0), longitude = 180.0, squeeze=1)
>>> w = vcs.init()
>>> w.plot(samp)
<vcs.displayplot.Dp object ...>
>>> f.close()
```

16.3 Plot Method

The `plot` method is documented in the CDAT Reference Manual. This section augments the documentation with a description of the optional keyword arguments. The general form of the plot command is:

```
canvas.plot(array [, args] [,key=value [, key=value [, ...] ] ])
```

where:

- canvas is a VCS Canvas object, created with the vcs.init method.
- array is a variable, masked array, or Numpy array having between two and five dimensions. The last dimensions of the array is termed the 'x' dimension, the next-to-last the 'y' dimension, then 'z', 't', and 'w'.
  - For example, if array is three-dimensional, the axes are (z,y,x), and if array is four-dimensional, the axes are (t,z,y,x).

    Note: that the t dimension need have no connection with time; any spatial axis can be mapped to any plot dimension.)
  - For a graphics method which is two-dimensional, such as boxfill, the y-axis is plotted on the horizontal, and the x-axis on the vertical.
  - If array is a gridded variable on a rectangular grid, the plot function uses a box-fill graphics method.
  - If it is non-rectangular, the meshfill graphics method is used.

    Note: that some plot keywords apply only to rectangular grids only.
- args are optional positional arguments:

  `args` := template_name, graphics_method, graphics_name

  `template_name`: the name of the VCS template (e.g., 'AMIP')

  `graphics_method`: the VCS graphics method (boxfill)

  `graphics_name`: the name of the specific graphics method ('default')

  See the CDAT Reference Manual and VCS Reference Manual for a detailed description of these arguments.
- `key=value, ...` are optional keyword/value pairs, listed in any order. These are defined in the table below.

16.3.1 Plot Keywords

Key	Type	Value
comment1	string	Comment plotted above `file_comment`
comment2	string	Comment plotted above `comment1`
comment3	string	Comment plotted above `comment2`
continents	0 or 1	**if 1, plot continental outlines (default:plot if** • `xaxis` is longitude, • `yaxis` is latitude -or- `xname` is 'longitude' and `yname` is 'latitude'
file_comment	string	**Comment,** • Defaults to `variable.parent.comment`
grid	CDMS grid object	**Grid associated with the data.** • Defaults to `variable.getGrid()`
hms	string	Hour, minute, second
long_name	string	**Descriptive variable name,** • Defaults to `variable.long_name`.
missing_value	same type as array	**Missing data value,** • Defaults to `variable.getMissing()`
`name`	string	**Variable name,** • Defaults to `variable.id`
time	cdtime relative or absolute	**Time associated with the data.** **Example:** • `cdtime.reltime(30.0, 'days since 1978-1-1')`.

16.3.2 Plot Keywords(cont'd)

Key	Type	Value
units	string	**Data units.** • Defaults to `variable.units`
variable	CDMS variable object	**Variable associated with the data.** • The variable grid must have the same shape as the data array.
xarray (`[y|z|t|w]array`)	1-D Numpy array	***Rectangular grids only*.** • Array of coordinate values, having the same length as the corresponding dimension. • Defaults to `xaxis[:\] (y|z|t|waxis[:])`
xaxis (`[y|z|t|w]axis`)	CDMS axis object	***Rectangular grids only*.** **Axis object.** • `xaxis` defaults to `grid.getAxis(0)` • `yaxis` defaults to `grid.getAxis(1)`
xbounds (`ybounds`)	2-D Numpy array	***Rectangular grids only*.** • Boundary array of shape `(n,2)` where `n` is the axis length. • Defaults to `xaxis.getBounds()`, or `xaxis.genGenericBounds()` if `None`, similarly for `ybounds`.
xname (`[y|z|t|w]name`)	string	***Rectangular grids only*.** **Axis name.** • Defaults to `xaxis.id` (`[y|z|t|w]axis.id`)
xrev (`yrev`)	0 or 1	If `xrev` (`yrev`) is 1, reverse the direction of the `x-axis` (`y-axis`). • Defaults to 0, with the following exceptions: • If the `y-axis` is latitude, and has decreasing values, `yrev` defaults to 1 • If the `y-axis` is a vertical level, and has increasing pressure levels, `yrev` defaults to 1.
xunits (`[y|z|t|w]units`)	string	***Rectangular grids only*. Axis units.** • Defaults to `xaxis.units` (`[y|z|t|w]axis.units`).

CHAPTER

SEVENTEEN

CLIMATE DATA MARKUP LANGUAGE (CDML)

17.1 Introduction

The Climate Data Markup Language (CDML) is the markup language used to represent metadata in CDMS. CDML is based on the W3C XML standard (https://www.w3.org). This chapter defines the syntax of CDML. Read this section if you will be building or maintaining a CDMS database.

XML, the eXtensible Markup Language, makes it possible to define interoperable dialects of markup languages. The most recent version of HTML, the Web hypertext markup language, is an XML dialect. CDML is also an XML dialect, geared toward the representation of gridded climate datasets. XML provides rigor to the metadata representation, ensuring that applications can access it correctly. XML also deals with internationalization issues, and holds forth the promise that utilities for browsing, editing, and other common tasks will be available in the future.

CDML files have the file extension .xml or .cdml.

17.2 Elements

A CDML document consists of a nested collection of elements. An element is a description of the metadata associated with a CDMS object. The form of an element is:

```
<tag attribute-list> element-content </tag>
```

or

```
<tag attribute-list />
```

where

- `tag` is a string which defines the type of element
- `attribute-list` is a blank-separated list of attribute-value pairs, of the form:

  ```
  attribute = "value"
  ```

- `element-content` depends on the type of element. It is either a list of elements, or text which defines the element values. For example, the content of an axis element either is a list of axis values, or is a linear element.
- For datasets, the content is the blank-separated list of elements corresponding to the axes, grids, and variables contained in the dataset.

The CDML elements are:

17.2.1 CDML Tags

Tag	Description
attr	Extra attribute
axis	Coordinate axis
domain	Axes on which a variable is defined
domElem	Element of a variable domain
linear	Linearly-spaced axis values
rectGrid	Rectilinear Grid
variable	Variable

17.3 Special Characters

XML reserves certain characters for markup. If they appear as content, they must be encoded to avoid confusion with markup:

17.3.1 Special Character Encodings

Character	Encoding
<	&lt;
>	&gt;
&	&amp;
“	&quot;
‘	&apos;

For example, the comment

Certain “special characters”, such as <, >, and ‘, must be encoded.

would appear in an attribute string as:

comment = “Certain "special characters", such as <, >, and ', must be encoded.”

17.4 Identifiers

In CDMS, all objects in a dataset have a unique string identifier. The id attribute holds the value of this identifier. If the variable, axis, or grid has a string name within a data file, then the id attribute ordinarily has this value. Alternatively, the name of the object in a data file can be stored in the name_in_file attribute, which can differ from the id. Datasets also have IDs, which can be used within a larger context (databases).

An identifer must start with an alphabetic character (upper or lower case), an underscore (_), or a colon (:). Characters after the first must be alphanumeric, an underscore, or colon. There is no restriction on the length of an identifier.

17.5 CF Metadata Standard

The CF metadata standard defines a set of conventions for usage of netCDF. This standard is supported by CDML. The document defines names and usage for metadata attributes. CF supersedes the GDT 1.3 standard.

17.6 CDML Syntax

The following notation is used in this section:

- A `monospaced block` is used for a syntax specification.
- **Bold** text indicates literals.
- (R|S) denotes either R or S.
- R* denotes zero or more R.
- R+ denotes one or more R.

A CDML document consists of a prolog followed by a single dataset element.

