UCRL-ID-123716

PCMDI

Program for Climate Model Diagnosis and Intercomparison

PCMDI Report No. 34

EzGet:
A Library of Fortran Subroutines to Facilitate Data Retrieval

by

Karl E. Taylor

April 1996 (original version)
August 1997 (revised)

PROGRAM FOR CLIMATE MODEL DIAGNOSIS AND INTERCOMPARISON
UNIVERSITY OF CALIFORNIA, LAWRENCE LIVERMORE NATIONAL LABORATORY
LIVERMORE, CA 94550

Contents

ABSTRACT

The software described in this document is designed to facilitate retrieval of modeled and observed climate data stored in popular formats including DRS, netCDF, GrADS, and, if a control file is supplied, GRIB. You can specify how the data should be structured and whether it should undergo a grid transformation before you receive it, even when you know little about the structure of the stored data (i.e., its dimension order, grid, and domain).

The software is referred to here as EzGet (pronounced "easy-get") and it comprises a set of subroutines that can be linked to any FORTRAN program. EzGet reads files through the cdunif interface which is available from the Program for Climate Model Diagnosis and Intercomparison (PCMDI), but use of EzGet does not require familiarity with cdunif. The main advantages of using this software instead of the lower level cdunif library include:

- Substantial error trapping capabilities and detailed error messages
- Versatile capability of conveniently selecting data from specified regions (e.g., oceans, North America, all land areas north of 45 degrees latitude, etc.)
- Ability to map data to a new grid at the time it is retrieved by EzGet
- Automatic creation of "weights" for use in subsequent averaging or masking of data
- Increased control in specifying the domain, grid and structure of the retrieved data.

Taken together these capabilities will simplify the process of writing programs for accessing data stored in different formats and structures, including all the observed data sets and the model output from various model intercomparison projects (AMIP, PMIP, CMIP, etc.) archived at PCMDI.

EzGet software and the latest version of this document are available through the PCMDI web site:

home page: http://www-pcmdi.llnl.gov/
EzGet location: http://www-pcmdi.llnl.gov/ktaylor/ezget/ezget.html

1 Introduction

It is usually not too difficult to write a FORTRAN program that performs the same series of calculations on several different sets of data, as long as all the data are stored on the same grid and in the same structure and format. If, however, data are found in different formats and structures, then the program can become quite complex, or will have to be revised to treat each new set of data. The software described here makes it easier to write programs that can accept data sets stored in a variety of structures. It can retrieve data that have been stored in the following cdunif-accessible formats: DRS, netCDF, GrADS, and GRIB (if a control file is supplied). It is designed to be especially useful in analyzing output from climate models. These models may have different grids, the data may be stored in different orders (e.g., south to north vs. north to south), and the dimensions may have slightly different names (e.g., "longitude" vs. "Longitude").

A powerful feature of this software is that it can both retrieve data from precisely specified domains and also map data to a common grid (specified by the user or taken from another file) so that point-wise intercomparisons between model results and between modeled and observed fields can be carried out without difficulty. Furthermore, you can automatically obtain the "weight" (which commonly is the grid-cell area) associated with each grid cell. The set of weights associated with an array of data make it easy to compute mean values (or other area-weighted statistics) and avoid the use of "if" tests in cases when data might be "missing" from some grid cells. Finally, for the models included in the Atmospheric Model Intercomparison Project (AMIP) and Paleoclimate Modeling Intercomparison Project (PMIP), special gridded maps are available from PCMDI (Program for Climate Model Diagnosis and Intercomparison) that allow one to retrieve data from individual geographical regions (e.g., North America, Indian Ocean, etc.). All the data archived at PCMDI, including the AMIP and PMIP model standard output, is accessible through this software. EzGet is limited to retrieving (multi-dimensional) rectangular arrays of data, but there is complete control over the limits of the domain, and with EzGet's masking capabilities, non-rectangular subsets of the domain can be selected.

EzGet serves a different purpose from other software tools developed at PCMDI. It is meant to be used in conjunction with FORTRAN programs that need access to data. For direct viewing and interactive manipulation of data, a Visualization and Computation System (VCS) has been developed. To transfer data to files, formats, or visualization systems, the Data and Dimensions Interface (DDI) has been developed. Neither of these can be called from a FORTRAN program.

As with any new software, some effort will be required to become familiar with EzGet's capabilities and to learn how to apply it. In the next section several tutorial examples are given that begin this process.

2 Tutorial Examples

With the help of EzGet, data can be retrieved from files (in formats readable by cdunif) through a set of subroutine calls that are described in this document. Parameters that control such capabilities as specifying precise domains, mapping data to specified grids, and masking various regions are set through subroutine calls. Typical applications of this software are illustrated by the following examples. The first example should be studied with some thoroughness, because it provides the basis for the subsequent examples.

2.1 Simple retrieval of data

Suppose we want to retrieve the surface air temperature data contained in a single file. The following short program accomplishes this:[1]

```
      program extract

c   This program
c     ** retrieves the first 12 months of global data for a variable
c        (named 'tas' in this example) that is stored in a file (named
c        '/scratch/staff/lisa/amip_obs/nasa-amip_t').
c     ** creates an array of "weights" with elements proportional to
c        grid cell area (except for grid cells with missing data
c        where the "weight" is set to 0.0).
c     ** obtains the longitude and latitude coordinates and the
c        length of each dimension retrieved.
c     ** prints out a portion of the retrieved data.


      parameter (nlon=100, nlat=50, nmon=12, n4=0)

      real adata(nlon,nlat,nmon), wtsmask(nlon,nlat,nmon),
     &      alon(nlon), alat(nlat)
      integer lons, lats, mons, i4

c     ----------------------------------------
c     Initialize EzGet:
```

[1] Further information about each of the EzGet subroutines can be found later in this section where a more completely annotated version of program `extract` appears.

```
      call initget

c     ------------------------------------------
c     Define "missing" data value.

      call defmisc('input missing value', 'real', 1.0e20)

c     ------------------------------------------
c     Define variable 1 as 'tas' and indicate path/filename where
c     it is stored.  In subsequent calls to EzGet (e.g., defdim
c     getdata, and getcoord), this field will be referred to by
c     the index assigned here (i.e., 1).

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

c     ------------------------------------------
c     Define domain for variable 1 and define order that EzGet will
c     retrieve data.

      call defdim(1, 1, 'longitude', 'width', 'nearest',
     &                                        -180.0, 180.0, 360.0)
      call defdim(1, 2, 'latitude', 'cosine', 'range',
     &                                         90.0, -90.0, 0.0)
      call defdim(1, 3, 'time',    'unit', 'nearest', 1.0, 12.0, 0.0)

c     ------------------------------------------
c     Extract variable 1 from file and create missing data mask.
c     Also return actual dimensions of retrieved data.

      lons = 0
      lats = 0
      mons = 12
      i4 = 0
      call getdata(1,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsmask, adata)

c     ------------------------------------------
c     Retrieve longitude and latitude coordinates of variable 1 (in
c     the same order as the retrieved data).

      call getcoord(1, 1, alon)
      call getcoord(1, 2, alat)

c     ------------------------------------------
c     Close all files opened by EzGet.
```

```
      call closeget

c     ----------------------------------------
c     Write data retrieved (last month only):

      write(*,'("longitudes = " / (10f8.3))') (alon(i), i=1,lons)

      write(*,'(/ "month = ", i3)') mons

      do 100 j=1,lats
        write(*,'(/"latitude = ", f8.3)') alat(j)
        write(*,'(10f8.3)') (adata(i,j,mons), i=1,lons)
  100 continue

      end
```

A fully annotated copy of this program appears below and each of the seven subroutines called by it is documented in Section 3, but here we briefly describe the purpose of each of these subroutines:

`initget` .... This subroutine must be called to initialize EzGet. It assigns default values to a few parameters and sets up some internal tables.

`defmisc` .... This subroutine ('define miscellaneous') can be used to override certain default parameters assumed by EzGet. (Note in the program above the parameter for missing data would be set to 1.0e20, but it happens that this *is* the default, so this call to `defmisc` is not really necessary.)

`defvar` .... This subroutine defines a variable that will be referenced subsequently by a simple integer index.

`defdim` .... This subroutine specifies the dimension ordering along with the desired domain for the data that will be retrieved. It also provides information for creating an array of "weights," typically set proportional to the grid-cell areas, but set to 0.0 in regions of missing data.

`getdata` .... This subroutine retrieves data and creates an appropriate mask (or set of "weights") associated with the data.

`getcoord` .... This subroutine retrieves the coordinates associated with a retrieved field.

`closeget` .... This subroutine closes any files opened by EzGet.

Here is the fully annotated version of the above program:

```
      program extract

c   This program
c     ** retrieves the first 12 months of global data for a variable
c        (named 'tas' in this example) that is stored in a file (named
c        '/scratch/staff/lisa/amip_obs/nasa-amip_t').
c     ** creates an array of "weights" with elements proportional to
c        grid cell area (except for grid cells with missing data
c        where the "weight" is set to 0.0).
c     ** obtains the longitude and latitude coordinates and the
c        length of each dimension retrieved.
c     ** prints out a portion of the retrieved data.
c
c   Concerning the structure of the stored data, we assume the
c   following:
c
c    1.  The field is a function of longitude, latitude, and time, but
c        the dimension order is unknown.
c    2.  The data are stored in a rectangular array.
c    3.  The size of the array retrieved is limited as follows:
c        a. The longitude dimension is no longer than 100 elements.
c        b. The latitude dimension is no longer than 50 elements.
c        c. The time dimension is no longer than 12 elements.
c    4.  The names of the dimensions (as stored in the file) are:
c        'latitude', 'longitude', and 'time' (but not necessarily in
c        that order).
c    5.  The longitude coordinates are evenly spaced.
c    6.  The latitude coordinates are evenly spaced.
c    7.  The units for longitude and latitude are degrees.
c    8.  Only one field in the file has the name, 'tas' (i.e., this
c        variable name is unique in the file).
c    9.  The number, 1.0e20, is stored in the place of the true data
c        anywhere that data are missing (i.e., this is the "missing
c        data" value or indicator).
c
c
c    This program obtains the following information:
c
c    lons = the actual length of the longitude dimension of the data
c             retrieved from storage.
c    lats = the actual length of the latitude dimension of the data
c             retrieved from storage.
c    mons = the actual number of months of data retrieved from storage.
c    alon(100) = a vector containing the longitude coordinates for the
```

```
c                data.
c    alat(50) = a vector containing the latitude coordinates for the
c                data.
c    adata(100,50,12) = the retrieved array of data, which according to
c       our specifications given below, will be put in the following
c       structure (regardless of how it was originally stored):
c       1. The dimension order will be: longitude, latitude, time
c          (i.e., in the array, "adata", the first index is associated
c          with longitude, the second with latitude, and the third with
c          time).
c       2. The longitudes will be ordered from west to east, starting
c          with the longitude nearest to 180 W.
c       3. The latitudes will be ordered north to south.
c       4. The months will be ordered consecutively.
c    wtsmask(100,50,12) = the created "missing data" mask which will
c       be ordered the same as "adata", with elements set proportional
c       to the grid cell area (except for grid cells with missing data.
c       where the elements will be set to 0.0)
c
c    Note that if lons < nlon and/or lats < nlat, and/or mons < nmon,
c    then the extra elements of the array, "adata" and "wtsmask" will be
c    assigned a value of 0.0 by EzGet.

      parameter (nlon=100, nlat=50, nmon=12, n4=0)

      real adata(nlon,nlat,nmon), wtsmask(nlon,nlat,nmon),
     &      alon(nlon), alat(nlat)
      integer lons, lats, mons, i4

c     ------------------------------------------
c     Initialize EzGet:

      call initget

c     ------------------------------------------
c     Tell EzGet to consider "missing" any data that have
c     values (within a small tolerance) of 1.0e20.

      call defmisc('input missing value', 'real', 1.0e20)

c     ------------------------------------------
c     Define variable 1 as 'tas' and indicate path/filename where
c     it is stored.  In subsequent calls to EzGet (e.g., defdim
c     getdata, and getcoord), this field will be referenced by
c     the index assigned here (i.e., 1).
```

```
      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

c     ----------------------------------------
c     Define domain for variable 1 and define order that EzGet will
c     retrieve data.

      call defdim(1, 1, 'longitude', 'width', 'nearest',
     &                                          -180.0, 180.0, 360.0)
      call defdim(1, 2, 'latitude', 'cosine', 'range',
     &                                            90.0, -90.0, 0.0)
      call defdim(1, 3, 'time',    'unit', 'nearest', 1.0, 12.0, 0.0)

c     where in the first call to defdim above the arguments indicate
c     the following:
c
c      1, 1, 'longitude':
c         The first 3 arguments indicate that for variable 1 (as
c         indicated by 1st argument), the data should be retrieved such
c         that the dimension named 'longitude' (as indicated by the
c         3rd argument) is the first dimension  (as indicated by the 2nd
c         argument).
c      'width': this argument controls the creation of the array of
c           "weights" that will be associated with the data. For each
c           array element the weight is equal to the product of the
c           weights defined by each dimension and here 'width' specifies
c           that the "weights" should be proportional to the
c           longitudinal width of each grid cell.
c      'nearest', -180.0, 180.0, 360.0:
c           these 4 arguments define the domain that will be retrieved
c           (and also the order of retrieval).  In this case all data
c           with a longitude coordinate roughly in the range -180 to 180
c           will be extracted, starting near -180.  The value 360.0
c           indicates that this coordinate is cyclical with a period of
c           360, so that EzGet will recognize equivalences such as
c           0. = 360. = -360. and -90. = 270.  Note that with 'nearest'
c           specified and with the range covering a complete cycle,
c           EzGet may shift the domain slightly so as to prevent a grid
c           cell near -180.0 (=180.0) from being split across the
c           boundary.  If, for example, the center of grid cells
c           are located at -180, -170, -160, ... 170, then EzGet will
c           shift the requested domain to -185 to 175 so the grid cell
c           at -180 will not be split.
c
c     and in the second call to defdim above the arguments indicate the
```

```
c     following:
c      'cosine': this option for controlling creation of weights
c           specifies that weights be generated equal approximately
c           to the cosine of latitude (i.e.,  abs(sin(bdry1) -
c           sin(bdry2)), where bdry1 and bdry2 are edges of the latitude
c           grid-cell, assumed to be half-way between grid-cell centers)
c           be assigned to all elements (except the weight will be 0.0
c           for grid cells with missing data).  Another option
c           ('gaussian') would be appropriate for spectral models with
c           gaussian grids (as opposed to the evenly spaced grid
c           accessed here).
c      'range', 90.0, -90.0, 0.0:
c           these 4 arguments define the domain that will be retrieved
c           (and also the order of retrieval).  In this case all data
c           with latitude coordinates in the range 90 to -90 will
c           be retrieved, starting near 90 (i.e. data will be retrieved
c           from north to south).  The value 0.0 indicates that this
c           coordinate is not cyclical.
c
c     and in the third call to defdim above the fourth argument
c     indicates the following:
c      'unit': this option for controlling creation of weights specifies
c           that unit weight should be given to each element of this
c           dimension (except the weight will be 0. for grid cells with
c           missing data).
c
c     ----------------------------------------
c     Extract variable 1 from file and create missing data mask.
c     The lengths of the longitude, latitude and time dimensions of
c     the data stored in the file are unknown, so initialize the
c     "expected" dimension lengths to 0.  Return the actual dimensions
c     of the retrieved data.

      lons = 0
      lats = 0
      mons = 12
      i4 = 0
      call getdata(1,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsmask, adata)

c     where on calling getdata the arguments have been defined as
c     follows:
c
c      1: The first argument has the value 1 and indicates that data
c         will be extracted for variable 1 (defined in defvar above).
```

```
c      nlon, nlat, nmon, n4:
c         These are the declared dimensions of the arrays, wtsmask and
c         adata.  Note that n4=0 because these arrays have only 3
c         dimensions.
c      lons, lats, mons, i4:
c         In this example, the user does not know how large the actual
c         longitude and latitude dimensions of the data being retrieved
c         will be, so these are set to 0.  The time dimension is
c         expected to be 12 (months) as specified in the earlier call to
c         defdim.  In general, if the user knows the size of the domain
c         to be retrieved, it is usually prudent to set these arguments
c         to the expected size of the domain, because then EzGet can
c         error exit if the actual size differs from what is expected.
c
c     On return from getdata:
c
c       1, nlon, nlat, nmon, n4:
c          The first 5 arguments will be unchanged.
c       lons, lats, mons, i4:
c          returns the actual dimensions of the data retrieved.
c       wtsmask: returns the weights associated with the data.
c          In this example, the weights will either be 0.0 or
c          proportional to grid cell area, depending on whether or not
c          the data are missing.
c       adata: returns the extracted data.

c     ----------------------------------------
c     Retrieve longitude and latitude coordinates of variable 1 (in
c     the same order as the retrieved data).

      call getcoord(1, 1, alon)
      call getcoord(1, 2, alat)

c     where in the first call to getcoord above, the arguments indicate
c     the following:
c
c      1: The 1st argument has the value 1 and indicates that we want
c           to obtain the coordinates of data extracted for variable 1
c           (as defined in the earlier call to defvar).
c      1: The 2nd argument has the value 1 and indicates that we want
c           to obtain the coordinates for the 1st dimension (as defined
c           in the earlier call to defdim).
c      alon: returns the coordinate values for the 1st dimension of
c           of variable 1 (i.e., 'longitude', as defined by defdim).
```

```
c     ----------------------------------------
c     Close all files opened by EzGet.

      call closeget

c     ----------------------------------------
c     Write data retrieved for last month.  Note that any missing data
c     will have been assigned the value 1.e20, so when written with
c     f8.3 format will appear as '********':

      write(*,'("longitudes = " / (10f8.3))') (alon(i), i=1,lons)

      write(*,'(/ "month = ", i3)') mons

      do 100 j=1,lats
        write(*,'(/"latitude = ", f8.3)') alat(j)
        write(*,'(10f8.3)') (adata(i,j,mons), i=1,lons)
  100 continue

      end
```

2.2 Masking geographical regions when retrieving data

In order to mask out data that lie outside some geographical region of interest, a file must be created containing data that uniquely identify each grid cell as belonging to a particular geographical region (such as 'land' or 'ocean' for land/sea type masks (which for AMIP 1 data are named 'sft', for AMIP 2 data, 'sftl', and for PMIP data, 'sftland'), or such as 'North America', 'Indian Ocean', 'South Atlantic', etc. for region type masks (which for AMIP and PMIP data are named 'sftbyrgn'). Geography data files compatible with EzGet are available for most model grids from PCMDI. Besides the EzGet subroutines discussed in example 1, the following subroutine will also be called:

`defgeog ....` This subroutine makes it possible for you to specify that certain geographical regions should be masked out.