```
CDML-document ::= prolog dataset-element
```

The prolog defines the XML version, and the Document Type Definition (DTD), a formal specification of the document syntax. See https://www.w3.org/TR/1998/REC-xml-19980210 for a formal definition of XML

Version 1.0.

```
prolog ::= <?xml version="1.0"?> <!DOCTYPE dataset SYSTEM "https://www-pcmdi.
llnl.gov/~drach/cdms/cdml.dtd">
```

17.6.1 Dataset Element

A dataset element describes a single dataset. The content is a list of elements corresponding to the axes, grids, and variables contained in the dataset. Axis, variable, and grid elements can be listed in any order, and an element ID can be used before the element is actually defined.

`dataset-element ::=` **<dataset** `dataset-attributes> dataset-content` **</dataset>**

```
dataset-content ::= (axis-element | grid-element | variable-element)*
extra-attribute-element+
```

17.6.2 Dataset Attributes

Attribute	Req?	CF	GDT	Notes
appendices	N	N	Y	Version number
calendar	N	N	Y	**Calendar used for encoding time axes.** • `gregorian` | `julian` | `noleap` |`360_day` | `proleptic_gregorian`|`standard` **Note:** for the CF convention, the calendar attribute is placed on the time axis.
comment	N	Y	Y	Additional dataset information
conventions	Y	Y	Y	The netCDF metadata standard. Example: 'CF-1.0'
cdms_filemap	Y	N	N	Map of partitioned axes to files. See note below.
directory	N	N	N	Root directory of the dataset
frequency	N	N	N	Temporal frequency
history	N	Y	Y	Evolution of the data
id	Y	N	N	Dataset identifier
institution	N	Y	Y	Who made or supplied the data
production	N	N	Y	How the data was produced (see source)
project	N	N	N	Project associated with the data Example: 'CMIP 2'
references	N	Y	N	Published or web-based references that describe the data or methods used to produce it
source	N	Y	N	The method of production of the original data.
template	N	N	N	Filename template. This is an alternate mechanism, other than cdms_filemap, for describing the file mapping. See 'cdimport -h' for details.
title	N	Y	N	A succinct description of the data.

Notes:

The `cdms_filemap` attribute describes how the dataset is partitioned into files. The format is:

- `filemap ::= [ varmap, varmap, ...]`
- `varmap ::= [ namelist, slicelist ]`
- `namelist ::= [ name, name, ... ]`
- `slicelist ::= [ indexlist, indexlist, ,,, ]`
- `indexlist ::= [ time0, time1, lev0, lev1, path ]`
- `name ::= variable name`
- `time0 ::= first index of time in the file, or '-' if not split on time`
- `time1 ::= last index of time + 1, in the file, or '-' if not split on time`
- `lev0 ::= first index of vertical levels in the file, or '-' if not split on level`
- `lev1 ::= last index +1 of vertical levels in the file, or '-' if not split on level`
- `path ::= pathname of the file containing data for this time/level range.`

The pathname is appended to the value of the directory attribute, to obtain an absolute pathname.

17.6.3 Axis Element

An axis element describes a single coordinate axis. The content can be a blank-separated list of axis values or a linear element. A linear element is a representation of a linearly-spaced axis as (start, delta, length).

`axis-element ::=` **<axis** `axis-attributes> axis-content` **</axis>**

`axis-content ::= (axis-values | linear-element) extra-attribute-element*`

`axis-values ::= [value*]`

`linear-element ::=` **<linear delta=** `"value"` **length=** `"Integer"` **start=** `"value"` **> </linear>**

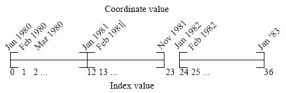
17.6.4 Axis Elements

Attribute	Req?	CF	GDT	Notes
associate	N	N	Y	IDs of variables containing alternative sets of coordinates.
axis	N	Y	Y	**The spatial type of the axis:** • 'T' - time • 'X' - longitude • 'Y' - latitude • 'Z' - vertical level • '-' - not spatiotemporal
bounds	N	Y	Y	ID of the boundary variable
calendar	N	Y	N	See dataset.calendar
climatology	N	Y	N	Range of dates to which climatological statistics apply.
comment	N	Y	N	String comment
compress	N	Y	Y	Dimensions which have been compressed by gathering
datatype	Y	N	N	Char, Short, Long, Float, Double, or String
dates	N	Y	N	Range of dates to which statistics for a typical diurnal cycle apply.
expand	N	N	Y	Coordinates prior to contraction
formula_terms	N	Y	N	Variables that correspond to the terms in a formula.
id	Y	N	N	Axis identifier. Also the name of the axis in the underlying file(s), if name_in_file is undefined.
isvar	N	N	N	**'true' | 'false'** • 'false' if the axis does not have coordinate values explicitly defined in the underlying file(s). • Default: 'true'
leap_month	N	Y	N	For a user-defined calendar, the month which is lengthened by a day in leap years.
leap_year	N	Y	N	An example of a leap year for a user-defined calendar. All years that differ from this year by a multiple of four are leap years.
length	N	N	N	Number of axis values, including values for which no data is defined. Cf. partition_length.
long_name	N	Y	Y	Long description of a physical quantity
modulo	N	N	Y	Arithmetic modulo of an axis with circular topology.
month_lengths	N	Y	N	Length of each month in a non-leap year for a user-defined calendar.
name_in_file	N	N	N	Name of the axis in the underlying file(s). See id.
partition	N	N	N	How the axis is split across files.
partition_length	N	N	N	Number of axis points for which data is actually defined. If data is missing for some values, this will be smaller than the length.
positive	N	Y	Y	Direction of positive for a vertical axis
standard_name	N	Y	N	Reference to an entry in the standard name table.
topology	N	N	Y	**Axis topology.** • 'circular' | 'linear'
units	Y	Y	Y	Units of a physical quantity
weights	N	N	N	Name of the weights array

17.6.5 Partition attribute

For an axis in a dataset, the .partition attribute describes how an axis is split across files. It is a list of the start and end indices of each axis partition.

FIGURE 4. Partitioned axis



For example, Figure 4 shows a time axis, representing the 36 months, January 1980 through December 1982, with December 1981 missing. The first partition interval is (0,12), the second is (12,23), and the third is (24,36), where the interval (i,j) represents all indices k such that i <= k < j. The .partition attribute for this axis would be the list:

```
[0, 12, 12, 23, 24, 36]
```

Note that the end index of the second interval is strictly less than the start index of the following interval. This indicates that data for that period is missing.

17.6.6 Grid Element

A grid element describes a horizontal, latitude-longitude grid which is rectilinear in topology,

`grid-element ::=` **<rectGrid** `grid-attributes>` `extra-attribute-element*` **</rectGrid>**

17.6.7 6.5 RectGrid Attributes

```
<th>Attribute</th> <th>Required?</th> <th>GDT?</th> <th>Notes</th>
```

```
<tr><td><code>id</code></td><td>Y</td><td>N</td><td>Grid identifier</td></tr>
<tr><td><code>type</code></td><td>Y</td><td>N</td><td><p>Grid classification</p><p>
↪"gaussian" | "uniform"
| "equalarea" |"generic"</p><p>Default: "generic"</p></td></tr>
<tr><td><code>latitude</code></td><td>Y</td><td>N</td><td>Latitude axis name</td></tr>
<tr><td><code>longitude</code></td><td>Y</td><td>N</td><td>Longitude axis name</td></
↪tr>
<tr><td><code>mask</code></td><td>N</td><td>N</td><td>Name of associated mask variable
↪</td></tr>
<tr><td><code>order</code></td><td>Y</td><td>N</td><td><p>Grid ordering "yx"
 | "xy"</p><p>Default: “yx”, axis order is latitude, longitude</p></td></tr>
```

17.6.8 Variable Element

A variable element describes a data variable. The domain of the variable is an ordered list of domain elements naming the axes on which the variable is defined. A domain element is a reference to an axis or grid in the dataset.

The length of a domain element is the number of axis points for which data can be retrieved. The partition_length is the number of points for which data is actually defined. If data is missing, this is less than the length.

`variable-element ::=` **<variable** `variable-attributes>` `variable-content` **</variable>**

`variable-content ::=` variable-domain extra-attributeelement*‘‘

`variable-domain ::=` **<domain>** `domain-element*` **</domain>**

domain-element ::= **<domElem name=**"axis-name"** start=**"‘‘Integer‘‘" **length=**"‘‘Integer‘‘" **partition_length=**"‘‘Integer‘‘"/>**

17.6.9 Variable Attributes

Attribute	Req?	CF	GDT	Notes
id	Y	N	N	Variable identifier. Also, the name of the variable in the underlying file(s), if name_in_file is undefined.
ad_offset	N	Y	Y	Additive offset for packing data. See scale_factor.
associate	N	N	Y	IDs of variables containing alternative sets of coordinates Spatio-temporal dimensions.
axis	N	N	Y	Example: TYX for a variable with domain (time, latitude, longitude) Note: for CF, applies to axes only.
cell_methods	N	Y	N	The method used to derive data that represents cell values, e.g., maximum,mean,variance, etc.
comments	N	N	N	Comment string
coordinates	N	Y	N	IDs of variables containing coordinate data.
datatype	Y	N	N	Char, Short, Long, Float, Double, or String
grid_name	N	N	N	Id of the grid.
grid_type	N	N	N	gaussian, uniform, equalarea, generic
long_name	N	Y	Y	Long description of a physical quantity.
missing_value	N	Y	Y	Value used for data that are unknown or missing.
name_in_file	N	N	N	Name of the variable in the underlying file(s). See id.
scale_factor	N	Y	Y	Multiplicative factor for packing data. See add_offset.
standard_name	N	Y	N	Reference to an entry in the standard name table.
subgrid	N	N	Y	Records how data values represent subgrid variation.
template	N	N	N	Name of the file template to use for this variable. Overrides the dataset value.
units	N	Y	Y	Units of a physical quantity.
valid_max	N	Y	Y	Largest valid value of a variable.
valid_min	N	Y	Y	Smallest valid value of a variable.
valid_range	N	Y	Y	Largest and smallest valid values of a variable.

17.6.10 Attribute Element

Attributes which are not explicitly defined by the GDT convention are represented as extra attribute elements. Any dataset, axis, grid, or variable element can have an extra attribute as part of its content. This representation is also useful if the attribute value has non-blank whitespace characters (carriage returns, tabs, linefeeds) which are significant.

The datatype is one of: **Char**, **Short**, **Long**, **Float**, **Double**, or **String**.

extra-attribute-element ::= **<attr name=**"attribute-name" **datatype=**"attribute-datatype"**>** attribute-value **</attr>**

17.7 A Sample CDML Document

Dataset “sample” has two variables, and six axes.

Note:

- The file is indented for readability. This is not required; the added whitespace is ignored.

- The dataset contains three axes and two variables. Variables u and v are functions of time, latitude, and longitude.
- The global attribute cdms_filemap describes the mapping between variables and files. The entry `[[u],[[0,1,-,-,u_2000.nc],[1,2,-,-,u_2001.nc],[2,3,,-,u_2002.nc] ]` indicates that variable `u` is contained in file u_2000.nc for time index 0, u_2001.nc for time index 1, etc.

```
<!DOCTYPE dataset SYSTEM "http://www-pcmdi.llnl.gov/software/cdms/cdml.dtd">

[-90. -78. -66. -54. -42. -30. -18. -6. 6. 18. 30. 42. 54. 66. 78. 90.]
```

```
<axis
    id ="longitude"
    length="32"
    units="degrees_east"
    datatype="Double"
>

    [ 0. 11.25 22.5 33.75 45. 56.25 67.5 78.75 90.

    101.25 112.5 123.75 135. 146.25 157.5 168.75 180. 191.25

    202.5 213.75 225. 236.25 247.5 258.75 270. 281.25 292.5

    303.75 315. 326.25 337.5 348.75]
    </axis>

<axis
    id ="time"
    partition="[0 1 1 2 2 3]"
    calendar="gregorian"
    units="days since 2000-1-1"
    datatype="Double"
    length="3"
    name_in_file="time"
>

   [ 0. 366. 731.]
</axis>

<variable
   id ="u"
   missing_value="-99.9"
   units="m/s"
   datatype="Double"
>
   <domain>
       <domElem name="time" length="3" start="0"/>
       <domElem name="latitude" length="16" start="0"/>
       <domElem name="longitude" length="32" start="0"/>
   </domain>
</variable>

<variable
   id ="v"
   missing_value="-99.9"
   units="m/s"
   datatype="Double"
>
```

(continues on next page)

(continued from previous page)