Suppose we want to obtain data from North America for the region north of the Tropic of Cancer. The following program accomplishes this task.

```
      program getregn
```

```
c   This program
c    ** retrieves surface temperature data for the region of
c       North America north of 23.5 N latitude.  (Note: data are
c       extracted only for grid cells whose coordinates are not outside
c       the range 23.5 N to 90.0 N and 190 W to 40 W.)
c    ** creates an array of "weights", with elements set proportional
c       to the area of the grid cells falling within the selected
c       geographical region (except for those grid cells that contain
c       "missing" data or which lie outside North America, in which case
c       the weight is set to 0.0).
c    ** obtains the longitude and latitude coordinates and
c       the length of each dimension retrieved.

c   This program is in many ways similar to example 1, and
c   further explanation can be found in the comments appearing in
c   that example (program extract).

c   Note that the region is selected in two ways.
c   First, all data for grid cells in the region 23.5 N to 90.0 N and
c   190 W and 40 W are extracted, and then all grid-cells that are
c   outside the North American boundaries are masked out.  The size of
c   the arrays, adata and wtsmask, only need to be large enough to
c   accommodate the region extracted (before the geography mask is
c   applied).

      parameter (nlon=35, nlat=25, nmon=15, n4=0)

      real adata(nlon,nlat,nmon), wtsmask(nlon,nlat,nmon),
     &      alon(nlon), alat(nlat)
      integer lons, lats, mons, i4
c     ----------------------------------------
c     Initialize EzGet and define "missing" data value:

      call initget
      call defmisc('input missing value', 'real', 1.0e20)

c     ----------------------------------------
c     Define variable 1 as 'tas' and define its domain and the order
c     that EzGet will retrieve data.

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

      call defdim(1, 1, 'longitude', 'width', 'range',
     &                                        -190.0, -40.0, 360.0)
```

```
c     >>> Note that the longitude range specified above is large enough
c     >>> to accommodate all North American grid cells.

      call defdim(1, 2, 'latitude', 'cosine', 'range', 23.5, 90.0, 0.0)
      call defdim(1, 3, 'time',    'unit', 'nearest', 1.0, 12.0, 0.0)

c     ------------------------------------------
c    >>>  Define variable 2 as 'sftbyrgn' and indicate path/filename
c    >>>  where it is stored.  This variable should contain a "geography
c    >>>  mask" that is compatible with the EzGet convention for
c    >>>  identifying different geographical regions (see documentation
c    >>>  of subroutine defgeog), and it should be on the same grid as
c    >>>  variable 1 defined above.  In subsequent calls to EzGet (e.g.,
c    >>>  defgeog) this field will be referred to by the index assigned
c    >>>  here (i.e., 2).

      call defvar(2, 'sftbyrgn', '/amipsp/drs/sftbyrgn/sftbyrgn_gla')

c     ------------------------------------------
c    >>>  Control geographical masking of retrieved data.

      call defgeog(1, 'in', 2, 'North America')

c    >>>  where the arguments indicate the following:
c
c    >>>   1:    The first argument has the value 1 and indicates that
c    >>>     the geography mask will be applied to variable 1.
c    >>>   'in': indicates that the masking should be done before any
c    >>>     regridding is performed. (See later examples for further
c    >>>     explanation.)
c    >>>   2:    Indicates that variable 2 contains the geography data.
c    >>>   'North America': indicates that this is the region of
c    >>>         interest and any data outside this region should be
c    >>>         masked.

c     ------------------------------------------
c     Extract variable 1 from file and mask data outside region of
c     interest.

      lons = 0
      lats = 0
      mons = 12
      i4 = 0
      call getdata(1,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsmask, adata)
```

```
c     ----------------------------------------
c     Retrieve longitude and latitude coordinates of variable 1.

      call getcoord(1, 1, alon)
      call getcoord(1, 2, alat)

c     ----------------------------------------
c     Close all files opened by EzGet.

      call closeget

c     ----------------------------------------
c     Write out weights and data retrieved (last month only):

      write(*,'("longitudes = " / (10f8.3))') (alon(i), i=1,lons)

      write(*,'(/ "month = ", i3)') mons

      do 100 j=1,lats
        write(*,'(/"latitude = ", f8.3)') alat(j)
        write(*,'("weights:")')
        write(*,'(9f9.6)') (wtsmask(i,j,mons), i=1,lons)
        write(*,'("temperature:")')
        write(*,'(9f9.3)') (adata(i,j,mons), i=1,lons)
  100 continue

      end
```

2.3 Mapping data to a specified grid

Suppose that data have been stored on a longitude-latitude grid that is different from the grid you might require for a particular application (e.g., to compare to data stored on a different grid). EzGet will map the data to some grid that you specify. Besides the EzGet subroutines discussed in example 1, the following subroutine will be called:

`defregrd ....` This subroutine controls mapping of data to a specified grid.

Suppose we want to retrieve data and regrid it to a 10° by 15° latitude-longitude grid. The following program accomplishes this task.

```
      program regrding

c   This program
c     ** retrieves global surface temperature data and maps
c        it to a 10 degree by 15 degree latitude-longitude "target"
c        grid using an area-weighted averaging scheme.
c     ** creates an array of "weights", with elements set
c        proportional to the sum of the areas of the original (i.e.,
c        source) grid cells that contribute to each target cell.
c     ** obtains the longitude and latitude coordinates
c        of the target grid and the length of each dimension of the
c        target grid.

c   This program is in many ways similar to example 1, and
c   further explanation can be found in the comments appearing in
c   that example (program extract).

c   Note that in order to apply the area-weighting regridding algorithm,
c   the user must specify what type of grid the original data were
c   stored on (e.g., gaussian, evenly-spaced, etc.)

c   The size of the arrays, adata and wtsmask, only need to be large
c   enough to accommodate the region extracted (i.e. after regridding),
c   so nlon .ge. 360/15 = 24 and nlat .ge. 180/10 = 18.

      parameter (nlon=24, nlat=18, nmon=12, n4=0)

      real adata(nlon,nlat,nmon), wtsmask(nlon,nlat,nmon),
     &      alon(nlon), alat(nlat)
      integer lons, lats, mons, i4
c     ----------------------------------------
c     Initialize EzGet and define "missing" data value:

      call initget
      call defmisc('input missing value', 'real', 1.0e20)

c     ----------------------------------------
c     Define variable 1 as 'tas' and indicate path/filename where
c     it is stored.

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

c     ----------------------------------------
c     Define domain for variable 1 and define order that EzGet will
c     retrieve data.   Also indicate what type of grid the source
```

```
c     data were stored on.

      call defdim(1, 1, 'longitude', 'width', 'range', 0.0, 0.0, 360.0)
      call defdim(1, 2, 'latitude', 'cosine', 'range', 0.0, 0.0,   0.0)
      call defdim(1, 3, 'time',    'unit', 'nearest',  1.0, 12.0, 0.0)
c
c      When regridding data, the "cycle" for longitude should always
c      be specified (set to 360.0 in the above example).
c      The domain limits specified for longitude and latitude will be
c      overridden by the arguments in the call to subroutine defregrd
c      below.

c     ----------------------------------------
c     Define target grid to which data should be mapped.

      call defregrd(1, 'uniform', 0, 'area-weighted', 18, 85.0, -10.0,
     &                                                 24, -172.5, 15.0)

c     where the arguments in the subroutine call indicate the following:
c
c      1:    The first argument has the value 1 and indicates that the
c            the regridding will be applied to variable 1.
c      'uniform': indicates that the target grid will be rectangular
c            grid of evenly-spaced latitude and longitude cells.
c      0:    This argument is ignored because the 2nd argument was set
c            to 'uniform'.
c      'area-weighted': indicates that an area-weighted mapping scheme
c            should be used.
c      18, 85.0, -10.0:
c            These 3 arguments define the latitude grid that will be
c            created.  In this case a grid with 18 latitude cells
c            is created with the first grid cell centered at 85.0
c            degrees north and proceeding southward in increments of
c            10 degrees.
c      24, -172.5, 15.0:
c            These 3 arguments define the longitude grid, indicating
c            that there will be 24 longitude grid cells, with the first
c            grid cell centered at -172.5 degrees west and proceeding
c            eastward in increments of 15 degrees.
c
c     ----------------------------------------
c     Extract variable 1 from file and map to target grid defined above.

      lons = 24
      lats = 18
```

```
      mons = 12
      i4 = 0
      call getdata(1,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsmask, adata)

c     Note: we have defined the expected longitude and latitude
c     dimension to be 24 and 18, respectively (as specified by lons and
c     lats) because we know the data will be regridded to the grid
c     defined by the call to defregrd, which specifies a global 10 x 15
c     degree latitude-longitude grid.  It is always good practice to
c     define the expected dimensions if they are known.

c     ----------------------------------------
c     Retrieve longitude and latitude coordinates of variable 1.

      call getcoord(1, 1, alon)
      call getcoord(1, 2, alat)

c     Note:  According to the parameters defining the target grid,
c          alon should contain -172.5, -157.5, . . . 172.5 and
c          alat should contain  85.0, 75.0. . . -85.0

c     ----------------------------------------
c     Close all files opened by EzGet.

      call closeget

c     ----------------------------------------
c     Write data retrieved (last month only):

      write(*,'("longitudes = " / (10f8.3))') (alon(i), i=1,lons)

      write(*,'(/ "month = ", i3)') mons

      do 100 j=1,lats
        write(*,'(/"latitude = ", f8.3)') alat(j)
        write(*,'(10f8.3)') (adata(i,j,mons), i=1,lons)
  100 continue

      end
```

2.4 Computing an area average

There are several ways that EzGet can facilitate the computation of area averages. Suppose we want to compute the average temperature over North America north of the Tropic of Cancer for each month of data stored in a file. Perhaps the simplest method makes use of the area-weighting regridding algorithm and the masking capabilities of EzGet as illustrated by the following program:

```
      program areamean

c   This program
c
c    ** retrieves surface temperature data for the region of
c       North America north of 23.5 N latitude.  It then uses the
c       regridding capability of EzGet to compute the area average of a
c       single "grid-cell" covering the region of North America north of
c       23.5 N.  Up to 120 months of data can be extracted at once and
c       the area means are returned to this program as a vector, one
c       element for each month of data extracted.
c
c    ** creates a vector of "weights" (one for each month)
c       proportional to the area of the region over which the means have
c       been computed.  The elements of this vector should be
c       identical (except possibly if grid cells are missing data).

c   This program is in many ways similar to examples 1, 2 and 3, and
c   further explanation can be found in the comments appearing in
c   those examples (programs extract, getregn, and regrding).

c  >>> Note that nlon and nlat can be declared as small as 1, because
c  >>> these dimensions reduce to a single grid cell after regridding.

      parameter (nmon=120, nlon=1, nlat=1, n4=0)

      real amean(nmon, nlon, nlat), wtsmask(nmon, nlon, nlat)
      integer lons, lats, mons, i4
c     ----------------------------------------
c     Initialize EzGet and define "missing" data value:

      call initget
      call defmisc('input missing value', 'real', 1.0e20)

c     ----------------------------------------
```

```
c     Define variable 1 as observed surface temperature data ('tas')
c     and define its domain:

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

      call defdim(1, 1, 'time', 'unit', 'as saved', 0.0, 0.0, 0.0)
      call defdim(1, 2, 'longitude', 'width', 'range', 0.0, 0.0, 360.0)
      call defdim(1, 3, 'latitude', 'cosine', 'range', 0.0, 0.0,   0.0)

c     >>>    Note that in the 1st call to defdim above, the last 4
c     >>>    arguments are:
c     >>>    'as saved', 0.0, 0.0, 0.0:
c     >>>    'as saved' specifies that all months should be retrieved in
c     >>>    the order that they were originally stored in the file.
c     >>>    (In this case the last 3 arguments [0.0, 0.0, 0.0] are
c     >>>    ignored.)

c     ------------------------------------------
c     Define variable 2 as the geography data needed and specify that
c     it be used to select North American data only for variable 1:

      call defvar(2, 'sftbyrgn', '/amipsp/drs/sftbyrgn/sftbyrgn_gla')
      call defgeog(1, 'in', 2, 'North America')

c     ------------------------------------------
c     Define target grid to which data should be mapped.

      call defregrd(1, 'uniform', 0, 'area-weighted',
     &                        1, 56.75, 66.5,  1, -125.0, 150.0)

c     where the target grid has been specified as a single grid cell
c         centered at 125 W longitude, 56.75 N latitude and has
c         latitude-longitude dimensions of 66.5 x 150.

c     ------------------------------------------
c     Extract variable 1, which, because of regridding to a single cell,
c     contains the area-weighted mean for each month.

      mons = 0
      lons = 1
      lats = 1
      i4 = 0
      call getdata(1,  nmon,nlon,nlat,n4,  mons,lons,lats,i4,
     &             wtsmask, amean)
```

```
c     ----------------------------------------
c     Close all files opened by EzGet.

      call closeget

c     ----------------------------------------
c     Write data retrieved:

      write(*,'("North American Mean (North of 23.5 N)" //
     &        " month      area      mean" / )')

      write(*,'(i5, f12.7, f10.3)')
     &                  (m, wtsmask(m,1,1), amean(m,1,1), m=1,mons)

      end
```

2.5 A typical amip application

Model output from the Atmospheric Model Intercomparison Project (AMIP) has been archived in a format readable by EzGet. It is often of interest to compare the model simulated fields by the AMIP models to observations. One measure of difference between two fields is the root-mean-square (RMS) difference. The following program computes the area-weighted RMS difference between model-simulated and observed fields of annual mean surface air temperature over North America in the year 1988 (i.e., year 10 of the AMIP simulations). Before computing the RMS difference, the model output is mapped to the same grid as the observations.

Note that it is not necessary for you to know either the resolution or the type of grid used by each model (because EzGet retrieves the resolution and for AMIP and PMIP models EzGet looks up the grid type in a table). The data extracted is mapped to the observed grid and all arrays you define are simply declared large enough to contain data on the observed grid. (EzGet dynamically allocates space needed to accommodate data on the original (i.e., source) grid.)