```
   <domain>
       <domElem name="time" length="3" start="0"/>
       <domElem name="latitude" length="16" start="0"/>
       <domElem name="longitude" length="32" start="0"/>
   </domain>
</variable>
```

CHAPTER

EIGHTEEN

CDMS UTILITIES

18.1 CdScan: Importing Datasets into CDMS

18.1.1 Overview

A dataset is a partitioned collection of files. To create a dataset, the files must be scanned to produce a text representation of the dataset. CDMS represents datasets as an ASCII metafile in the CDML markup language. The file contains all metadata, together with information describing how the dataset is partitioned into files. (Note: CDMS provides a direct interface to individual files as well. It is not necessary to scan an individual file in order to access it.)

For CDMS applications to work correctly, it is important that the CDML metafile be valid. The `cdscan` utility generates a metafile from a collection of data files.

CDMS assumes that there is some regularity in how datasets are partitioned:

- A variable can be partitioned (split across files) in at most two dimensions.
- The partitioned dimension(s) must be either time or vertical level dimensions; variables may not be partitioned across longitude or latitude.
- Datasets can be parti-tioned by variable as well.
- For example, one set of files might contain heat fluxes, while another set contains wind speeds.

Otherwise, there is considerable flexibility in how a dataset can be partitioned:

- Files can contain a single variable or all variables in the dataset.
- The time axis can have gaps.
- Horizontal grid boundary information and related information can be duplicated across files.
- Variables can be on different grids.
- Files may be in any of the self-describing formats supported by CDMS, including netCDF, HDF, GrADS/GRIB, and DRS.

18.1.2 Syntax

The syntax of the `cdscan` command is

- cdscan [options] file1 file2 ...

or

- cdscan [options] -f file_list

where

- `file1 file2 ...` is a blank-separated list of files to scan
- `file_list` is the name of a file containing a list of files to scan, one pathname per line.

Output is written to standard output by default. Use the -x option to specify an output filename.

18.1.3 CDScan Command Options

Option	Description
`-a alias_file`	**Change variable names to the aliases defined in an alias file.** **Each line of the alias file consists of two blank separated fields:** • `variable_id alias`. • `variable_id` is the ID of the variable in the file, and • `alias` is the name that will be substituted for it in the output dataset. • Only variables with entries in the `alias_file` are renamed.
`-c calendar`	**Specify the dataset calendar attribute.** One of: * gregorian (default) * julian * noleap * proleptic_gregorian * standard * 360_day
`-d dataset_id`	**String identifier of the dataset.** • Should not contain blanks or non-printing characters. • Default: 'None'
`-e newattr`	**Add or modify attributes of a file, variable, or axis.** **The form of `newattr` is either:** • `var.attr = value` to modify a variable or attribute, or • `.attr = value` to modify a global (file) attribute. • In either case, value may be quoted to preserve spaces or force the attribute to be treated as a string. • If value is not quoted and the first character is a digit, it is converted to integer or floating-point. This option does not modify the input datafiles. See notes and examples below.
`--exclude var,var,...`	**Exclude specified variables.** • The argument is a comma-separated list of variables containing no blanks. • Also see `--include`.

18.1.4 CDScan Command Options(cont'd)

Option	Description
`-f file_list`	File containing a list of absolute data file names, one per line.
`-h`	Print a help message.
`-i time_delta`	**Causes the time dimension to be represented as linear, producing a more compact representation.** • This is useful if the time dimension is very long. • `time_delta` is a float or integer. • For example, if the time delta is 6 hours, and the reference units are `hours since xxxx` , set the time delta to 6. See the `-r` option. See Note 2.
`--include var,var,...`	**Only include specified variables in the output.** • The argument is a comma-separated list of variables containing no blanks. • Also see `--exclude`.
`-j`	**Scan time as a vector dimension.** • Time values are listed individually. **Note:** Turns off the -i option.
`-l levels`	**Specify that the files are partitioned by vertical level. That is, data for different vertical levels may appear in different files.** • `levels` is a comma-separated list of levels containing no blanks. • See Note 3.
`-m levelid`	**Name of the vertical level dimension.** • The default is the vertical dimension as determined by CDMS. • See Note 3.

18.1.5 CDScan Command Options(cont'd)

Option	Description
`-p template`	**Add a file template string, for compatibility with pre-V3.0 datasets.** • `cdimport -h` describes template strings.
`-q`	Quiet mode.
`-r time_units`	**Time units of the form `units since yyyy-mm-dd hh:mi:ss`, where:** • `units` is one of 'year', 'month', 'day', 'hour', 'minute', 'second'.
`-s suffix_file`	**Append a suffix to variable names, depending on the directory containing the data file.** This can be used to distinguish variables having the same name but generated by different models or ensemble runs. • `suffix_file` is the name of a file describing a mapping between directories and suffixes. • Each line consists of two blank-separated fields: `directory suffix`. • Each file path is compared to the directories in the suffix file. • If the file path is in that directory or a subdirectory, the corresponding suffix is appended to the variable IDs in the file. • If more than one such directory is found, the first directory found is used. • If no match is made, the variable ids are not altered. Regular expressions can be used: see the example in the Notes section.
`-t timeid`	**ID of the partitioned time dimension.** • The default is the name of the time dimension as determined by CDMS. • See Note 1.
`--time-linear tzero,delta, units[, calendar]`	**Override the time dimensions(s) with a linear time dimension.** **The arguments are comma-separated list:** • zero is the initial time point, a floating-point value. • delta is the time delta, floating-point. • units are time units as specified in the [-r] option. • calendar is optional, and is specified as in the [-c] option. • If omitted, it defaults to the value specified by [-c], otherwise as specified in the file. **Example:** `--time-linear '0,1,months since 1980,noleap'`
`-x xmlfile`	Output file name. By default, output is written to standard output.

Notes:

1. Files can be in netCDF, GrADS/GRIB, HDF, or DRS format, and can be listed in any order. Most commonly, the files are the result of a single experiment, and the 'partitioned' dimension is time. The time dimension of a variable is the coordinate variable having a name that starts with 'time' or having an attribute axis='T'. If this is not the case, specify the time dimension with the -t option. The time dimension should be in the form supported by cdtime. If this is not the case (or to override them) use the -r option.

2. By default, the time values are listed explicitly in the output XML. This can cause a problem if the time dimension is very long, say for 6-hourly data. To handle this the form cdscan -i delta <files> may be used. This generates a compact time representation of the form <start, length, delta>. An exception is raised if the time dimension for a given file is not linear.

3. Another form of the command is cdscan -l lev1,lev2,..,levn <files>. This asserts that the dataset is partitioned in both time and vertical level dimensions. The level dimension of a variable is the dimension having a name that starts with "lev", or having an attribute "axis=Z". If this is not the case, set the level name with the -m option.

- **Adding or modifying attributes with the -e option:**
  - time.units = "days since 1979-1-1"
- sets the units of all variables/axes to "days since 1979-1-1". Note that since this is done before any other processing is done, it allows overriding of non-COARDS time units.
  - .newattr=newvalue
- Set the global file attribute 'newattr' to 'newvalue'.
- The `[--time-linear]` option overrides the time values in the file(s). The resulting dimension does not have any gaps. In contrast, the `[-i]`, `[-r]` options use the specified time units (from `[-r]`), and calendar from `[-c]` if specified, to convert the file times to the new units. The resulting linear dimension may have gaps.