The following program will perform the RMS calculation:

```
      program rmscalc

c   This program loops through the AMIP models and computes the area-
c   weighted RMS difference between the modeled and observed annual mean
c   surface air temperature over North America (north of the Tropic of
```

```
c   Cancer) for the year 1988.  Only grid cells with observed values for
c   every month of this year are included in computing the RMS
c   difference.  Because some models may have unrealistic
c   representations of the continental geography, some data from the
c   model's North American grid may not map onto the observed North
c   American grid. Any grid cells where either observations or model
c   data are missing are excluded from the RMS difference computed
c   here.

      parameter (nlon=35, nlat=20, nmon=12, n4=0, nmods=28)

      real adata(nlon, nlat, nmon), wtsmask(nlon, nlat, nmon),
     &     yrobs(nlon,nlat), yrmodel(nlon,nlat),
     &     wtobs(nlon,nlat), wtmodel(nlon,nlat)
      double precision asum, rmsdiff, wtsum
      integer lons, lats, mons, i4, i, j, m, n, mm
      character*3 modlname(nmods)
      data modlname /   'bmr', 'ccc', 'col', 'cnr', 'csi', 'csu', 'der',
     &    'dnm', 'ecm', 'gfd', 'gis', 'gla', 'gsf', 'iap', 'jma', 'lmd',
     &    'mgo', 'mpi', 'mri', 'nca', 'nmc', 'nrl', 'sng', 'sun', 'ucl',
     &    'uiu', 'ukm', 'yon' /

c      Note:  the 'rpn' and 'uga' models were left out of the above list
c      because the variable 'tas' is not available for these
c      models.

c     ------------------------------------------
c     Initialize EzGet and define "missing" data value:

      call initget
      call defmisc('input missing value', 'real', 1.0e20)

c     ------------------------------------------
c     Define variable 1 as the observed surface air temperature and
c     indicate path/filename where data are stored.

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

c     ------------------------------------------
c     Define domain for variable 1.

      call defdim(1, 1, 'longitude', 'width', 'range',
     &                                          -190.0, -40.0, 360.0)
      call defdim(1, 2, 'latitude', 'cosine', 'range', 23.5, 90.0, 0.0)
      call defdim(1, 3, 'time',    'unit', 'nearest', 109., 120., 0.0)
```

```
c     ------------------------------------------
c     Define variable 2 as the geography data on the observational grid,
c     and specify that North American data should be selected:

      call defvar(2, 'sftbyrgn', '/amipsp/drs/sftbyrgn/sftbyrgn_gla')
      call defgeog(1, 'in', 2, 'North America')

c     ------------------------------------------
c     Extract variable 1 (observed field).

      lons = 0
      lats = 0
      mons = nmon
      i4 = 0
      call getdata(1,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsmask, adata)

c     ------------------------------------------
c     Compute annual mean at each grid cell, but only if all 12 months
c     of observed data are available.

      do 300 j=1,nlat
        do 200 i=1,nlon

          asum = 0.0
          wtsum = 0.0
          mm = 0

          do 100 m=1,12

            if (wtsmask(i,j,m) .gt. 0.) then
              mm = mm + 1
              wtsum = wtsum + wtsmask(i,j,m)
              asum = asum + adata(i,j,m)
            endif

  100     continue

          if (mm .eq. 12) then
            yrobs(i,j) = asum/12.
            wtobs(i,j) = wtsum/12.
          else
            yrobs(i,j) = 0.0
            wtobs(i,j) = 0.0
```

```
          endif

  200   continue
  300 continue

c     ----------------------------------------
c     Loop through AMIP models and compute RMS differences between
c     modeled and observed fields.

      do 1000 n=1,nmods

c       ---------------------------------------
c       Define variable 3 as the model simulated surface air temperature
c       and specify the type of grid this model has:

        call defvar(3, 'tas', '/amipsp/drs/tas/tas_'//modlname(n))

        call defdim(3, 1, 'longitude', modlname(n), 'nearest',
     &                                              0.0, 0.0, 360.0)
        call defdim(3, 2, 'latitude', modlname(n), 'nearest',
     &                                               0.0, 0.0, 0.0)
        call defdim(3, 3, 'time',   'unit', 'nearest', 109., 120., 0.0)

c       >>> Note that the domains specified above for latitude and
c       >>> longitude are ignored by EzGet because the data for this
c       >>> variable will be regridded to a target grid, which
c       >>> determines the domain.

c       >>> Note that the name of the model can be used to specify the
c       >>> type of longitude and latitude weights that will be
c       >>> generated by EzGet.  EzGet contains a table that allows
c       >>> it to generate the proper weights proportional to grid cell
c       >>> area, if the model name is a standard AMIP or PMIP name.

c       >>> Note that for AMIP data, months 109 through 120 are the
c       >>> months of the calendar year 1988.

c       ---------------------------------------
c       Define variable 4 as the geography data on the model grid, and
c       specify that North American data should be selected for
c       variable 3.

        call defvar(4, 'sftbyrgn',
     &              '/amipsp/drs/sftbyrgn/sftbyrgn_'//modlname(n))
        call defgeog(3, 'in', 4, 'North America')
```

```
c       ----------------------------------------
c       Instruct EzGet to map modeled field to observational grid upon
c       retrieving data.

        call defregrd(3, 'to', 1,  'area-weighted',
     &                                0, 0.0, 0.0,  0, 0.0, 0.0)

c       ----------------------------------------
c       Extract variable 3 and map to observed grid.

        lons = 0
        lats = 0
        mons = nmon
        i4 = 0
        call getdata(3,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsmask, adata)

c       ----------------------------------------
c       Compute annual mean at each grid cell, but only if all 12 months
c       of data are available.

        do 600 j=1,nlat
          do 500 i=1,nlon

            asum = 0.0
            wtsum = 0.0
            mm = 0

            do 400 m=1,12

              if (wtsmask(i,j,m) .gt. 0.) then
                mm = mm + 1
                wtsum = wtsum + wtsmask(i,j,m)
                asum = asum + adata(i,j,m)
              endif

  400       continue

            if (mm .eq. 12) then
              yrmodel(i,j) = asum/12.
              wtmodel(i,j) = wtsum/12.
            else
              yrmodel(i,j) = 0.0
              wtmodel(i,j) = 0.0
```

```
            endif

  500     continue
  600   continue

c       ----------------------------------------
c       Compute area-weighted RMS difference between model and observed
c       fields.

        wtsum = 0.0
        rmsdiff = 0.0

          do 800 j=1,nlat
          do 700 i=1,nlon

            if (wtmodel(i,j) .gt. 0.0) then
              wtsum = wtsum + wtobs(i,j)
              rmsdiff = rmsdiff +
     &                        wtobs(i,j)*((yrobs(i,j)-yrmodel(i,j))**2)
            endif

  700     continue
  800   continue

c       ----------------------------------------
c       Write area-weighted RMS difference between model and observed
c       fields.

        if (wtsum .gt. 0.0) then

          rmsdiff = dsqrt(rmsdiff/wtsum)
          write(*,'(1x, a8, f12.7, f12.3)') modlname(n), wtsum, rmsdiff

        else

          write(*,'(1x, a8, " data missing")') modlname(n)

        endif

 1000 continue

c     ----------------------------------------
c     Close all files opened by EzGet.

      call closeget
```

```
      end
```

2.6 Placing data in contiguous memory

There are times when you might prefer that the data retrieved by EzGet be forced to occupy contiguous memory. Suppose, for example, that after you read the data, you plan to treat it as a vector string of elements (perhaps because you do not care about the longitude-latitude structure of the data). Then instead of calling `getdata`, it may be more convenient to call `getnogap`, which always places the data retrieved by EzGet in contiguous memory. There is often a computational advantage to treating a single, one-dimensional vector of data rather than a multidimensional array containing the same data. Also it may be easier to reduce the internal memory required by EzGet if the data set is thought of as a 1-dimensional vector of elements.

In the following example, the maximum and minimum values of a field of unknown structure are found. By calling `getnogap` rather than `getdata`, we can make the program more general (while using memory efficiently). Besides the EzGet subroutines discussed in example 1, the following subroutines will be called:

`getnogap ....` This subroutine retrieves data and creates an appropriate mask (or set of "weights") associated with the data. It differs from `getdata` in that the data are forced to occupy contiguous memory.

`domain ....` This subroutine retrieves from a file the number of dimensions, the names of the dimensions and the full domain of each dimension of a field (as originally stored) for any variable.

`lendims ....` After retrieving data, this subroutine returns the actual length of each dimension of the data that has been retrieved.

```
      program maxmin

c   This program
c     ** obtains the structure of a variable (the number and length
c        of each of its dimensions).
c     ** finds the maximum and minimum value stored for the variable
c        (but skips 'missing' data).
c     ** prints out the information retrieved.
c
c   This program will error exit if a data set's size exceeds maxsize
c     (declared in the parameter statement below)
```

```
      parameter (maxsize=500000)

      real adata(maxsize), wtsmask(maxsize)
      real begdom(4), enddom(4), rmax, rmin
      integer ldim(4), isize, ndim, n
      character*16 dimnames(4)

c     ------------------------------------------
c     Initialize EzGet:

      call initget

c     ------------------------------------------
c     Define "missing" data value.

      call defmisc('input missing value', 'real', 1.0e20)

c     ------------------------------------------
c     Define variable 1 as 'tas' and indicate path/filename where
c     it is stored.  In subsequent calls to EzGet (e.g., defdim
c     getdata, and getcoord), this field will be referred to by
c     the index assigned here (i.e., 1).

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

c     ------------------------------------------
c     Obtain from the file the number of dimensions, the names of the
c     dimensions and the full domain of each dimension of the variable
c     (as originally stored).

      call domain(1, ndim, dimnames, begdom, enddom)

c     where
c      1:    The first argument has the value 1 and indicates that the
c            you want to retrieve dimension information for variable 1.
c      ndim: This argument returns the number of dimensions there are
c            for variable 1.
c      dimnames: This vector argument returns the dimension names.
c      begdom:  This vector argument returns the first coordinate value
c            stored for each dimension.
c      enddom:  This vector argument returns the last coordinate value
c            stored for each dimension.

c     ------------------------------------------
```

```
c     Define domain for variable 1:  retrieve all data stored

      do 100 n=1,ndim
         call defdim(1, n, dimnames(n), 'unit', 'as saved', 0., 0., 0.)
  100 continue

c     ------------------------------------------
c     Extract variable 1 from file and create missing data mask.

      call getnogap(1, maxsize, wtsmask, adata)

c     where
c     1:     The first argument has the value 1 and indicates that the
c            you want to retrieve data from variable 1.
c     wtsmask: returns the weights associated with the data.
c     adata:   returns the extracted data.

c     ------------------------------------------
c     Retrieve the length of each dimension of the variable and the
c     total length of the vector of data retrieved:

      call lendims(1, ldim(1), ldim(2), ldim(3), ldim(4), isize)

c     where
c     1:    The first argument has the value 1 and specifies that the
c           the dimension lengths for variable 1 should be obtained.
c     ldim(1), ldim(2), ldim(3), ldim(4):  return the lengths of
c           dimensions 1, 2, 3, and 4, respectively.
c     isize:  isize = ldim(1)*ldim(2)*ldim(3)*ldim(4), but if any
c     dimension is length 0, a 1 is substituted in the above formula.

c     ------------------------------------------
c     Close all files opened by EzGet.

      call closeget

c     ------------------------------------------
c     Find the maximum and minimum values retrieved (skipping "missing"
c     values).

      rmax = -1.e40
      rmin =  1.e40

      do 200 n=1,isize
         if (wtsmask(n) .gt. 0.0) then
```

```
            rmax = amax1(rmax, adata(n))
            rmin = amin1(rmin, adata(n))
          endif
  200 continue

c     ----------------------------------------
c     Report the structure of the data and the maximum and minimum
c     values stored.

      write(*,'("         Data Structure" /
     &          " Dimension Name       Length " / /
     &      4(a16, i10 /))')
     &      (dimnames(n), ldim(n), n=1,ndim)

      if (rmax .ge. rmin) then
        write(*,'(" Maximum value found: ", 1pe14.5)') rmax
        write(*,'(" Minimum value found: ", 1pe14.5)') rmin
      else
        write(*,'(" Data set includes only ''missing'' data.")')
      endif

      end
```

2.7 Reducing memory requirements

In the previous example the arrays might have been declared much larger than necessary, just to make the program general enough to retrieve the largest data set likely to be encountered. When memory is scarce, it would be better to declare the array to be just large enough to accomodate the data being retrieved. Furthermore, the data mask is not really needed in this case since missing values are also indicated by the value, 1.0e20, stored in array `adata`. The following revised version of the program given in the previous example dynamically allocates (upon execution) just enough memory to accomodate the data retrieved. It also avoids creating the data mask, which is not needed. Besides the EzGet subroutines discussed previously, the following subroutines will be called:

`getfield` .... This subroutine simply retrieves data, but does no masking or mapping to a different grid. Unlike `getnogap`, no data mask or weights are returned.

`shape` .... Before retrieving data, this subroutine returns the length of each dimension of a field as it will be retrieved by EzGet.

```
      program smmaxmin

c   This program
c     ** obtains the structure of a variable (the number and length
c        of each of its dimensions).
c     ** finds the maximum and minimum value stored for the variable
c        (but skips 'missing' data).
c     ** prints out the information retrieved.
c
c   This program dynamically allocates enough memory to accomodate
c     the retrieved data. It also avoids creating a data mask.

      pointer (ptadata, adata)
      real adata(*)
      real begdom(4), enddom(4), rmax, rmin, err
      integer ldim(4), isize, ndim, n
      character*16 dimnames(4)

c     ------------------------------------------
c     Initialize EzGet:

      call initget

c     ------------------------------------------
c     Define "missing" data value.  (Neither of these calls is
c     actually necessary, since the default input and output missing
c     value is 1.0e20.

      call defmisc('input missing value', 'real', 1.0e20)
      call defmisc('output missing value', 'real', 1.0e20)

c     ------------------------------------------
c     Define variable 1 as 'tas' and indicate path/filename where
c     it is stored.  In subsequent calls to EzGet (e.g., defdim
c     getdata, and getcoord), this field will be referred to by
c     the index assigned here (i.e., 1).

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

c     ------------------------------------------
c     Obtain from the file the number of dimensions, the names of the
c     dimensions and the full domain of each dimension of the variable
c     (as originally stored).

      call domain(1, ndim, dimnames, begdom, enddom)
```

```
c     ------------------------------------------
c     Define domain for variable 1:  retrieve all data stored

      do 100 n=1,ndim
         call defdim(1, n, dimnames(n), 'unit', 'as saved', 0., 0., 0.)
  100 continue

c     ------------------------------------------
c     Retrieve the length of each dimension of the variable and the
c     total length of the vector of data that will be extracted:

      call shape(1, ldim(1), ldim(2), ldim(3), ldim(4), isize)

c     where
c     1:    The first argument has the value 1 and specifies that
c           the dimension lengths that will be returned for variable
c           1 are being requested.
c     ldim(1), ldim(2), etc.:  return the lengths of dimensions 1, 2, 3,
c           and 4, respectively.
c     isize:   returns the size of the array needed to accomodate the
c           retrieved data (which may be different from the number
c           of elements stored in the file for this variable).

c     ------------------------------------------
c     Allocate memory for array.
c     Note: some platforms may have slightly different functions
c         for allocating memory dynamically.

      ptadata = malloc(isize*4)

c     where
c     ptadata:  is a pointer to array adata as declared at the beginning
c           of this program.
c     isize:    specifies how many words to allocate for
c           array adata.

c     ------------------------------------------
c     Inform EzGet of array size.

      call defmisc('data size', 'integer', isize)

c     ------------------------------------------
c     Extract variable 1 from file.
```

```
      call getfield(1, adata)

c     where
c     1:    The first argument specifes that data should be retrieved
c           from variable 1.
c     adata:  returns the extracted data.

c     ----------------------------------------
c     Close all files opened by EzGet.

      call closeget

c     ----------------------------------------
c     Find the maximum and minimum values retrieved (skipping "missing"
c     values).

      rmax = -1.e40
      rmin =  1.e40

      do 200 n=1,isize
         if (abs(adata(n)-1.0e20) .gt. 1.e15) then
            rmax = amax1(rmax, adata(n))
            rmin = amin1(rmin, adata(n))
         endif
  200 continue

c     ----------------------------------------
c     Release memory allocated for adata.
c     Note: some platforms may have slightly different functions
c         for releasing memory.

      err = free(ptadata)

c     where
c     ptadata:  is the pointer to array adata, as declared at the
c     beginning of this program.

c     ----------------------------------------
c     Report the structure of the data and the maximum and minimum
c     values stored.

      write(*,'("         Data Structure" /
     &          " Dimension Name       Length " / /
     &      4(a16, i10 /))')
     &      (dimnames(n), ldim(n), n=1,ndim)
```

```
      if (rmax .ge. rmin) then
        write(*,'(" Maximum value found: ", 1pe14.5)') rmax
        write(*,'(" Minimum value found: ", 1pe14.5)') rmin
      else
        write(*,'(" Data set includes only ''missing'' data.")')
      endif

      end
```

2.8 Another example of area-averaging

One of the advantages of retrieving data with EzGet is that an array of weights can be generated proportional to the area of the grid cells (and in the following example also proportional to the number of days in each month). This makes it easy to compute annual averages and area averages. Suppose, for example, you want to compute the global mean difference between modeled and observed annual mean temperature. The array of weights can be used to select only those regions of the globe where data are available for every month of the year. Then the same weights can be used to weight the data appropriately.

There is more than one way to proceed, but in the following program the model output is mapped to the grid of the observations before masking out regions of missing data.

```
      program meandiff

c   This program
c
c    ** retrieves observed and model-simulated monthly surface air
c       temperature for the year 1979 (first year of AMIP run).
c    ** determines which grid cells are not missing any data and then
c       for these grid cells
c    ** computes the global average, annual mean difference between the
c       observed and model-simulated temperatures.
c
c       The data contributing to the averages are area-weighted and also
c       weighted by the number of days in each month.

      parameter (nlon=100, nlat=50, nmon=12, n4=0)
```

```
      real datamodl(nlon,nlat,nmon), wtsmodl(nlon,nlat,nmon),
     &     dataobs(nlon,nlat,nmon), wtsobs(nlon,nlat,nmon),
     &     wtobs(nlon,nlat)
      double precision asum, wtsum
      integer lons, lats, mons, i4, i, j, m, mm

c     ----------------------------------------
c     Initialize EzGet and define "missing" data value:

      call initget
      call defmisc('input missing value', 'real', 1.0e20)

c     ----------------------------------------
c     Define variable 1 as the observed surface air temperature and
c     indicate path/filename where data are stored.

      call defvar(1, 'tas', '/scratch/staff/lisa/amip_obs/nasa-amip_t')

c     ----------------------------------------
c     Define domain for variable 1.

      call defdim(1, 1, 'longitude', 'width', 'range',
     &                                          -180.0, 180.0, 360.0)
      call defdim(1, 2, 'latitude', 'cosine', 'range', -90.0, 90.0, 0.0)
      call defdim(1, 3, 'time',    'month', 'nearest', 109., 120., 0.0)

c     ----------------------------------------
c     Extract variable 1 (observed field).

      lons = 0
      lats = 0
      mons = 0
      i4 = 0
      call getdata(1,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsobs, dataobs)

c     ----------------------------------------
c     Define variable 2 as the modeled surface air temperature and
c     indicate path/filename where data are stored.

      call defvar(2, 'tas', '/amipsp/drs/tas/tas_bmr')

c     ----------------------------------------
c     Define domain for variable 2.
```

```
      call defdim(2, 1, 'longitude', 'width', 'range',
     &                                          0.0, 0.0, 360.0)
      call defdim(2, 2, 'latitude', 'gaussian', 'range',
     &                                           0.0, 0.0, 0.0)
      call defdim(2, 3, 'time',    'month', 'nearest', 109., 120., 0.0)

c     ----------------------------------------
c      Regrid model output to observed grid

      call defregrd(2, 'to', 1, 'area-weighted', 0,0.0,0.0, 0,0.0,0.0)

c     ----------------------------------------
c     Extract variable 2 (modeled field).

      call getdata(2,  nlon,nlat,nmon,n4,  lons,lats,mons,i4,
     &             wtsmodl, datamodl)


c       ----------------------------------------
c       Compute annual mean at each grid cell, but only if all 12 months
c       of data are available for both the model and the observations.

c       If data is available, weight by the area of the grid cell and
c       the number of days in a month.

        do 300 j=1,lats
          do 200 i=1,lons

            mm = 0
            do 100 m=1,mons
              if (wtsmodl(i,j,m)*wtsobs(i,j,m) .gt. 0.0) mm = mm + 1
  100       continue

            if (mm .eq. 12) then
              wtobs(i,j) = wtsobs(i,j,1)
            else
              wtobs(i,j) = 0.0
            endif

  200     continue
  300   continue

c     ----------------------------------------
c     Compute global average, annual mean difference
```

```
      asum = 0.0
      wtsum = 0.0

      do 600 m=1,nmon
        do 500 j=1,nlat
          do 400 i=1,nlon

            asum = asum + wtobs(i,j)*(datamodl(i,j,m)-dataobs(i,j,m))
            wtsum = wtsum + wtobs(i,j)

  400     continue
  500   continue
  600 continue

c     ----------------------------------------
c     Write area-weighted annual mean difference between model and
c     observed fields.

      if (wtsum .gt. 0.0) then

        asum = asum/wtsum

        write(*,'("annually-averaged fraction of globe with data: ",
     &           f12.7)') wtsum
        write(*,'("global, annual mean difference: ", f12.3)') asum

      else

        write(*,'(
     &  " data missing everywhere for at least 1 month of the year")')

      endif

c     ----------------------------------------
c     Close all files opened by EzGet.

      call closeget

      end
```

3 EzGet Subroutines

The EzGet subroutines that you will most likely need to call are documented in this section. First brief descriptions of the alphabetically ordered EzGet subroutines are provided:

`closeget` .... This subroutine closes any files opened by EzGet.

`clrtable` .... This subroutine clears a "dimension table" internal to EzGet. This may be necessary from time to time to make room for new dimension information.

`defdim` .... This subroutine specifies both the dimension ordering for data that will be retrieved and the desired domain. It also provides information for creating an array of "weights," typically set proportional to the grid-cell areas.

`defdimi` .... This subroutine is similar to `defdim`, but the domain is specified by index, rather than by coordinate range.

`defgeog` .... This subroutine is called to select data from certain geographical regions, so that when a field is retrieved, all data outside those regions are masked out.

`defmisc` .... This subroutine ('define miscellaneous') can be used to override certain default parameters assumed by EzGet (e.g., the value used to identify missing data, a parameter controlling the reporting of EzGet error messages, etc.).

`defregrd` .... This subroutine controls mapping of data to a specified grid.

`defvar` .... This subroutine defines a variable that will be referenced subsequently by a simple integer index.

`defvarex` .... This subroutine is similar to `defvar` but must be used if more than one variable with the same name is stored in a file.

`domain` .... This subroutine retrieves from a file the number of dimensions, the names of the dimensions and the full domain of each dimension of a field (as originally stored) for any variable.

`domlimit` .... After data have been retrieved (or after `shape` has been called), this subroutine returns the domain limits of a specified dimension (which are determined by your specifications through calls to `defdim`).

`getcoord` .... After data have been retrieved (or after `shape` has been called), this subroutine returns a vector of coordinate values of a specified dimension.

`getdata` .... This subroutine retrieves data, possibly mapping it to a different grid and masking user-specified geographical regions, and creates an appropriate mask (or set of "weights") associated with the data. It differs from `getnogap` in that the data are put into a multidimensional array structure and do not necessarily occupy contiguous memory.

`getdimwt` .... After data have been retrieved (or after `shape` has been called), this subroutine returns a vector containing the weights associated with a specified dimension.

`getedges` .... After data have been retrieved (or after `shape` has been called), this subroutine returns a vector containing the locations of the grid cell edges (i.e., grid cell boundaries) for a specified dimension.

`getfield` .... This subroutine simply retrieves data, but does no masking or mapping to a different grid.

`getgeog` .... This subroutine creates a geography mask for a specified region. Normally you do not need to create this mask as a separate step because `defgeog` will already have made it possible to select the desired geographical regions.

`getnogap` .... This subroutine retrieves data, possibly mapping it to a different grid and masking user-specified geographical regions, and creates an appropriate mask (or set of "weights") associated with the data. It differs from `getdata` in that the data are forced to occupy contiguous memory.

`initget` .... This subroutine must be called to initialize EzGet. It assigns default values to a few parameters and sets up some internal tables.

`lendims` .... After retrieving data, this subroutine returns the length of each dimension of a variable that has been retrieved by EzGet. It would typically be called after retrieving data with `getnogap` or `getfield`.

`shape` .... Before retrieving data, this subroutine returns the length of each dimension of a field as it will be retrieved by EzGet.

`varinfo` .... This subroutine returns descriptive information retrievable from the file containing a defined variable. Information retrievable includes the data source, title, units, date, time, and variable-type.

3.1 Subroutine closeget

This subroutine closes any files opened by EzGet and prints out a warning message if any errors or warnings were encountered by EzGet. It is generally a good idea to call this subroutine before a program ends. Subroutine `closeget` has no arguments, so it is called as follows:

```
call closeget
```

Note that while EzGet is active (i.e., after `initget` is called and before `closeget` is called), unit numbers 90–94 and 95–96 are reserved for EzGet. (See Section 4.1 for further information.)

3.2 Subroutine clrtable

This subroutine clears a "dimension table" internal to EzGet. The dimension table may become filled after too many fields with different coordinate dimensions have been retrieved by EzGet, and it may be necessary to make room for new dimension information.[2] You will learn of this necessity by an explicit error message transmitted by EzGet, so you may choose to wait for such a message before including a call to `clrtable`. Subroutine `clrtable` has no arguments, so it is called as follows:

```
call clrtable
```

Once the dimension table has been cleared, it will be impossible to obtain the information about a variable previously accessed by EzGet, which would normally be retrievable by subroutines `domlimit`, `lendims`, `getcoord`, `getedges`, and `getdimwt`.

3.3 Subroutine defdim

This subroutine specifies the dimension ordering and the desired domain for data that will be retrieved. It also is called to specify information for creating an array of "weights," typically set proportional to the grid-cell areas. *After creating the set of weights as specified by calls to* `defdim`*, EzGet may reset weights to zero where data are missing or have been masked* (e.g., if only a limited geographical region has been selected through a call to subroutine `defgeog`). A call to `defdim` is of the form:

```
 call defdim(var-index, dim-position, dim-name, weight-type,
&            domain-type, bdry1, bdry2, dcycle)
```

where

`var-index` (integer) specifies which variable (defined by calling `defvar` or `defvarex`) will be provided with dimension information by this call to `defdim`. If `var-index` is set to 0, then the dimension information will be assigned to all variables.

`dim-position` (integer) specifies which dimension will be defined by this call to `defdim`. When the data are retrieved, the ordering of the dimensions is determined by this number. For example, if `dim-position` has a value of 2, then the data will be retrieved such that this is the second dimension. (Note that, following the FORTRAN convention, the first dimension varies most rapidly as we move through contiguous storage.) When EzGet is initialized, all dimensions

[2]The dimension table contains room for 100 different coordinate dimensions, so it normally will become filled only when processing many different models.

are designated as being 'not defined'. You may want to reset all dimensions to their 'not defined' state, which can be done by calling `defdim` with `dim-position` set to 0. You may also 'undefine' a single dimension by setting `dim-position` to a negative value. For example if `dim-position` is assigned a value of -2, then the second dimension will be reset to 'not defined'. When `dim-position` is assigned 0, -1, -2, -3, or -4, the other arguments in the subroutine call are ignored (except, of course, `var-index`).

`dim-name` (character string) is the name of the dimension being defined.

`weight-type` (character string) controls the creation of the weighting-factor associated with the dimension. The total weight for a grid cell is the product of the weighting-factors for each of the grid-cell's dimensions. If data are missing or masked out, then the corresponding weights are set to 0. The following options are available for this parameter:

- `'unit'` — weight by 1 each grid-cell within the domain retrieved. This differs from `'equal'` in that here the sum of the weights is equal to the number of grid-points in the retrieved domain of the dimension (before masking).
- `'equal'` — weight equally each grid-cell within the domain retrieved. This differs from `'unit'` in that here the sum of the weights over the full stored domain is equal to 1 (before masking). If only a fraction of the full domain is retrieved, then the sum of the weights will equal this fraction.
- `'width'` — Set weights proportional to the width of each grid-cell, assuming the grid cells are equally spaced (i.e., assuming the grid-cell boundaries are half-way between the grid-cell centers). (Note, however, the edge of a pressure dimension is assumed to be greater than or equal to 0.) The weights generated are normalized by the total width of the domain that was originally stored. If a fraction of that domain is retrieved, then the sum of the weights will equal that fraction. (For a longitude dimension, the normalization is carried out under the assumption that the stored data spans 360°, even when it does not.)
- `'bmr'`, `'ccc'`, `'col'`, etc. — can be specified only for longitude or latitude dimensions and will create appropriate weights proportional to grid cell area for each of the AMIP or PMIP models. See Appendix A for a table identifying the model acronyms recognized by EzGet.
- `'gaussian'` — weight each grid-cell assuming the dimension represents gaussian latitudes. Note that in creating gaussian weights, EzGet assumes as a default that in the file containing the data to be retrieved, the number of latitude grid-points spans the globe. For T21 resolution, for example, the number of latitude grid points stored in the file should be 32. If, in fact, data from some nonglobal latitude domain have been stored, then you must specify that the data have been stored on a subdomain of a T21 grid by calling subroutine `defmisc` as described in Section 3.6.

`'cosine'` — weight by the cosine of latitude (i.e., $|\sin(\texttt{edge1}) - \sin(\texttt{edge2})|$, where `edge1` and `edge2` are edges of a latitude grid-cell, assumed to be half-way between grid-cell centers).

`'month'` — weight by the number of days in a month. This assumes that the time dimension has units of months with January referenced by 1., 13., 25., etc., February referenced by 2., 14., 26., etc. The sum of the weights for a full year of data will be 1. This weighting works correctly only for data from a 365-day year.

`'leapyr'` — weight by the number of days in a month. This assumes that the time dimension has units of months with January referenced by 1., 13., 25., etc., February referenced by 2., 14., 26., etc. The sum of the weights for a full year of data will be 1. This weighting works correctly only for data from a 366-day year.

`domain-type` (character string) is the argument determining the extent of the domain that will be retrieved. The following options are available for this parameter:

`'as saved'` — retrieve the full domain for this dimension in the same order that it appears in the file.

`'range'` — retrieve data from the domain specified by `bdry1` and `bdry2`. If the weighting option (`weight-type`) is specified as `'width'`, `'gaussian'`, `'cosine'`, or a model acronym (e.g., `'bmr'`, `'ccc'`, etc.), then if *any part* of a grid cell falls within these boundaries, data will be retrieved from that grid cell (even if the actual coordinates of that grid cell lie outside the specified range). If the weighting option (`weight-type`) is specified as `'equal'`, `'unit'`, `'month'` or `'leapyr'`, then only data with coordinates that lie within the domain specified by `bdry1` and `bdry2` will be retrieved. (Note that in this case, if you specify a narrow range for the domain, smaller than the width of a grid cell, it is possible that no grid-cell centers will lie within the domain, and no data will be retrieved.) Under all weighting options, if all grid cells lie completely outside the domain requested, then no data will be retrieved.

`'nearest'` — retrieve data with coordinates within the range specified by `bdry1` and `bdry2`, except under the following two circumstances: 1) if no coordinates fall within `bdry1` and `bdry2`, then retrieve the plane of data nearest to the region specified by `bdry1` and `bdry2`, or 2) if `dcycle` is not 0.0 and exactly one complete cycle is specified (i.e., $|\texttt{bdry2} - \texttt{bdry1}| = \texttt{dcycle}$), then EzGet will shift the domain by up to 1/2 grid-cell in order to avoid splitting the grid-cell at the edge of the domain. (Examples will be given later in this section.)

`bdry1` (real) specifies the desired beginning boundary of the domain to be retrieved, but note that this argument is ignored if `domain-type` is set to `'as saved'`. If

`bdry1>bdry2`, then the order of data retrieval will be such that this coordinate *decreases* monotonically.

`bdry2` (real) specifies the desired ending boundary of the domain to be retrieved, but note that this argument is ignored if `domain-type` is set to `'as saved'`.