  - In either case, the files are ordered by the time values in the files.
  - The `[--time-linear]` option should be used with caution, as it is applied to all the time dimensions found.

18.1.6 Examples

- cdscan -c noleap -d test -x test.xml [uv]*.nc
- cdscan -d pcmdi_6h -i 0.25 -r 'days since 1979-1-1' *6h*.ctl

18.1.7 File Formats

Data may be represented in a variety of self-describing binary file formats, including

- netCDF, the Unidata Network Common Data Format
- HDF, the NCSA Hierarchical Data Format
- GrADS/GRIB, WMO GRIB plus a GrADS control file (.ctl) The first non-comment line of the control file must be a dset specification.
- DRS, the PCMDI legacy format.

18.1.8 Name Aliasing

A problem can occur if variables in different files are defined on different grids. What if the axis names are the same? CDMS requires that within a dataset, axis and variable IDs (names) be unique. What should the longitude axes be named in CDMS to ensure uniqueness? The answer is to allow CDMS IDs to differ from file names.

If a variable or axis has a CDMS ID which differs from its name in the file, it is said to have an alias. The actual name of the object in the file is stored in the attribute `name_in_file`. `cdscan` uses this mechanism (with the `-a` and `s` options) to resolve name conflicts; a new axis or variable ID is generated, and the `name_in_file` is set to the axis name in the file.

Name aliases also can be used to enforce naming standards. For data received from an outside organization, variable names may not be recognized by existing applications. Often it is simpler and safer to add an alias to the metafile rather than rewrite the data

CHAPTER

NINETEEN

APPENDIX A

19.1 CDMS Classes

Figure 1, “CDMS Classes”, illustrates the class inheritance structure of CDMS. The classes may be categorized as abstract or concrete. Only concrete classes are meant to be used directly. In contrast an abstract class defines the common interface of its subclasses. For example, the class AbstractAxis2D defines the common interface for two-dimensional coordinate axes. It has concrete subclasses DatasetAxis2D, FileAxis2D, and TransientAxis2D, which are used in applications. Abstract classes are denoted in italics.

For many abstract classes there are three ‘flavors’ of subclass: dataset, file, and transient. Dataset-related objects are thought of as being contained in datasets in the sense that operations on those objects result in I/O operations on the corresponding dataset. The same is true of file-related objects. Objects in datasets and files are examples of persistent objects, whose state persists after the application exits. On the other hand, transient objects live in memory and are not persistent.

In general the concrete subclasses closely mirror the interface of the abstract parent class. For this reason this document defines the interfaces of the abstract classes, and only discusses a concrete class in the few cases where the interface has been extended. This allows applications to treat the behavior of, say a dataset axis and file axis, as identical.

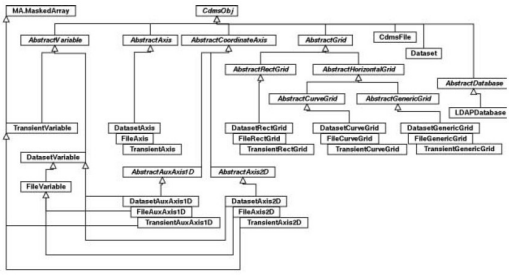


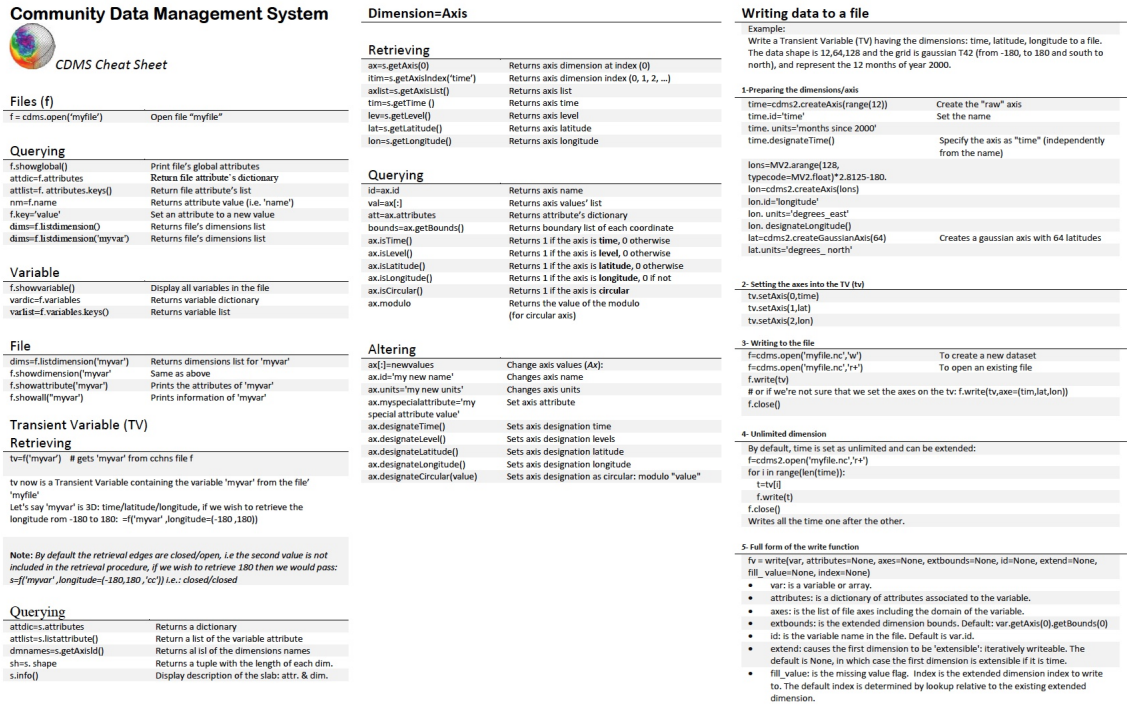
FIGURE 1. CDMS Classes

CHAPTER

TWENTY

APPENDIX B

20.1 Quick Start (Cheat Sheet)



Community Data Management System

CDMS Cheat Sheet

Files (f)

f = cdms.open('myfile')	Open file "myfile"

Querying

f.showglobal()	Print file's global attributes
attdic=f.attributes	Return file attribute`s dictionary
attlist=f. attributes.keys()	Return file attribute's list
nm=f.name	Returns attribute value (i.e. 'name')
f.key='value'	Set an attribute to a new value
dims=f.listdimension()	Returns file's dimensions list
dims=f.listdimension('myvar')	Returns file's dimensions list

Variable

f.showvariable()	Display all variables in the file
vardic=f.variables	Returns variable dictionary
varlist=f.variables.keys()	Returns variable list

File

dims=f.listdimension('myvar')	Returns dimensions list for 'myvar'
f.showdimension('myvar'	Same as above
f.showattribute('myvar')	Prints the attributes of 'myvar'
f.showall("myvar')	Prints information of 'myvar'

Transient Variable (TV)

Retrieving

tv=f('myvar') # gets 'myvar' from cchns file f

tv now is a Transient Variable containing the variable 'myvar' from the file' 'myfile'
Let's say 'myvar' is 3D: time/latitude/longitude, if we wish to retrieve the longitude rom -180 to 180: =f('myvar' ,longitude=(-180 ,180))

Note: *By default the retrieval edges are closed/open, i.e the second value is not included in the retrieval procedure, if we wish to retrieve 180 then we would pass: s=f('myvar' ,longitude=(-180,180 ,'cc')) i.e.: closed/closed*

Querying

attdic=s.attributes	Returns a dictionary
attlist=s.listattribute()	Return a list of the variable attribute
dmnames=s.getAxisId()	Returns al isl of the dimensions names
sh=s. shape	Returns a tuple with the length of each dim.
s.info()	Display description of the slab: attr. & dim.

Dimension=Axis

Retrieving

ax=s.getAxis(0)	Returns axis dimension at index (0)
itim=s.getAxisIndex('time')	Returns axis dimension index (0, 1, 2, ...)
axlist=s.getAxisList()	Returns axis list
tim=s.getTime ()	Returns axis time
lev=s.getLevel()	Returns axis level
lat=s.getLatitude()	Returns axis latitude
lon=s.getLongitude()	Returns axis longitude

Querying

id=ax.id	Returns axis name
val=ax[:]	Returns axis values' list
att=ax.attributes	Returns attribute's dictionary
bounds=ax.getBounds()	Returns boundary list of each coordinate
ax.isTime()	Returns 1 if the axis is **time,** 0 otherwise
ax.isLevel()	Returns 1 if the axis is **level,** 0 otherwise
ax.isLatitude()	Returns 1 if the axis is **latitude,** 0 otherwise
ax.isLongitude()	Returns 1 if the axis is **longitude,** 0 if not
ax.isCircular()	Returns 1 if the axis is **circular**
ax.modulo	Returns the value of the modulo (for circular axis)

Altering

ax[:]=newvalues	Change axis values (*Ax*):
ax.id='my new name'	Changes axis name
ax.units='my new units'	Changes axis units
ax.myspecialattribute='my special attribute value'	Set axis attribute
ax.designateTime()	Sets axis designation time
ax.designateLevel()	Sets axis designation levels
ax.designateLatitude()	Sets axis designation latitude
ax.designateLongitude()	Sets axis designation longitude
ax.designateCircular(value)	Sets axis designation as circular: modulo "value"

Writing data to a file

Example:
Write a Transient Variable (TV) having the dimensions: time, latitude, longitude to a file. The data shape is 12,64,128 and the grid is gaussian T42 (from -180, to 180 and south to north), and represent the 12 months of year 2000.

1-Preparing the dimensions/axis

time=cdms2.createAxis(range(12))	Create the "raw" axis
time.id='time'	Set the name
time. units='months since 2000'	
time.designateTime()	Specify the axis as "time" (independently from the name)
lons=MV2.arange(128, typecode=MV2.float)*2.8125-180.	
lon=cdms2.createAxis(lons)	
lon.id='longitude'	
lon. units='degrees_east'	
lon. designateLongitude()	
lat=cdms2.createGaussianAxis(64)	Creates a gaussian axis with 64 latitudes
lat.units='degrees_ north'	

2- Setting the axes into the TV (tv)

tv.setAxis(0,time)
tv.setAxis(1,lat)
tv.setAxis(2,lon)

3- Writing to the file

f=cdms.open('myfile.nc','w')	To create a new dataset
f=cdms.open('myfile.nc','r+')	To open an existing file

f.write(tv)
# or if we're not sure that we set the axes on the tv: f.write(tv,axe=(tim,lat,lon))
f.close()

4- Unlimited dimension

By default, time is set as unlimited and can be extended:
f=cdms2.open('myfile.nc','r+')
for i in range(len(time)):
t=tv[i]
f.write(t)
f.close()
Writes all the time one after the other.

5- Full form of the write function

fv = write(var, attributes=None, axes=None, extbounds=None, id=None, extend=None, fill_value=None, index=None)

- var: is a variable or array.
- attributes: is a dictionary of attributes associated to the variable.
- axes: is the list of file axes including the domain of the variable.
- extbounds: is the extended dimension bounds. Default: var.getAxis(0).getBounds(0)
- id: is the variable name in the file. Default is var.id.
- extend: causes the first dimension to be 'extensible': iteratively writeable. The default is None, in which case the first dimension is extensible if it is time.
- fill_value: is the missing value flag. Index is the extended dimension index to write to. The default index is determined by lookup relative to the existing extended dimension.