`dcycle` (real) indicates the period of cyclic coordinates such as longitude (e.g., if the units are degrees longitude, then -180.0 = 180.0 = 540.0 and the cycle interval is 360.0 degrees) or possibly the annual cycle (e.g., if the units are months, then 1 = 13 = 25, and the cycle interval is 12 months). With this information, EzGet can find data that appear to lie outside of the range specified by `bdry1` and `bdry2`, but in fact are simply assigned a different but equivalent coordinate value. For non-cyclic coordinates, dcycle should be set to 0.0. This argument is ignored if `domain-type` is set to `'as saved'`. In all cases of cyclic data only a *single* cycle should reside on the file (i.e., do not include any "wrap-around points"). This implies that if multiple years of monthly data reside on the file, you should always set `cycle` to 0.0, not 12.0, for the time dimension.

The assignments made by a call to `defdim` remain in effect unless you subsequently call either `defdim` or `defdimi` and redefine dimension `dim-position` for variable `var-index`, or unless you reinitializes EzGet with a call to `initget`. If you instruct EzGet to map data to a new grid and if the call to subroutine `defregrd` specifies `target-cntrl` as `'to'`, then `domain-type`, `bdry1`, and `bdry2` will be overridden and the domain will be that specified for the target grid (but which dimension varies most rapidly will still be determined by the `defdim` calls for the source data). If you instruct EzGet to map data to a new grid and if the call to subroutine `defregrd` specifies `target-cntrl` as `'uniform'` and `nlat>0` or `nlon>0`, then `domain-type`, `bdry1`, and `bdry2` will be overridden and the domain will be determined by the arguments in your call to `defregrd`.

At first you might be somewhat baffled as to what to specify for `weight-type` and `domain-type`, but the most usual choices are described here:

- *Latitude:* Usually you will want to extract a precisely defined domain (not necessarily coinciding with grid-cell boundaries) and create weights proportional to grid cell area. If this is the case then `domain-type` should be set to `'range'` and `weight-type` should be set to either `'cosine'` or `'gaussian'`, depending on the grid. In many cases a convenient way to generate area-weights is to set `weight-type` to the model acronym as given in Appendix A.

- *Longitude:* Usually you will want to extract a precisely defined domain (not necessarily coinciding with grid-cell boundaries) and create weights proportional to grid cell area. If this is the case then `domain-type` should be set to `'range'`

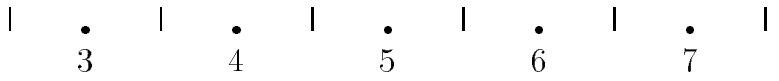
and `weight-type` should be set to either `'width'` or the model acronym as given in Appendix A. If, however, you want to retrieve data spanning 360 degrees, but without possibly splitting one grid cell in two, then `domain-type` should be set to `'nearest'`.

- *Time (month):* The most common choice for `domain-type` is probably `'nearest'` when monthly data are retrieved. In order to create weights appropriate for computing annual means from monthly data, specify either `'month'` or `'leapyr'`, depending on whether the year is of normal length or a leap year.[3] If, however, the model year comprises twelve 30-day months (rather than the realistically defined calendar months), then one would normally specify `'equal'`.
- *Level or Time (hour, day, year, etc.):* In this case the usual choice for `domain-type` will be `'nearest'`. You will usually want to assign equal weighting to each grid cell, so set `weight-type` either to `'unit'` or `'equal'`. Note that pressure (i.e., mass) weighting cannot be generated by EzGet. Also note that if a dimension is very long (more than 20,000 elements in the file from which you are retrieving data or more than 2000 elements actually retrieved, then `'unit'` weighting should be specified, unless the default lengths just quoted are overridden by a call to subroutine `defmisc` as described in Section 3.6.

If a given dimension of a variable (as it is stored) contains only 1 element, some of the above weighting factors will not work properly. For this special case, EzGet will override your specifications and assign this dimension `'unit'` (except if `'month'` or `'leapyr'` has been specified, in which case the weight will be equal to the number of days in the month divided by the number of days in the year).

If a given dimension of a variable (as it is stored) contains only 1 element, it is also not even necessary to call `defdim` to define this dimension. In this case EzGet will assume that you want the only domain available for this dimension. EzGet will extract the data and assign `'unit'` weighting to this dimension.

For further information concerning the difference between the various specifications of `weight-type` and `domain-type` consider the following. Suppose one dimension of an array of stored data has coordinates 3.0, 4.0, 5.0, 6.0, and 7.0 as illustrated below (with the boundaries of each grid cell indicated by vertical line segments and the coordinate positions indicated by the dots):

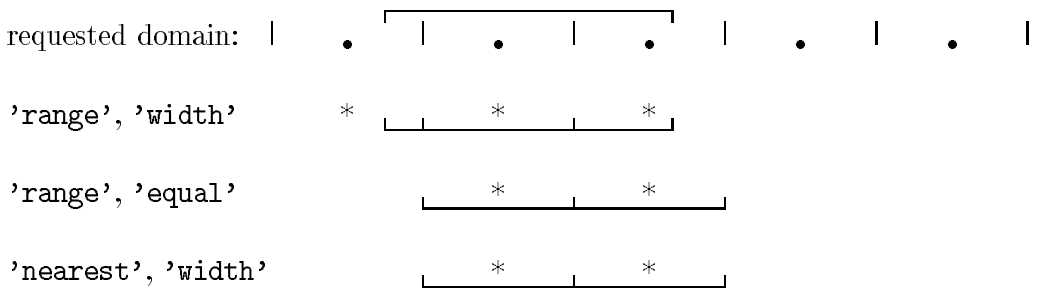


[3]Note that this only works when the time dimension being defined is in units of months with month 1, 13, 25, etc. corresponding to January.

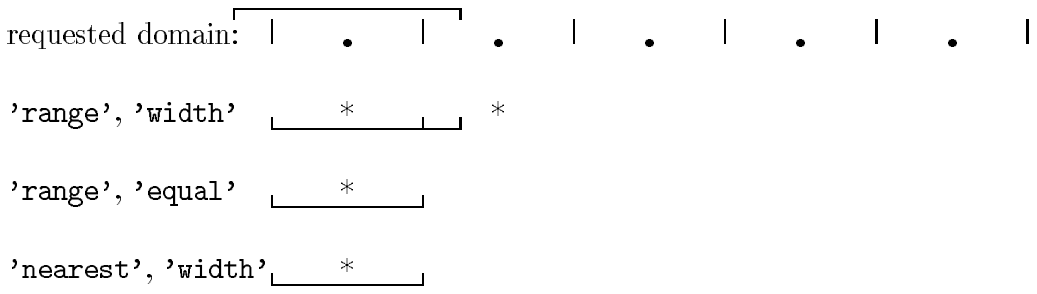
Unless `'gaussian'`, `'month'`, `'leapyr'`, `'csu'`, `'ucl'`, `'lmd'`, or `'lmd'` has been specified as the weighting option, EzGet assumes that the boundaries of the grid-cells lie half-way between the coordinates (i.e., at 2.5, 3.5, etc.). For this dimension suppose we want to extract a certain sub-domain of this data. The following examples indicate some options for controlling the domain that will be retrieved and the weights that will be assigned to each cell containing data. In these examples we assume no missing data and no masking.

In the following illustrations the domain you request by calling `defdim` is indicated by the over-bracket and the extracted grid-points are indicated by the symbols, '*'. The weights assigned each retrieved grid point are proportional to the intervals defined by the underbrackets. The illustrations show that the extracted grid-points and the weights depend on the values assigned to `weight-type` and `domain-type`:

Example 1:

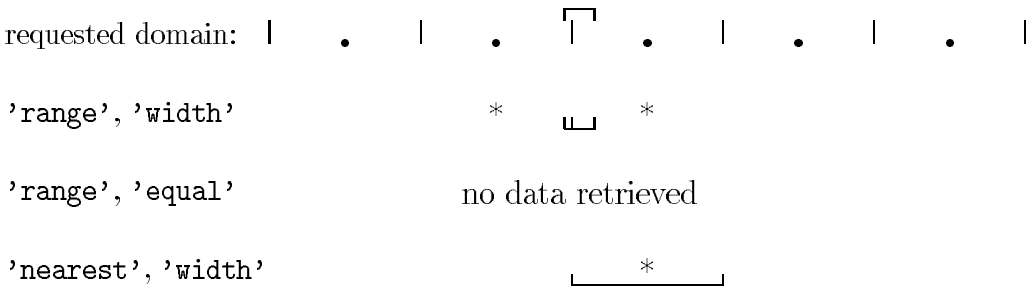


Example 2: No wrap-around permitted (i.e. you have specified `dcycle` to be 0.0).

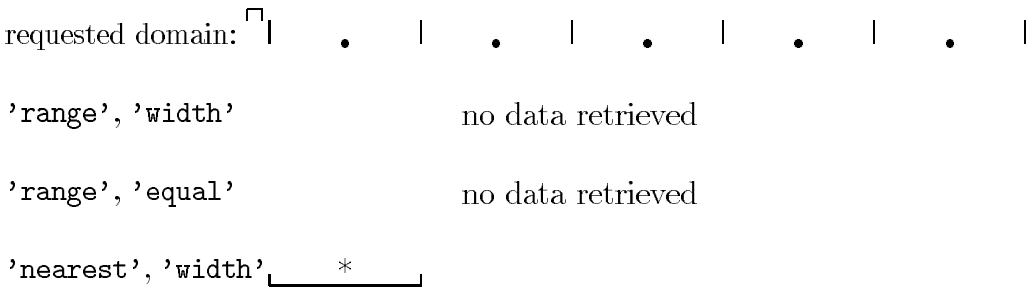


Example 3:

Example 4:

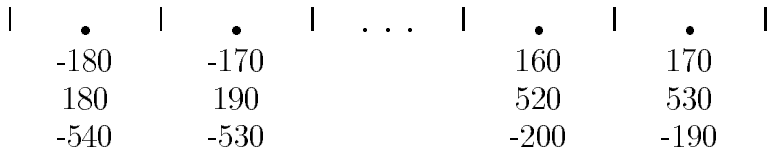


Example 5: No wrap-around permitted.



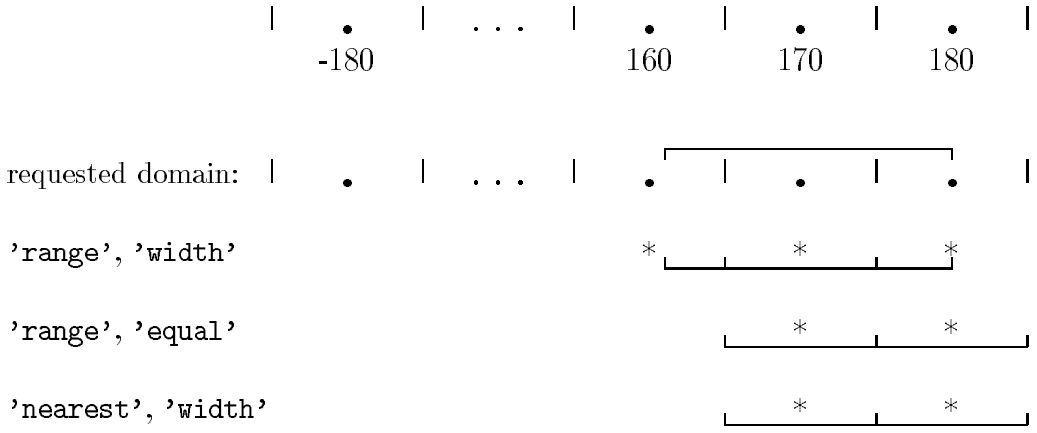
Suppose one dimension of an array of data is longitude with coordinates -180., -170., . . ., 160., 170. as illustrated below:

If the wrap-around option is in effect, then each grid-cell can be identified by more than one coordinate value. For example,



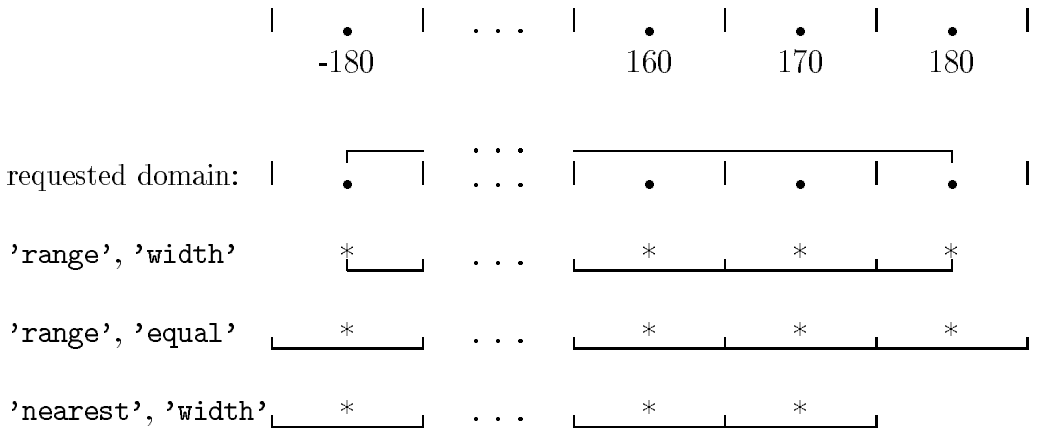
Example 6: Wrap-around activated (i.e., `dcycle` set to 360.0).

Suppose `bdry1` is assigned the value 162. and `bdry2` is assigned the value 180., then the wrap-around of data implies the following (because the last grid cell extends only to 175, the data is wrapped around, duplicating the first grid-cell in the last position):



Example 7: Wrap-around activated, complete cycle requested.

Suppose `bdry1` is assigned the value -180. and `bdry2` is assigned the value 180., then the wrap-around of data implies the following:



Note that when `'range'` and `'width'` are specified, data are extracted covering the full domain (from -180 to 180), which requires one grid cell to be split into two, each given half the weight of the original. To avoid splitting grid cells (and to extract a complete cycle of 360°), `'nearest'` should be specified in which case the domain extracted is shifted slightly to the interval -185 to 175.

3.4 Subroutine defdimi

This subroutine is similar to subroutine `defdim` but the domain is specified by index, rather than by a coordinate range. A call to `defdimi` is of the form:

```
 call defdimi(var-index, dim-position, dim-name, weight-type,
&             indx1, indx2)
```

where

`var-index, dim-position, dim-name`, and `weight-type` are defined exactly as in subroutine `defdim`.

`indx1, indx2` (integers) specify by index the first and last planes of data that should be retrieved. For example, if `indx1` and `indx2` are specified as 3 and 5, respectively, then the third, fourth and fifth planes of dimension `dim-name` will be retrieved. If `indx1` and `indx2` are specified as 5 and 3, respectively, then the fifth, fourth and third planes of dimension `dim-name` will be retrieved (in that order). If `indx1` and `indx2` are specified as -5 and -3, respectively, then the fifth, fourth and third planes *from the end* of the domain will be retrieved (i.e., `jj-4, jj-3`, and `jj-1`, where `jj` is the length of the dimension. And as a final example, if `indx1` and `indx2` are specified as -3 and -5, respectively, then the third, fourth and fifth planes *from the end* of the domain will be retrieved.

3.5 Subroutine defgeog

This subroutine is called in order to specify the geographical regions (within the domain defined by calls to `defdim`), from which data will be retrieved. All data outside the selected regions will be masked out (i.e., the "weights" will be set to 0.0 and the data will be set to the "missing value").

For selection of geographical regions, an input data set must be provided, which may either be a simple land-ocean-sea ice mask (or even more simply, a land-sea mask), a land fraction (expressed as a percent), a sea ice fraction (expressed as a percent), or a more detailed geographical data set uniquely defining regions typically the size of a continent (as described in more detail below). Appendix B contains the list of regions identified on the standard geographical maps for AMIP and PMIP models available from PCMDI. The grid on which the geography mask is stored must be identical to either the grid of the source data being retrieved or a new grid to which the data will be mapped.

A call to `defgeog` is of the form:

```
call defgeog(var-index, inorout, mask-index, select)
```

where

`var-index` (integer) specifies to which variable (defined previously or subsequently by a call to `defvar` or `defvarex`) the geography mask will be applied.

`inorout` (character string) indicates whether the geography mask should be applied before or after mapping data to a new grid. This argument should be assigned one of the following strings:

`'in'` — if the geography mask should be applied before mapping data to a new grid.

`'out'` — if the geography mask should be applied after mapping data to a new grid.

`mask-index` (integer) specifies the variable (defined previously or subsequently by a call to `defvar` or `defvarex`) that contains the geography data.