```
cdms quick start
```

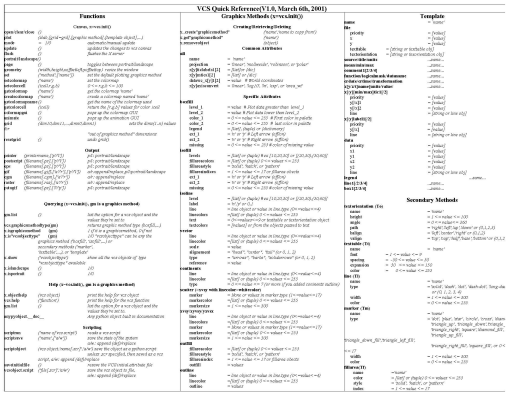
20.2 VCS Quick Reference (Cheat Sheet)

20.3 Release Notes

20.3.1 Release 4.0

CDMS version 4.0 adds support for nonrectangular grids:

- **The following grid classes were added:**



  - AbstractHorizontalGrid
  - AbstractCurve-Grid
  - AbstractGenericGrid
  - DatasetCurveGrid
  - FileCurveGrid
  - TransientCurve-Grid
  - DatasetGenericGrid
  - FileGenericGrid
  - TransientGenericGrid.

- **The following axis classes were added:**
  - AbstractCoordinateAxis
  - AbstractAuxAxis1D
  - AbstractAxis2D
  - DatasetAuxAxis1D
  - FileAuxAxis1D,
  - TransientAuxAxis1D
  - DatasetAxis2D
  - FileAxis2D
  - TransientAxis2D.
- The getMesh and clone methods were added for grids.
- An interface to the SCRIP package was added.

20.3.2 Release 3.0 Overview

CDMS version 3.0 is a significant enhancement of previous versions. The major changes were:

- CDAT/CDMS was integrated with the Numpyal Python masked array class MV2.MaskedVariable. The MV submodule was added as a wrapper around MV.

- Methods that read data, such as subRegion, subSlice, and the slice operations, return instances of class TransientVariable. The plot and regrid modules were modified to handle masked array input. The specifiers time=..., latitude=..., etc. were added to the I/O routines.
- The class TransientVariable was added.
- A number of new functions were added, notably subRegion and subSlice, which return instances of TransientVariable.
- When a masked array is returned from a method, it is "squeezed": singleton dimensions are removed. In contrast, transient variables are not squeezed. I/O functions have a squeeze option. The method setAutoReshapeMode was removed.
- Internal attributes are handled in the InternalAttributes class. This allows CDMS classes to be subclassed more readily.
- The class Variable was renamed DatasetVariable.
- The cu module was emulated in cdms. cu and cdms methods can be mixed.
- The code was modularized, so that Python, CDMS, and Numpyal Python can be built and installed separately. This significantly enhances the portability of the code.

20.3.3 Details

AbstractVariable

- The functions getRegion, getSlice, getValue, and the slice operators all return an instance of MV, a masked array. Singleton dimensions are squeezed.
- The functions subRegion and subSlice return an instance of TransientVariable. Singleton dimensions are not squeezed.
- The xxSlice and xxRegion functions have keywords time, level, latitude, and longitude.
- The input functions have the keyword squeeze.
- AbstractVariable inherits from class Slab. The following functions previously available in module cu are Slab methods:
  - getattribute
  - setattribute
  - listdimattributes
  - getdimattribute
  - listall, and info
- AbstractVariable implements arithmetic functions, astype.
- The write function was added.

AbstractAxis

- subaxis was renamed subAxis for consistency.
- Generalized wraparound was implemented, to handle multiple cycles, reversing, and negative strides. By default, coordinate intervals are closed. The intersection options 'n','e','b',and 's' were added to the interval indicator - see mapIntervalExt.

AbstractDatabase

- The function open is synonymous with openDataset.

Dataset

- The function open is synonymous with openDataset.

Cdms Module

- **The following functions were added:**
  - asVariable
  - isVariable
  - createVariable
- The function setAutoReshapeMode was removed. It is replaced by the squeeze option for all I/O functions.

CdmsFile

- The function createVariable has a keyword fill_value. The datatype may be a Numpy/MV typecode.
- The function write was added.

CDMSError

- All errors are an instance of the class CDMSError.

AbstractRectGrid

- The function createGaussianGrid was added.

InternalAttributes

- The class InternalAttributes was added.
- It has methods:
  - add_internal_attribute
  - is_internal_attribute
  - replace_external_attributes

TransientVariable

- The class TransientVariable was added. It inherits from both AbstractVariable and MV.
- The cdms module function createVariable returns a transient variable.
- This class does not implement the functions getPaths or getTemplate.

MV

- The MV submodule of cdms was added.

CHAPTER

TWENTYONE

APPENDIX C

21.1 Module *cu*

The `cu` module is the original CDAT I/O interface. As of version 3 it is emulated in the `cdms` module. It is maintained for backward compatibility.

The `cu` classes are `Slab`, corresponding to `TransientVariable` in CDMS, and `cuDataset`, corresponding to `Dataset` in CDMS.

21.2 Slab

21.2.1 Table Slab Methods

Type	Method	Definition
Various	`getdimattribute(dim, field)`	**Get the value of a dimension attribute.** • `dim` is the dimension number, an integer in the range 0..rank- 1. • `field` is a string, one of: 'name', 'values', 'length', 'units', 'weights', 'bounds'.
Various	`getattribute(name)`	**Get the value of an attribute.** • `name` is the string name of the attribute. **The following special names can always be used:** • `filename`, `comments`, `grid_name`, `grid_type`, `time_statistic`, `long_name`, `units`.
None	`info(flag=None, device=sys.stdout)`	**Print slab information.** • If `flag` is nonzero, dimension values, weights, and bounds are also printed. Output is sent to `device`.
List	`listall(all=None)`	**Print slab information.** • If `all` is nonzero, dimension values, weights, and bounds are also printed.
List	`listdimattributes(dim, field)`	**List dimension attributes. Returns a list of string** attribute names which can be input to `getdimattribute`. • `dim` is the dimension number, an integer in the range 0..rank-1. • `field` is a string, one of: 'name', 'values', 'length', 'units', 'weights', 'bounds'.
None	`setattribute(name, value)`	**Set an attribute.** • `name` is the string name of the attribute. • `value` is the value of the attribute.

21.3 cuDataset

21.3.1 Table cuDataset Methods

Type	Method	Definition
None	`cleardefault()`	Clear the default variable name.
None	`default_variable(vname)`	**Set the default variable name.** • `vname` is the string variable name.
Array	`dimensionarray(dname, vname=None)`	**Values of the axis named dname.** • `dname1` is the string axis name. • `vname` is the string variable name. **Note**: The default is the variable name set by default_variable.
Axis	`dimensionobject(dname, vname=None)`	**Get an axis. dname is the string name of an axis.** • `vname` is a string variable name. **Note:** The default is the variable name set by default_variable.
Various	`getattribute (vname, attribute)`	**Get an attribute value.** • `vname` is a string variable name. • `attribute` is the string attribute name.
String	`getdimensionunits (dname,vname=None)`	**Get the units for the given dimension.** • `dname` is the string name of an axis. • `vname` is a string variable name. **Note:** The default is the variable name set by default_variable.
Various	`getglobal (attribute)`	**Get the value of the global attribute.** • `attribute` is the string attribute name.

21.3.2 Table cuDataset Methods(cont'd)

Type	Method	Definition
Variable	`getslab (vname, \*args)`	**Read data for a variable.** • `vname` is the string name of the variable. • `args` is an argument list corresponding to the dimensions of the variable. **Arguments for each dimension can be:** • ':' or None – select the entire dimension • Ellipsis – select entire dimensions between the ones given. • a pair of successive arguments giving an interval in world coordinates. • a CDMS-style tuple of world coordinates e.g. (start, stop, 'cc')
List	`listall (vname=None, all=None)`	**Get info about data from the file.** • `vname` is the string name of the variable. **Note:** If all is non-zero, dimension values, weights, and bounds are returned as well
List	`listattribute (vname=None )`	**Return a list of attribute names.** • `vname` is the name of the variable. **Note:** The default is the variable name set by default_variable.
List	`listdimension (vname=None)`	**Return a list of the dimension names associated with a variable.** • `vname` is the name of the variable. **Note:** The default is the variable name set by default_variable.
List	`listglobal ()`	Return a list of the global attribute names.
List	`listvariable ()`	Return a list of the variables in the file.

21.3.3 Table cuDataset Methods(cont'd)

Type	Method	Definition
None	`showall (vname=None, all=None, device=sys.stdout)`	**Print a description of the variable.** • `vname` is the string name of the variable. **Note:** If all is non-zero, dimension values, weights, and bounds are returned as well. Output is sent to device.
None	`showattribute (vname=None, device=sys.stdout)`	**Print the attributes of a variable.** • `vname` is the string name of the variable. **Note:** Output is sent to device.
None	`showdimension (vname=None, device=sys.stdout)`	**Print the dimension names associated with a variable.** • `vname` is the string name of the variable. **Note:** Output is sent to device.
None	`showglobal (device=sys.stdout)`	Print the global file attributes. Output is sent to device.
None	`showvariable (device=sys.stdout)`	Print the list of variables in the file.

CHAPTER

TWENTYTWO

CDMS SAMPLE DATASET

22.1 THREDDS Data Server (TDS)

- https://aims3.llnl.gov/thredds/catalog/esgcet/254/CDAT-sample.v1.html

22.2 Direct https donwload

- https://cdat.llnl.gov/cdat/sample_data/161122_RobertPincus_multiple_input4MIPs_radiation_RFMIP_UColorado-RFMIP-20161122_none.nc
- https://cdat.llnl.gov/cdat/sample_data/20160520.A_WCYCL1850.ne30_oEC.edison.alpha6_01_ANN_climo_Q.nc
- https://cdat.llnl.gov/cdat/sample_data/area_weights.nc
- https://cdat.llnl.gov/cdat/sample_data/BlueMarble.ppm
- https://cdat.llnl.gov/cdat/sample_data/CCTM_ACONC.D1.001
- https://cdat.llnl.gov/cdat/sample_data/CCTM_ACONC.D1.002
- https://cdat.llnl.gov/cdat/sample_data/CCTM_ACONC.D1.003
- https://cdat.llnl.gov/cdat/sample_data/CCTM_ACONC.D1.004
- https://cdat.llnl.gov/cdat/sample_data/CCTM_ACONC.D1.005
- https://cdat.llnl.gov/cdat/sample_data/CCTM_CONC.D1.001
- https://cdat.llnl.gov/cdat/sample_data/cdtest10.xml
- https://cdat.llnl.gov/cdat/sample_data/cdtest13.xml
- https://cdat.llnl.gov/cdat/sample_data/cdtest14.xml
- https://cdat.llnl.gov/cdat/sample_data/clt2.nc
- https://cdat.llnl.gov/cdat/sample_data/clt.nc
- https://cdat.llnl.gov/cdat/sample_data/dar.meteo.mod.cam3.era.v31.h0.l3.nrstpt.cp.2000070100.2000080100.tau.12.36.nc
- https://cdat.llnl.gov/cdat/sample_data/dvtest1.dat
- https://cdat.llnl.gov/cdat/sample_data/dvtest1.dic
- https://cdat.llnl.gov/cdat/sample_data/era15_tas_sample.nc
- https://cdat.llnl.gov/cdat/sample_data/era40_sst_sample.nc

- https://cdat.llnl.gov/cdat/sample_data/era40_tas_sample.nc
- https://cdat.llnl.gov/cdat/sample_data/genutil_statistics.nc
- https://cdat.llnl.gov/cdat/sample_data/geo.1deg.ctl
- https://cdat.llnl.gov/cdat/sample_data/geo.1deg.gmp
- https://cdat.llnl.gov/cdat/sample_data/geo.1deg.grb
- https://cdat.llnl.gov/cdat/sample_data/geos5-sample.nc
- https://cdat.llnl.gov/cdat/sample_data/gfs.globl.b20121028_00z.e20121102_00z.ctl
- https://cdat.llnl.gov/cdat/sample_data/gfs.globl.b20121028_00z.e20121102_00z.data
- https://cdat.llnl.gov/cdat/sample_data/GPCP_ANN_climo.nc
- https://cdat.llnl.gov/cdat/sample_data/gpcp_cdr_v23rB1_y2016_m08.nc
- https://cdat.llnl.gov/cdat/sample_data/GRIDCRO2D_D1.001
- https://cdat.llnl.gov/cdat/sample_data/hadcrut2_sample.nc
- https://cdat.llnl.gov/cdat/sample_data/junk.nc
- https://cdat.llnl.gov/cdat/sample_data/MERRA_ANN_climo_SHUM.nc
- https://cdat.llnl.gov/cdat/sample_data/meshfill.nc
- https://cdat.llnl.gov/cdat/sample_data/model_ANN_climo.nc
- https://cdat.llnl.gov/cdat/sample_data/navy_land.nc
- https://cdat.llnl.gov/cdat/sample_data/netcdf4_compressed_example.nc
- https://cdat.llnl.gov/cdat/sample_data/obs_timeseries.nc
- https://cdat.llnl.gov/cdat/sample_data/prcp_1951.nc
- https://cdat.llnl.gov/cdat/sample_data/psl_6h.nc
- https://cdat.llnl.gov/cdat/sample_data/ps.wrap.test.0E.nc
- https://cdat.llnl.gov/cdat/sample_data/ps.wrap.test.180E.nc
- https://cdat.llnl.gov/cdat/sample_data/ps.wrap.test.180W.nc
- https://cdat.llnl.gov/cdat/sample_data/ps.wrap.test.60E.nc