`select` (character string) is a list of regions, separated by commas, specifying which geographical regions of the globe should be selected (i.e., not masked). If the geography data simply indicates regions of land, ocean, and possibly sea-ice, then this argument might be, for example, `'ocean, sea ice'`, which would select all regions except land. If the geography data contains the more detailed geographical information, then this argument might be, for example, `'North America, South America, Greenland'`, which selects the major land areas of the Western Hemisphere. Note that the commas separating the listed regions may optionally be followed by blanks. For example, either `'Africa, Australia, Antarctica'` or `'Africa,Australia,Antarctica'` is acceptable.

Note that if you call `degeog` twice with `inorout` set to `'in'` for one call and `'out'` for the other call, then one geography mask can be applied before mapping to a new grid, and another mask can be applied after the mapping. The argument passed as `select` would usually be the same when selecting geography both before and after mapping to a new grid, but this is not required. You might, for example, select `'land'` before mapping and `'North America'` after mapping to the new grid.

Even when data remain on the original grid, two different masks can be applied by calling `defgeog` twice, once with `'in'` specified and once with `'out'` specified. You might want to do this, for example, in order to select ice free regions of the North Atlantic. You would call `defgeog` to select `'N. Atl. ocean'` (available from `'sftbyrgn'` variable) and then call `defgeog` a second time to eliminate regions of sea (by selecting `'ocean'` from the `'sft'` variable).

If this subroutine is not called, then EzGet will not select geographical regions (other than rectangular regions determined by the domain specifications). If after calling this

subroutine, you want to return to this default state of no masking, `defgeog` should be called, but with `mask-index` set to 0. If you had specified that masks be applied both before and after mapping the data to a new grid, then `defgeog` would have to be called twice with `mask-index` set to 0, once with `inorout` set to `'in'` and a second time with it set to `'out'`. When `mask-index` is set to 0, the last argument, `select`, is ignored.

Note that the geography must be at least a function of longitude and latitude. It may also be a function of time (and perhaps in unusual circumstances, level). If it is a function of time (and if the time-dimension is greater than 1), then the time-dimension should be consistent with the time dimension for the data that you want to retrieve. If the geography field has fewer dimensions than the array of data, then the same geography mask will be applied to all planes of the data array associated with the missing dimension(s). Suppose, for example, that the data being retrieved is a function of longitude, latitude and month, but the geography field is not a function of time. Then the same geography field will be applied to each month of data.

Four types of geography data can be used by EzGet: land-ocean-sea ice masks (with sea ice optional), land fraction expressed as a percent, sea ice fraction expressed as a percent, or a more detailed geographical data set uniquely distinguishing regions typically the size of a continent. Data can either be stored as integers or floating point (real) numbers.

For land-ocean-sea ice type geography data, EzGet should find either the integers, 0, 1, and 2, or the floating point (real) numbers, 0.0, 1.0, and 2.0, stored for land, ocean, and sea-ice grid cells, respectively. In this case you should select geographical regions by specifying `'land'` and/or `'ocean'` and/or `'sea ice'` in a call to subroutine `defgeog`.

For land fraction surface-type information (expressed as a percent), EzGet should find either integers or floating point (real) numbers in the range 0 to 100. To use land fraction data of this type for the purpose of masking data retrieved by EzGet, you should either make sure the name of the variable is `'sftl'` or `'sftland`, or you should call `defmisc` with the first argument set to `'mask type'` and the last argument set to `'1'`. You should select geographical regions by specifying `'land'` or `'ocean, sea ice'` in a call to subroutine `defgeog`.

For sea ice fraction surface-type information (expressed as a percent), EzGet should find either integers or floating point (real) numbers in the range 0 to 100. To use sea ice fraction data of this type for the purpose of masking data retrieved by EzGet, you should either make sure the name of the variable is `'sifrc'` or `'seaice'` or you should call `defmisc` with the first argument set to `'mask type'` and the last argument set to `'2'`. You should select a region by specifying `'sea ice'` (which will select regions of sea ice) or `'land, ocean'` (which will mask regions of sea ice) in a call to subroutine

`defgeog`.

For the more detailed geography data, the table in Appendix B shows the integers that EzGet assumes have been stored in the geography file to define the various geographical regions.

3.6 Subroutine defmisc

This subroutine ('define miscellaneous') can be used to override certain default parameters assumed by EzGet (e.g., the value used to identify missing data, a parameter controlling the reporting of EzGet error messages, etc.). A call to `defmisc` is of the form:

```
call defmisc(param-name, strng-param, numer-value)
```

where

`param-name` (character string) indicates which parameter to define. The options are:

- `'input missing value'` — to assign the value you expect missing data to be identified by on files you are reading. For example, `call defmisc('input missing value', 'integer', -99)` instructs EzGet to interpret as missing any data point assigned the value -99. For real data, the second argument should be assigned the value, `'real'`. (If this parameter is not set, the data that will be retrieved are assumed to be floating point (real) numbers and the value indicating missing data is 1.0e20.)
- `'output missing value'` — to specify the value you want missing data to be assigned after it is retrieved. For example, `call defmisc('output missing value', real, 0.0)` instructs EzGet to assign the value 0.0 to any data points that are missing. For integer data, the second argument should be assigned the value, `'integer'`. (If this parameter is not set, the data that will be retrieved are assumed to be floating point (real) numbers and the default value assigned to missing data is 1.0e20.)
- `'mask type'` — to indicate the type of data that is contained in geography files used for masking data. If `numer-value=1` then EzGet will assume that land fraction data (expressed as a percent) can be found in geography files used for data masking. If `numer-value=2`, then EzGet will assume that sea ice fraction data (expressed as a percent) can be found in geography files used for data masking. If `numer-value=3`, then EzGet will assume that land-ocean-sea ice data are stored (identified by either 0's, 1's, and 2's, or 0.0's, 1.0's, and 2.0's). If `numer-value=4`, then EzGet will assume more detailed geographical mask is stored (see Appendix B). If `numer-value=0`

(the default), then EzGet will determine what kind of data are stored in the geography files, based on the name of the variable – if `'sftl'` or `'sftland'`, then land fraction (expressed as a percent); if `'sic'` or `'sea ice'`, then sea ice fraction (expressed as a percent). If the variable name is not one of these (`'sftl'`, `'sftland'`, `'sic'`, or `'sea ice'`), then EzGet will determine the type of geography data based on the values stored in the file. EzGet can automatically determine whether simple land-ocean-sea ice data are stored (0's, 1's, and 2's, or 0.0's, 1.0's, and 2.0's), or whether the more detailed regional geographical data are stored, but EzGet cannot always automatically determine that the data are land or sea ice fractions. In the case of land fraction (expressed as a percent) the variable should be named either `'sftl'` or `'sftland'`, and for sea ice fraction, either `'sic'` or `'sea ice'`; otherwise you should call `defmisc` as just described. For example, if the variable containing land fraction data (expressed as a percent and stored as an integer) is named `'landpcnt'`, then you could `call defmisc('mask type', 'integer', 1)` so that EzGet could properly interpret the data and use it in masking land or ocean areas. The type of geography data expected will remain the same until `defmisc` is called again to override it.

`'mask type in'` — to indicate the type of data contained in geography files used for masking data before it has been regridded. This parameter along with the parameter `'mask type out'` make it possible to specify that different types of masks be used before and after regridding. See the discussion directly above for proper specification of `numer-value`.

`'mask type out'` — to indicate the type of data contained in geography files used for masking data after regridding. This parameter along with the parameter `'mask type in'` make it possible to specify that different types of masks be used before and after regridding. See the discussion above for proper specification of `numer-value`.

`'longitude name'` — to assign an alias for the name EzGet will interpret as the longitude dimension. In this case `numer-value` is ignored and the longitude name (a character string) is passed as the second argument (`strng-param`). If this parameter is not set, the default value is `'longitude'` (which already has aliases such as `'lon'`, `'Longitude'`, and `'LONGITUDE'` because EzGet is generally case insensitive and often only checks the first three characters of a string for equivalence). For example, if data have been stored with the longitude dimension named `'x'`, then `defmisc('longitude name', 'x', dummy)` should be called to inform EzGet that `'longitude'` is also known as `'x'`. (Note that the third argument is ignored by EzGet.) EzGet needs to know which dimension (if any) is longitude in order to correctly trap errors. (If this parameter is not set, the default value is `'longitude'`).

`'latitude name'` — to assign an alias for the name EzGet will interpret as the latitude dimension. In this case `numer-value` is ignored and the latitude

name (a character string) is passed as the second argument (`strng-param`). If this parameter is not set, the default value is `'latitude'` (which already has aliases such as `'LATITUDE'`, `'Latitude'`, and `'lat'` because EzGet is generally case insensitive and often only checks the first three characters of a string for equivalence). For example, if data have been stored with the latitude dimension named `'y'`, then `defmisc('latitude name', 'y', dummy)` should be called to inform EzGet that `'latitude'` is also known as `'y'`. (Note that the third argument is ignored by EzGet.) EzGet needs to know which dimension (if any) is latitude in order to correctly trap errors. (If this parameter is not set, the default value is `'latitude'`).

`'data size'` — to provide EzGet with the declared size of the data arrays passed to subroutine `getfield`. In this case `param- value` should be an integer. It is recommended that this parameter be set before calling subroutine `getfield`. (If this parameter is not set, EzGet will not error exit if the data actually retrieved by this subroutine exceeds the declared array size. For example, if the total size of the array of data that will be retrived is expected to be 1200, then `call defmisc('data size', 'integer', 1200)`. See Section 3.16 for further information.)

`'maximum dimension kept'` — to limit the size of dimensions that will be stored by EzGet in a table for later retrieval. In this case `numer-value` should be an integer. The size limit applies to the dimension length extracted by EzGet, not the dimension as it appears in the original file. (Compare with the `'longest dimension'` option below.) Any dimension longer than this value must be specified as having `'unit'` weighting. Furthermore, if the length exceeds the limit, you will be unable to obtain the values of the coordinates, weights or grid-cell edges by calling `getcoord`, `getedges`, or `getdimwt`. All longitude and latitude dimensions used in mapping data to a new grid or associated with geography files must be shorter than the limit. (If this parameter is not set, the default value is 2000. To double this, for example, `call defmisc('maximum dimension kept', 'integer', 4000)`. If this limit is made too large (greater than about 16000) there is a risk that an absolute EzGet dimension limit will be exceeded.)

`'longest dimension'` — to reduce the storage required when accessing unusually long dimensions (usually associated with very long time-series). In this case `numer-value` should be an integer. If a dimension is longer than the limit specified by 'longest dimension', then you must specify `'unit'` weights for this dimension. The size limit applies to the number of elements stored in the original file, not the number of elements extracted for this dimension (compare with the `'maximum dimension kept'` option above). If the length exceeds the limit, you will be unable to obtain the values of the coordinates, weights or grid-cell edges by calling `getcoord`, `getedges`,

or `getdimwt`. All longitude and latitude dimensions used in mapping data to a new grid or associated with geography files must be shorter than the limit. (If this parameter is not set, the default value is 20000.)

`'truncation'` — to indicate to EzGet how many latitude zones span the globe from pole to pole for a spectral model. EzGet may need to know this in order to create correct gaussian weights when data being retrieved have been stored on a non-global domain. In this case `numer-value` should be an integer. If EzGet retrieves data from a non-global domain, it is not necessary to set this parameter as long as the source data were stored on a global domain. (If this parameter is not set, EzGet assumes that the original latitude dimension spans the globe and creates gaussian weights accordingly.)

`'error control'` — to indicate how many errors detected by EzGet will be allowed before halting program execution and also whether or not error messages will be displayed. If `numer-value` (an integer in this case) is less than or equal to 0, then no messages will be displayed. If `numer-value` equals 0, the program execution will not be halted by EzGet no matter how many errors are encountered. If `numer-value` is greater than 0, then warnings and error messages will be displayed and execution will be halted when the the number of errors encountered equals the integer specified as `numer-value`. (If this parameter is not set, EzGet will print errors and warnings and will halt execution after encountering 2 errors.)

`'version'` — to request that EzGet print (to your terminal) the EzGet version number and date of release – `call defmisc('version', ' ', dummy)`. (Note that both the second and third arguments are ignored by EzGet.)

`strng-param` (character string) indicates what type of variable (`'real'` or `'integer`) will be passed as the third argument (`numer-value`), or in the case of `param-name='longitude name'` or `param-name='latitude name'` contains an alias for the longitude or latitude dimensions, respectively. If, for example, `numer-value` is a floating point (real) number (as it might be, for example, if `param-name` were `'input missing value'`), then `strng-param` should be set to `'real'`.

`numer-value` (real or integer) passes the value that will be assigned to the parameter selected through the first argument (`param-name`), except in the case of `param-name='longitude name'` or `param-name='latitude name'` in which case the second argument (`param-name`) contains this information.

3.7 Subroutine defregrd

This subroutine controls mapping of data to another grid. The target grid can be

defined by reference to another defined variable, or you can define it as a regular or gaussian grid by passing the appropriate subroutine arguments. Currently EzGet maps data to the target grid using an area-weighting alogrithm, which preserves area averages of the field. For correct mapping, you must define the spatial dimensions (with calls to `defdim`), correctly specifying the `weight-type` (for longitude, typically `'width'` and for latitude, typically `'cosine'` or `'gaussian'`, or in either case simply by specifying the acronym for one of the models appearing in Appendix A).

When this subroutine has been called, the data will be mapped to the target grid before returning it to you via `getdata` or `getnogap`. If the target grid is specified by an index to a defined variable, then the domain retrieved is determined by the domain specified for that defined variable (and the domain of the source grid is ignored).[4] The type of weighting appropriate to the source grid and the specification of which dimension of the retrieved array varies most rapidly are always determined by the calls to `defdim` for the *source* data set. If you specify the target grid through the arguments of subroutine `defregrd`, then the domain is either obtained indirectly from those arguments or from the domain specified for the source grid, as discussed further below.

A call to `defregrd` is of the form:

```
call defregrd(var-index, target-cntrl, target-index, method,
                        nlat, alat, dellat, nlon, alon, dellon)
```

where

`var-index` (integer in the range 1 to 10) specifies which variable (defined previously or subsequently by a call to `defvar`), will be mapped to the new grid.

`target-cntrl` (character string) indicates how the target grid will be defined. The options are:

- `'none'` — to turn off all mapping specifications so that the variable will remain on the source (i.e., original) grid.
- `'to'` — to map data to the grid of the variable identified by `target-index`. The domain is determined by the calls to `defdim` for the variable referenced by `target-index`, so that you will receive the data as if it were originally stored on the target grid.
- `'uniform'` — to map data to a user-defined grid that is evenly spaced in latitude and evenly spaced in longitude. The grid is defined by the last six arguments in the subroutine call (described below).
- `'gaussian'` — to map data to a user-defined gaussian grid. The grid is defined by the last six arguments passed to this subroutine (as described below).

[4] Note that the domain will exactly coincide with the range for the target grid, as if the source data had actually been stored on the target grid.

`target-index` (integer in the range 1 to 10) specifies which variable defines the target grid. The latitude and longitude coordinates as well as the latitude and longitude domain will be taken from the variable referenced by `target-index`, as if the source data were actually stored on the target grid. If `target-index` = 0, then no mapping will be done. If `target-cntrl` has not been set to `'to'`, then this argument is ignored. Be careful not to set `target-index` to a defined variable that might itself be mapped to a different grid, or you may encounter an error.

`method` (character string) specifies the interpolation method that should be used to map data to the new grid. Currently the only available method is an area-weighting algorithm, so `method` should be set to `'area-weighted'`.

`nlat` (integer) controls the latitude domain of the target grid. To generate a gaussian grid, set `nlat` to the number of latitude grid cells spanning the globe from pole to pole (and set `target-cntrl` to `'gaussian'`). In this case the actual domain will be determined by a call to `defdim` for the variable identified as `var-index`. To generate a uniformly spaced grid you may either: 1) set `nlat` to 0, in which case the domain will be deterermined by a call to `defdim` for the variable identified as `var-index`, or 2) set `nlat` to the number of latitude grid cells you want to generate. In this second case, the first grid cell will be located at `alat` and the last grid cell will be located at `alat+(nlat-1)*dlat` (and the domain will extend half a grid cell beyond these locations). Thus for a uniformly spaced grid, if `nlat>0`, any latitude domain specifications made by calls to `defdim` are overridden. If `target-cntrl` has been set to `'to'`, then the value assigned to `nlat` is ignored.

`alat` (real) specifies the location of one of the latitude grid cells, but is completely ignored for gaussian grids. If `nlat>0` and `dlat>0.0`, then `alat` will be the southern-most grid-cell generated; if `nlat>0` and `dlat<0.0`, then `alat` will be the northern-most cell. If `nlat=0` then one of the target grid cells generated will be located at `alat`, but the domain of the data and the order that it will be retrieved are specified by calls to `defdim` (so, for example, even if `dlat>0.0`, data will be retrieved from north to south if in your call to `defdim`, `bdry2` is specified to be less than `bdry1`). If `target-cntrl` has been set to `'to'` or if a gaussian grid is specified, then this argument is ignored.

`dlat` (real) is the distance between neighboring latitude grid cells, but is completely ignored for guassian grids. This argument is also ignored if `target-cntrl` has been set to `'to'`.

`nlon` (integer) controls the longitude domain of the target grid. You may either: 1) set `nlon` to 0, in which case the domain will be deterermined by a call to `defdim` for the variable identified as `var-index`, or 2) set `nlon` to the number of longitude grid cells you want to generate. In this second case, the first grid cell will be located at `alon` and the last grid cell will be located at `alon+(nlon-1)*dlon` (and the domain will extend half a grid cell beyond these locations). Thus if `nlon>0`,

any longitude domain specifications made by calls to `defdim` are overridden. If `target-cntrl` has been set to `'to'`, then the value assigned to `nlon` is ignored.

`alon` (real) specifies the location of one of the target longitude grid cells. If `nlon>0` and `dlon>0.0`, then `alon` will be the western-most grid-cell generated; if `nlon>0` and `dlon<0.0`, then `alon` will be the eastern-most cell. If `nlon=0` then one of the target grid cells generated will be centered at `alon`, but the domain of the data and the order that it will be retrieved are specified by calls to `defdim` (so, for example, even if `dlon>0.0`, data will be retrieved from east to west if in your call to `defdim`, `bdry2` is specified to be less than `bdry1`). If `target-cntrl` has been set to `'to'`, then this argument is ignored.

`dlon` (real) is the distance between neighboring longitude grid cells. If `target-cntrl` has been set to `'to'`, then this argument is ignored.

Although there are several ways to instruct EzGet to map data to a new grid as documented above, the four most common procedures are summarized here by way of example.

To map variable 1 (originally stored on a gaussian grid) to the grid of variable 2 (which is uniformly spaced in latitude and longitude):

```
 call defdim(1, 1, 'longitude', 'width', 'range', 0.0, 0.0, 360.0)
 call defdim(1, 2, 'latitude', 'gaussian', 'range', 0.0, 0.0, 0.0)
 call defdim(2, 2, 'longitude', 'width', 'range', 30.0, 60.0, 360.0)
 call defdim(2, 1, 'latitude', 'cosine', 'range', 23.5, 90.0, 0.0)

 call defregrd(1, 'to', 2, 'area-weighted',
&                      0, 0.0, 0.0, 0, 0.0, 0.0)
```

The domain is specified through the `defdim` calls for variable 2, which indicate that data should be retreived for the region bounded by 30° and 60° E longitude and 23.5° and 90° N latitude. Note that the *order* of the dimensions is determined by the `defdim` specifications for variable 1, so that the data will be ordered with the longitude index varying most rapidly.

To map variable 1 (originally stored on a regular grid) to a different, uniformly spaced 10° by 20° latitude-longitude grid, but for a domain limited to the Northern Hemisphere and with latitudes stored from north to south:

```
 call defdim(1, 1, 'longitude', 'width', 'range', 0.0, 0.0, 360.0)
 call defdim(1, 2, 'latitude', 'cosine', 'range', 0.0, 0.0, 0.0)

 call defregrd(1, 'uniform', 0, 'area-weighted',
&                           9, 85.0, -10.0, 18, 10.0, 20.0)
```

With this method of specifying the target grid, the domain boundaries coincide with grid cell edges (which lie half-way between the grid cell centers). In the above example, the domain extends from 90° to 0° N latitude and 0° to 360° E longitude. (The first latitude grid cell, for example, is 10° wide and its center is at 85° N, which means it extends from 90° N to 80° N.)

To map variable 1 to a regular (uniformly spaced) 10° by 20° latitude-longitude grid, but for a domain specified through calls to subroutine `defdim` (which will provide more precise domain control than in the previous example):

```
 call defdim(1, 1, 'longitude', 'width', 'range', 0.0, 90.0, 360.0)
 call defdim(1, 2, 'latitude', 'cosine', 'range', 0.0, 30.0, 0.0)

 call defregrd(1, 'uniform', 0, 'area-weighted',
&                            0, -85.0, 10.0, 0, 10.0, 20.0)
```

Note that unlike the previous example, the domain boundaries do not necessarily coincide with the edges of grid cells. In fact the last latitude grid cell (centered at 30 N) extends from 25 N to 35 N, but because the domain stops at 30 N, the weights assigned this grid cell will be proportional to the area of a 5° by 20° cell, whereas the other grid cells will be assigned weights proportional to the area of 10° by 20° cells.

To map variable 1 (originally stored on a uniformly spaced grid) to a gaussian grid at T42 resolution:

```
 call defdim(1, 1, 'longitude', 'width', 'range', 0.0, 90.0, 360.0)
 call defdim(1, 2, 'latitude', 'cosine', 'range', 0.0, 30.0, 0.0)

 call defregrd(1, 'gaussian', 0, 'area-weighted',
&                       64, 0.0, 0.0, 0, 0.0, 2.8125)
```

For gaussian grids the latitude domain is always set by a call to `defdim`, but the longitude domain is set by a call to `defdim` only if the number of longitudes specified in the call to `defregrd` is 0, as it is in the above example.

3.8 Subroutine defvar

This subroutine defines a variable that can be referenced subsequently by the integer index you pass to EzGet as the first argument. A call to `defvar` is of the form:

```
call defvar(variable-index, variable-name, file-name)
```

where

`variable-index` (non-zero integer in the range -10 to 10) is an index. The absolute value of this index will be used subsequently to identify the data defined by `variable-name` and `file-name`. If a positive value is passed to `defdim`, then EzGet error exits if the file or variable cannot be found. If a negative integer is passed, then EzGet returns the absolute value of `variable-index` if no errors are encountered, but EzGet returns -1000 if the file cannot be found, -1001 if the variable cannot be found, and -1002 if some other error is encountered.

`variable-name` (character string of no more than 64 characters) is the name of the variable as it appears in the file from which data will be retrieved.

`file-name` (character string of no more than 120 characters) is the full path (including file name) of the file that will be accessed by EzGet. (For files in DRS format, the '.dic' and '.dat' suffixes should be omitted, but for GrADS (or GRIB) files, the '.ctl' suffix should be included.)

If the name of the variable is not unique (i.e., more than one variable with the same name resides in the file being accessed by EzGet), then you must define the title and/or the source by calling subroutine `defvarex` as described below.

3.9 Subroutine defvarex

This subroutine is similar to `defvar` but must be used if more than one variable with the same name is stored in a file. In this case the "title" and/or the "source" of the data must be specified, so that EzGet can determine which field to retrieve. A call to `defvarex` is of the form:

```
call defvarex(var-index, var-name, var-title, var-source, file-name)
```

where

`var-index` (integer in the range 1 to 10) that will be used subsequently to identify the data defined by `var-name`, `var-title`, `var-source`, and `file-name`.

`var-name` (character string of no more than 64 characters) is the name of the variable as it appears in the file from which data will be retrieved.

`var-title` (character string of no more than 80 characters) is the title of the variable as it appears in the file from which data will be retrieved. If you specify `' '`, then the title will be ignored in determining which variable to retrieve.

`var-source` (character string of no more than 120 characters) is the source of the variable as it appears in the file from which data will be retrieved. If you specify `' '`, then the source will be ignored in determining which variable to retrieve.

`file-name` (character string of no more than 120 characters) is the full path (including file name) of the file that will be accessed by EzGet. (For files in DRS format, the '.dic' and '.dat' suffixes should be omitted, but for GrADS (or GRIB) files, the '.ctl' suffix should be included.)

3.10 Subroutine domain

This subroutine retrieves the number of dimensions, the names of the dimensions and the domain limits of each dimension *as they appear in the file you are accessing* with EzGet. The domain returned extends from the first coordinate value to the last coordinate value store in the file. This subroutine may be called after defining a variable (by `defvar` or `devarex`). A call to `domain` is of the form:

```
call domain(var-index, ndim, dimnames, beg, end)
```

where

`var-index` (integer in the range 1 to 10) specifies from which variable (defined by a calling `defvar` or `defvarex`) the domain information should be retrieved.

`ndim` (integer) returns the number of dimensions.

`dimnames` (character string, of 16 characters) returns a vector of length `ndim` that contains the names of the dimensions.

`beg` (real) returns a vector of length `ndim` that will contain the first coordinate value stored for each dimension.

`end` (real) returns a vector of length `ndim` that will contain the last coordinate value stored for each dimension.

3.11 Subroutine domlimit

This subroutine retrieves the domain limits of a single dimension and may be called after retrieving the data or calling subroutine `shape`. The domain limits as defined here extend from the leading edge of the first grid cell *retrieved* to the trailing edge of the last grid cell *retrieved*. A call to `domlimit` is of the form:

```
call domlimit(var-index, dimname, beg, end)
```

where

`var-index` (integer in the range 1 to 10) specifies from which variable (defined by calling `defvar` or `defvarex`) the domain information should be retrieved.

`dimname` (character string, limited to no more than 16 characters) specifies the name of the dimension for which the domain limits are being requested.

`beg` (real) returns the leading edge of the domain for dimension `dimname`.

`end` (real) returns the trailing edge of the domain for dimension `dimname`.

3.12 Subroutine getcoord

After data have been retrieved (or after `shape` has been called), this subroutine returns a vector of coordinate values of a specified dimension. The coordinate information refers to what has been or will be retrieved by EzGet, which may, for example, be a subset of what appears in the original file. A call to `getcoord` is of the form:

```
call getcoord(var-index, idim, coords)
```

where

`var-index` (integer in the range 1 to 10) specifies from which variable (defined by calling `defvar` or `defvarex`) coordinates should be retrieved.

`idim` (integer) is the dimension for which coordinates are being requested (1 for the first dimension of the variable, 2 for the second, etc.)

`coords` (real) is a vector that will receive the coordinates.

3.13 Subroutine getdata

This subroutine retrieves data, possibly mapping it to a different grid and masking user-specified geographical regions, and creates an appropriate mask (or set of "weights") associated with the data. It differs from `getnogap` in that the data are put into a multidimensional array structure and do not necessarily occupy contiguous memory. This subroutine checks whether the dimensions expected are correct or at least that the arrays are dimensioned large enough to accomodate the retrieved data.

A call to `getdata` is of the form:

```
 call getdata(var-index, ndim1, ndim2, ndim3, ndim4,
+        isiz1, isiz2, isiz3, isiz4, amask, field)
```

where, on entry to `getdata`,

`var-index` (integer in the range 1 to 10) specifies for which variable (defined by calling `defvar` or `defvarex`) data should be retrieved.

`ndim1, ... ndim4` (integers) should be identical to the dimensions of `amask` and `field`. If these arrays have fewer than 4 dimensions, the unused dimensions should be set to either 0 or 1. If set to 0, then an error message will be displayed if the field actually has more dimensions than allowed for. For example, if `field` is dimensioned `field(64,32)`, then you should specify `ndim1=64`, `ndim2=32`, `ndim3=0`, and `ndim4=0`. (Each of these arguments is left unaltered by EzGet, so for these first 4 arguements it is permissible to pass scalars defined in a FORTRAN parameter statement or explicit scalar values.)

`isiz1, ... isiz4` are what you expect the length of each dimension of the retrieved field to actually be. A warning will be provided if the dimensions of the retrieved field differ from what you expect. The warning for any dimension can be suppressed by assigning 0 to the corresponding `isiz`. (Each of these arguments might be reset by EzGet, so you must assiduously avoid passing scalars defined in a FORTRAN parameter statement or explicit scalar values. Instead of passing the explicit scalar `'32'`, you should assign the value '32' to some named integer scalar and then pass the scalar.)

`amask` is a real array that will be filled (or partially filled) by "weights" associated with `field`. It can have 1, 2, 3, or 4 dimensions, and these dimensions should be consistent with `ndim1, ndim2, ndim3`, and `ndim4`.

`field` is a real or integer array that will be filled (or partially filled) by the data identified by `var-index`. It can have 1, 2, 3, or 4 dimensions, and these dimensions should be consistent with `ndim1, ndim2, ndim3`, and `ndim4`.

and, on return from `getdata`,

`var-index` will have been left unmodified.

`ndim1, ... ndim4` will have been left unmodified.

`isiz1, ... isiz4` will be the lengths of each dimension of the retrieved field *as returned* by `getdata` and thus will define the subdomain of the arrays that actually contain data.

`amask` will contain the "weights" associated with `field`. All elements outside the domain of the field retrieved by `getdata` will be set to 0. If an element in the `field` array is "missing" or has been masked out, then the corresponding element in `amask` will also be 0.

`field` will contain the data identified by `var-index`. All elements outside the domain of the field retrieved by `getdata` (i.e., if `ndimi > isizi`) will be set to 0. "Missing

data" within the domain retrieved will be set to the value defined by a call to `defmisc` or by default will be set to 1.e20.

3.14 Subroutine getdimwt

After data have been retrieved (or after `shape` has been called), this subroutine returns a vector containing the weights associated with a specified dimension. A call to `getdimwt` is of the form:

```
call getdimwt(var-index, idim, wts)
```

where

`var-index` (integer in the range 1 to 10) specifies for which variable (defined by calling `defvar` or `defvarex`) weights should be retrieved.

`idim` (integer) is the dimension for which the weights are being requested (1 for the first dimension of the variable, 2 for the second, etc.)

`wts` (real) is a vector that will receive the weights associated with dimension `idim`.

3.15 Subroutine getedges

After data have been retrieved (or after `shape` has been called), this subroutine returns a vector containing the location of the grid-cell edges (i.e. boundaries) for a specified dimension. The coordinate information is consistent with what has been or will be retrieved by EzGet, which may, for example, be a subset of what appears in the original file. A call to `getedges` is of the form:

```
call getedges(var-index, idim, edges)
```

where

`var-index` (integer in the range 1 to 10) specifies for which variable (defined by calling `defvar` or `defvarex`) grid-cell information should be retrieved.

`idim` (integer) is the dimension for which grid-cell edges are being requested (1 for the first dimension of the variable, 2 for the second, etc.)

`edges` (real) is a vector that will receive the location of the grid-cell edges for dimension `idim`. This vector should be dimensioned at least 1 larger than the number of grid-cells retrieved.

3.16 Subroutine getfield

This subroutine simply retrieves data, but unlike `getdata` and `getnogap` it does no masking or mapping to a different grid. No mask is created and the data returned are forced to occupy contiguous memory. Before calling this subroutine it is recommended that you first call `defmisc` with the `'data size'` option selected, informing EzGet of the actual declared size of the array that will receive the data. This allows EzGet to check that your array is large enough to receive the data.

A call to `getfield` is of the form:

```
call getfield(var-index, field)
```

where,

`var-index` (integer in the range 1 to 10) specifies from which variable (defined by calling `defvar` or `defvarex`) data should be retrieved.

`field` returns the data identified by `var-index`. "Missing data" within the domain retrieved will be set to the value defined by a call to `defmisc` or by default will be set to 1.e20.

3.17 Subroutine getgeog

This subroutine creates a geography mask for a specified region. Normally you do not need to create this mask as a separate step because `defgeog` will already have made it possible to select the desired geographical regions. The input geography data set that will be accessed may be a simple land-ocean-sea ice mask (or even more simply, a land-sea mask), a land fraction (expressed as a percent), a sea ice fraction (expressed as a percent), or a more detailed geographical data set uniquely defining regions typically the size of a continent (as described in more detail in Section 3.5). Appendix B contains a list of regions identified on the standard geographical maps for AMIP and PMIP models available from PCMDI. A call to `getgeog` is of the form:

```
 call getgeog(var-index, ndim1, ndim2, ndim3, ndim4,
+        isiz1, isiz2, isiz3, isiz4, amask, select)
```

where

`var-index` (integer in the range 1 to 10) specifies which variable (defined by calling `defvar` or `defvarex`) contains the geography data.

`ndim1, ... ndim4` (integers) should be identical to the dimensions of `amask`. If this array has fewer than 4 dimensions, the unused dimensions should be set to either 0 or 1. If set to 0, then an error message will be displayed if the field actually has more dimensions than allowed for. For example, if `amask` is dimensioned `amask(64,32)`, then you should specify `ndim1=64`, `ndim2=32`, `ndim3=0`, and `ndim4=0`. (Each of these arguments is left unaltered by EzGet, so for these first 4 arguements it is permissible to pass scalars defined in a FORTRAN parameter statement or explicit scalar values.)