- https://cdat.llnl.gov/cdat/sample_data/readonly.nc
- https://cdat.llnl.gov/cdat/sample_data/remap_grid_POP43.nc
- https://cdat.llnl.gov/cdat/sample_data/remap_grid_T42.nc
- https://cdat.llnl.gov/cdat/sample_data/rlut_CERES_000001-000012_ac.nc
- https://cdat.llnl.gov/cdat/sample_data/rmp_C02562_to_T42_conserv.nc
- https://cdat.llnl.gov/cdat/sample_data/rmp_POP43_to_T42_conserv.nc
- https://cdat.llnl.gov/cdat/sample_data/rmp_T42_to_C02562_conserv.nc
- https://cdat.llnl.gov/cdat/sample_data/rmp_T42_to_POP43_conserv.nc
- https://cdat.llnl.gov/cdat/sample_data/sampleCurveGrid4.nc
- https://cdat.llnl.gov/cdat/sample_data/sampleGenGrid3.nc
- https://cdat.llnl.gov/cdat/sample_data/sample.nc

- https://cdat.llnl.gov/cdat/sample_data/sample_polar.nc
- https://cdat.llnl.gov/cdat/sample_data/sftbyrgn.nc
- https://cdat.llnl.gov/cdat/sample_data/sftlf_10x10.nc
- https://cdat.llnl.gov/cdat/sample_data/sftlf_ccsr.nc
- https://cdat.llnl.gov/cdat/sample_data/sftlf_dnm.nc
- https://cdat.llnl.gov/cdat/sample_data/sftlf_visus.nc
- https://cdat.llnl.gov/cdat/sample_data/so_Omon_ACCESS1-0_historical_r1i1p1_185001-185412_2timesteps.nc
- https://cdat.llnl.gov/cdat/sample_data/so_Omon_GISS-E2-R_historicalNat_r5i1p1_185001-187512_2timesteps.nc
- https://cdat.llnl.gov/cdat/sample_data/so_Omon_HadGEM2-CC_historical_r1i1p1_185912-186911_2timesteps.nc
- https://cdat.llnl.gov/cdat/sample_data/so_Omon_MPI-ESM-LR_1pctCO2_r1i1p1_185001-185912_2timesteps.nc
- https://cdat.llnl.gov/cdat/sample_data/stereographic.nc
- https://cdat.llnl.gov/cdat/sample_data/swan.four.nc
- https://cdat.llnl.gov/cdat/sample_data/ta_300_850_PCM_O_mm_xy_wa_r0000_0000.dat
- https://cdat.llnl.gov/cdat/sample_data/ta_300_850_PCM_O_mm_xy_wa_r0000_0000.dic
- https://cdat.llnl.gov/cdat/sample_data/ta_ncep_87-6-88-4.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_6h.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ccsr-95a_1979.01-1979.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ccsr-95a_1980.01-1980.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ccsr-95a_1981.01-1981.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ccsr-95a_1982.01-1982.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ccsr-95a_1983.01-1983.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ccsr-95a_1984.01-1984.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ccsr-95a.xml
- https://cdat.llnl.gov/cdat/sample_data/tas_cru_1979.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_dnm-95a_1979.01-1979.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_dnm-95a_1980.01-1980.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_dnm-95a_1981.01-1981.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_dnm-95a_1982.01-1982.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_dnm-95a_1983.01-1983.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_dnm-95a_1984.01-1984.12.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_dnm-95a.xml
- https://cdat.llnl.gov/cdat/sample_data/tas_ecm_1979.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_gavg_rnl_ecm.nc

- https://cdat.llnl.gov/cdat/sample_data/tas_GFDL-ESM2G_Amon_historical_r1i1p1_198501-200512-clim.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_mo_clim.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_mo.nc
- https://cdat.llnl.gov/cdat/sample_data/tas_ukmo_con.nc
- https://cdat.llnl.gov/cdat/sample_data/taylor.nc
- https://cdat.llnl.gov/cdat/sample_data/tdata.hdf
- https://cdat.llnl.gov/cdat/sample_data/test.2.bin
- https://cdat.llnl.gov/cdat/sample_data/test_anim.nc
- https://cdat.llnl.gov/cdat/sample_data/test.bin
- https://cdat.llnl.gov/cdat/sample_data/test.cdms
- https://cdat.llnl.gov/cdat/sample_data/test_col.asc
- https://cdat.llnl.gov/cdat/sample_data/testgrib2.ctl
- https://cdat.llnl.gov/cdat/sample_data/testgrib2.grib2
- https://cdat.llnl.gov/cdat/sample_data/testgrib2.idx
- https://cdat.llnl.gov/cdat/sample_data/test_mesa_leak.nc
- https://cdat.llnl.gov/cdat/sample_data/testpp.pp
- https://cdat.llnl.gov/cdat/sample_data/test.xml
- https://cdat.llnl.gov/cdat/sample_data/thermo.nc
- https://cdat.llnl.gov/cdat/sample_data/th_yr.nc
- https://cdat.llnl.gov/cdat/sample_data/ts_da.nc
- https://cdat.llnl.gov/cdat/sample_data/u_2000.nc
- https://cdat.llnl.gov/cdat/sample_data/u_2001.nc
- https://cdat.llnl.gov/cdat/sample_data/u_2002.nc
- https://cdat.llnl.gov/cdat/sample_data/v_2000.nc
- https://cdat.llnl.gov/cdat/sample_data/v_2001.nc
- https://cdat.llnl.gov/cdat/sample_data/v_2002.nc
- https://cdat.llnl.gov/cdat/sample_data/vertical.nc
- https://cdat.llnl.gov/cdat/sample_data/vmro3_input4MIPs_ozone_DAMIP_CCMI-hist-stratO3-1-0_gr_185001_nco.nc
- https://cdat.llnl.gov/cdat/sample_data/wk_data.nc
- https://cdat.llnl.gov/cdat/sample_data/wk_results.nc
- https://cdat.llnl.gov/cdat/sample_data/wspd.coads.nc
- https://cdat.llnl.gov/cdat/sample_data/wspd.nc
- https://cdat.llnl.gov/cdat/sample_data/xieArkin-T42.nc

CHAPTER

TWENTYTHREE

API

23.1 Classes

Type	Constructor
avariable	CDMS Variable objects, abstract interface
axis	CDMS Axis objects
fvariable	CDMS File-based variables.
bindex	Bin index for non-rectilinear grids
cache	CDMS cache management and file movement objects
variable	DatasetVariable: Dataset-based variables
cdurllib	Customized URLopener
cdurlparse	Parse (absolute and relative) URLs.
cudsinterface	Emulation of old cu package
database	CDMS database objects
dataset	CDMS dataset and file objects
forecast	CDMS Forecast
gengrid	CDMS Generic Grids
grid	CDMS Grid objects
hgrid	CDMS HorizontalGrid objects
MV2	CDMS Variable objects, MaskedArray interface
selectors	Classes to support easy selection of climate data
tvariable	TransientVariable (created by createVariable) is a child of both AbstractVariable and the masked array class.
slabinterface	Read part of the old cu slab interface implemented over CDMS

23.2 Regridder

Type	Constructor
horizontal	Create a horizontal regridder
esmf	Regrid source grid data to destination grid data using ESMF
crossSection	
mvESMFRegrid	
mvGenericRegrid	
pressure	
scrip	

CHAPTER

TWENTYFOUR

INDICES AND TABLES

- genindex
- modindex
- search