`isiz1, ... isiz4` are what you expect the length of each dimension of the retrieved field to actually be. A warning will be provided if the dimensions of the retrieved field differ from what you expect. The warning for any dimension can be suppressed by assigning 0 to the corresponding `isiz`. (Each of these arguments might be reset by EzGet, so you must assiduously avoid passing scalars defined in a FORTRAN parameter statement or explicit scalar values. Instead of passing the explicit scalar `'32'`, you should assign the value '32' to some named integer scalar and then pass the scalar.)

`amask` is a real array that on return indicates whether or not a grid cell lies within the selected region (a '0.0' indicates the cell lies outside the selected region and a '1.0' indicates it lies inside the region). In the case of land fraction or sea ice fraction input data, `amask` will be returned with a fraction in the range 0.0 to 1.0. `amask` can have 1, 2, 3, or 4 dimensions, and the declared dimension lengths should be the same as `ndim1, ndim2, ndim3`, and `ndim4`.

`select` (character string) is a list of regions, separated by commas, specifying which geographical regions of the globe should be selected. If the geography data simply indicates regions of land, ocean, and possibly sea-ice (as stored in `'sft'` files) then this argument might be, for example, `'ocean, sea ice'`, which would select all regions except land. If the geography data contains the more detailed geographical information (as stored in `sftbyrgn` files), then this argument might be, for example, `'North America, South America, Greenland'`, which selects the major land areas of the western hemisphere. Note that the commas separating the listed regions may optionally be followed by blanks. For example, either `'Africa, Australia, Antarctica'` or `'Africa,Australia,Antarctica'` is acceptable. See Section 3.5 for further explanation.

3.18 Subroutine getnogap

This subroutine retrieves data, possibly mapping it to a different grid and masking user-specified geographical regions, and creates an appropriate mask (or set of "weights") associated with the data. It differs from `getdata` in that the data returned

are forced to occupy contiguous memory.

A call to `getnogap` is of the form:

```
call getnogap(var-index, nsize, amask, field)
```

where,

`var-index` (integer in the range 1 to 10) specifies from which variable (defined by calling `defvar` or `defvarex`) data should be retrieved.

`nsize` is the declared size of arrays `amask` and `field`, so that EzGet can check that they are large enough to receive all the extracted data.

`amask` returns the "weights" associated with `field`. If an element in the `field` array is "missing" or has been masked out, then the corresponding element in `amask` will also be 0.

`field` returns the data identified by `var-index`. "Missing data" within the domain retrieved will be set to the value defined by a call to `defmisc` or by default will be set to 1.e20.

3.19 Subroutine initget

This subroutine must be called to initialize EzGet. It assigns default values to a few parameters and sets up some internal tables. Subroutine `initget` has no arguments, so it is called as follows:

```
call initget
```

3.20 Subroutine lendims

After a field has been retrieved, this subroutine can return the length of each dimension of the retrieved variable. (The lengths returned give the dimensions of the data retrieved, which may not be the same as the data in the source file.) Normally this subroutine would be called after the field has been extracted by either subroutine `getnogap` or `getfield`. A call to `lendims` is of the form:

```
call lendims(var-index, ldim1, ldim2, ldim3, ldim4, isize)
```

where

`var-index` (integer in the range 1 to 10) specifies for which variable (defined by calling `defvar` or `defvarex`) dimension information should be retrieved.

`ldim1, ldim2, ldim3, ldim4` (integers) returns the lengths of dimensions 1, 2, 3, and 4, respectively. If the second, third or fourth dimensions do not exist, then `ldim` for that dimension is set to 0.

`isize` (integer) returns the size of the array retrieved (= ldim1*ldim2*ldim3*ldim4, but with any 0's replaced by 1's).

3.21 Subroutine shape

Before EzGet retrieves data, this subroutine can obtain the length of each dimension of a variable. (The lengths returned indicate the size that will be retrieved, not necessarily the size stored.) Typically, this subroutine would be called in order to determine the size of the arrays that will be needed to accomodate the data that will be retrieved. A call to `shape` is of the form:

```
call shape(var-index, ldim1, ldim2, ldim3, ldim4, isize)
```

where

`var-index` (integer in the range 1 to 10) specifies for which variable (defined by calling `defvar` or `defvarex`) dimension information should be retrieved.

`ldim1, ldim2, ldim3, ldim4` (integers) return the lengths of dimensions 1, 2, 3, and 4, respectively. If the second, third or fourth dimensions do not exist, then `ldim` for that dimension is set to 0.

`isize` (integer) returns the size of the array needed to accomodate the retrieved data (= ldim1*ldim2*ldim3*ldim4, but with any 0's replaced by 1's).

3.22 Subroutine varinfo

This subroutine returns descriptive information retrievable from the file containing a defined variable. Information retrievable includes the data source, title, units, date, time, and variable-type. A call to `varinfo` is of the form:

```
call varinfo(var-index, param, info)
```

where

`var-index` (integer in the range 1 to 10) specifies from which variable (defined by calling `defvar` or `defvarex`) the information should be obtained.

`param` (character string) indicates what information should be returned. The options are:

- `'units'` — to obtain the units of the variable (40 characters).
- `'source'` — to obtain the source description for the variable (120 characters).
- `'title'` — to obtain the title of the variable (80 characters).
- `'date'` — to obtain the date the data were generated (8 characters).
- `'time'` — to obtain the time the data were generated (8 characters).
- `'type'` — type of variable (e.g., R*4, I*4, R*8, I*8, C*16) (8 characters).
- `'weight'` — type of weighting (e.g., 'cosine', 'gaussian', 'uniform', etc.) (vector of 4 elements, each 8 characters long).

`info` (character string) returns the requested information. The length of the character string depends on what is specified for `param`, as indicated above.

4 Avoiding Errors

4.1 Input/output devices

Because EzGet (through cdunif) opens files, it may internally assign and reference the following FORTRAN input/output device numbers: 91–94 and 96–99. You should therefore avoid using these same device numbers in your code (unless you have completed your calls to EzGet subroutines and have been careful to close all files opened by EzGet by calling subroutine `closeget`).

You should also note that the DRS library, which may be accessed by EzGet, has a limit of 6 DRS files that can be simultaneously open. Because EzGet might open up to 4 different DRS files, you should be careful to avoid opening more than 2 more DRS files (outside EzGet), or else the limit of 6 files will be exceeded.

4.2 Subroutine and common names

When you write programs that will be linked to EzGet, make sure that the names identifying your subroutines, functions and commons are different from names already used in EzGet. Specifically avoid the following names:

- *Commons:* cdimtbl, cdomain, ciotbl, cregddim, cscalars, cvartbl, usersget
- *Integer functions:* applywts, doregrid, genwts, getdimen, getfld, mkdimtbl, opendrs, univunit, xgeog, xregion
- *Logical functions:* caseindp
- *Subroutines:* bsslzr, closeget, clrtable, defdim, defdimi, defgeog, defmisc, defregrd, defvar, defvarex, domain, domlimit, errcheck, gauaw, getcoord, getdata, getedges, getdimwt, getfield, getgeog, getnogap, getvdata, initget, lendims, maparea, rgdarea, shape, varinfo
- *Combined List:* applywts, bsslzr, caseindp, cdimtbl, cdomain, ciotbl, closeget, clrtable, cregddim, cscalars, cvartbl, defdim, defdimi, defgeog, defmisc, defregrd, defvar, defvarex, domain, domlimit, doregrid, errcheck, gauaw, genwts, getcoord, getdata, getdimen, getdimwt, getedges, getfield, getfld, getgeog, getnogap, getvdata, initget, lendims, maparea, mkdimtbl, opendrs, rgdarea, shape, univunit, usersget, varinfo, xgeog, xregion

4.3 EzGet size limits

To limit the amount of memory required in running EzGet, there are limits on how many different variables and dimensions can be simultaneously accessed. The maximum number of variables that can be defined is 10 (which is why the index in a call to `defvar` is limited to the range 1 to 10). The maximum number of different dimensions that will be accomodated by EzGet is 100. This limit is usually sufficient unless you loop through some long dimension, each time extracting a single plane. For example if you have a 10-year long time-series of monthly data of surface air temperature and you cycle through the individual months, then when you get to month 99, you will encounter an error because you already will have saved 100 dimensions (the latitude and longitude dimensions plus 98 time dimensions, one for each of the first 98 months). If, on the other hand, you had extracted the full three-dimensional array at one time, then you would have generated only 3 dimensions (2 spatial dimensions and 1 time dimension). You may make room for more dimensions by calling subroutine `clrtable` at some convenient point.

There are also limits on how many coordinate values can be extracted and placed in a table for future reference by EzGet. See subroutine `defmisc` for further information.

5 Obtaining and Installing EzGet Software

The EzGet software is currently available for the following platforms/operating systems:[5]

- Sun/SunOS 4.1.3
- Sun/Solaris 2.4
- IBM RS6000/AIX 3.2
- HP/HP-UX 9.0
- SGI Irix 5.3

You may obtain the FORTRAN libraries comprising EzGet from the PCMDI web site:

home page: http://www-pcmdi.llnl.gov/
EzGet location: http://www-pcmdi.llnl.gov/ktaylor/ezget/ezget.html

In addition, if you will be reading netCDF files, you will need to acquire the netCDF library from the unidata web site:

ftp://ftp.unidata.ucar.edu/pub/netcdf/

There are geography data sets available for the AMIP and PMIP models that allow you to extract data from specific geographical regions (e.g., North America, South Atlantic, Australia, etc.). Information on how to obtain these geography data sets along with copies of the examples given in section 2 are available at the PCMDI web site.

Acknowledgments

I thank Peter Gleckler, Ben Santer, and Curt Covey for exercising early versions of this software and uncovering several bugs; I appreciate their patience and feedback. I also thank them and Jean-Yves Peterschmitt, Emannuelle Cohen-Solal, and Larry Gates for reading parts of this documentation and making comments and suggestions to improve it. Bob Drach (the author of cdunif) has been helpful in several ways, including providing assistance in porting this software to different platforms.

This work was performed under the auspices of the U.S. Department of Energy Environmental Sciences Division by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

[5]EzGet will also be ported to the Cray/Unicos and DEC Alpha/OSF on request.

Appendix A

TABLE 1. The AMIP and PMIP model acronyms given below are recognized by EzGet and may be used in calls to subroutines `defdim` or `defdimi` to specify "weights" for longitude and latitude. The weights may be needed to map data to a different grid or may be used to compute area-weighted statistics. You may specify the correct weights either by the model acronym shown in column one below, or more explicitly by specifying the options indicated in the third and fourth columns below.

EzGet Acronym[1]	AMIP/PMIP Group	Longitude Weights	Latitude Weights
`'bmr*'`	Bureau of Meteorology Research Centre (BMRC)	`'width'`	`'gaussian'`
`'ccc*'`	Canadian Centre for Climate Modelling and Analysis (CCCMA)	`'width'`	`'gaussian'`
`'ccm*'`	various groups and versions of the Community Climate Model	`'width'`	`'gaussian'`
`'ccs*'`	Center for Climate System Research (CCSR)	`'width'`	`'gaussian'`
`'cnr*'`	Centre National de Recherches Météorologiques (CNRM)	`'width'`	`'gaussian'`
`'col*'`	Center for Ocean-Land-Atmosphere Studies (COLA)	`'width'`	`'gaussian'`
`'csi*'`	Commonwealth Scientific and Industrial Research Organization (CSIRO)	`'width'`	`'gaussian'`
`'csu*'`	Colorado State University (CSU)	`'width'`	`'csu'`[2]
`'der*'`	Dynamical Extended Range Forecasting (at GFDL)	`'width'`	`'gaussian'`
`'dnm*'`	Department of Numerical Mathematics (of the Russian Academy of Sciences)	`'width'`	`'cosine'`
`'ech*'`	Max-Planck-Institut für Meteorologie (ECHAM model)	`'width'`	`'gaussian'`
`'ecm*'`	European Centre for Medium-Range Weather Forecasts (ECMWF)	`'width'`	`'gaussian'`

EzGet Acronym[1]	AMIP/PMIP Group	Longitude Weights	Latitude Weights
`'gen*'`	Various groups and versions of the GENESIS model	`'width'`	`'gaussian'`
`'gfd*'`	Geophysical Fluid Dynamics Laboratory (GFDL)	`'width'`	`'gaussian'`
`'gis*'`	Goddard Institute for Space Studies (GISS)	`'width'`	`'cosine'`
`'gla*'`	Goddard Laboratory for Atmospheres (GLA)	`'width'`	`'cosine'`
`'gsf*'`	Goddard Space Flight Center (GSFC)	`'width'`	`'cosine'`
`'iap*'`	Institute of Atmospheric Physics (of the Chinese Academy of Sciences)	`'width'`	`'cosine'`
`'jma*'`	Japan Meteorological Agency (JMA)	`'width'`	`'cosine'`[3]
`'lmc*'`	various version of the LMD model used by the Laboratoire de Modélisation du Climat et de l'Environnement (LMCE)	`'width'`	`'lmc'`[4]
`'lmd*'`	Laboratoire de Météorologie Dynamique (LMD)	`'width'`	`'lmd'`[4]
`'mgo*'`	Main Geophysical Observatory (MGO)	`'width'`	`'gaussian'`
`'mpi*'`	Max-Planck-Institut für Meteorologie (MPI)	`'width'`	`'gaussian'`
`'mri*'`	Meteorological Research Institute (MRI)	`'width'`	`'cosine'`
`'nca*'`	National Center for Atmospheric Research (NCAR)	`'width'`	`'gaussian'`
`'nce*'`	National Center for Environmental Prediction (NCEP)	`'width'`	`'gaussian'`
`'nmc*'`	National Meteorological Center (NMC)	`'width'`	`'gaussian'`
`'nrl*'`	Naval Research Laboratory (NRL)	`'width'`	`'gaussian'`
`'rpn*'`	Recherche en Prévision Numérique (RPN)	`'width'`	`'gaussian'`
`'sng*'`	State University of New York at Albany / National Center for Atmospheric Research (SUNYA/NCAR)	`'width'`	`'gaussian'`

EzGet Acronym[1]	AMIP/PMIP Group	Longitude Weights	Latitude Weights
`'sun*'`	State University of New York at Albany (SUNYA)	`'width'`	`'gaussian'`
`'ucl*'`	University of California at Los Angeles (UCLA)	`'width'`	`'ucl'`[2]
`'uga*'`	The UK Universities' Global Atmospheric Modelling Programme (UGAMP)	`'width'`	`'gaussian'`
`'uiu*'`	University of Illinois at Urbana-Champaign (UIUC)	`'width'`	`'cosine'`
`'ukm*'`	United Kingdom Meteorological Office (UKMO)	`'width'`	`'cosine'`
`'yon*'`	Yonsei University (YONU)	`'width'`	`'cosine'`

[1]The model acronyms are truncated by EzGet to three characters so in the table above the character `'*'` is 'wild' meaning that it represents any string of characters, including the null string.

[2]The UCLA and CSU models have regularly spaced latitudes, but there is no grid point at the poles, so the most poleward grid cells in each hemisphere have a latitudinal width of 1.5 times the other grid cells. For proper construction of area weights, specify `'csu'` or `'ucl'` (*not* `'cosine'`) for the latitude weights.

[3]The JMA model has a gaussian grid, but the data are reported on a 2.5x2.5 degree regular grid.

[4]The LMD and LMCE models have equal area grid cells which become elongated in latitude away from the equator. For proper construction of area weights, specify `'lmd'` or `'lmc'` for the latitude weights.

Appendix B

TABLE 2. Specifications for Geographical Regions

Integer I.D.	Region	EzGet Specification
0	ocean	`'oce*'`
1	land	`'lan*'`
2	sea ice	`'seai*', 'sea-i*', 'sea i*'`
217	North America	`'n* ame*'`
216	South America	`'s* ame*'`
215	Greenland	`'gre*'`
218	Africa	`'afr*'`
219	Europe & Asia	`'eur*asia*'`
221	Australia	`'aus*'`
220	Antarctica	`'ant*'`
222	Indo Pacific Islands	`'ind* i*'`
211	American Lakes	`'ame* l*'`
212	Baffin & Hudson_Bays	`'baf*'`
208	Asian & African Lakes	`'asi* l*'`
207	Mediterranean Sea	`'med*'`
201	North Pacific Ocean	`'n* pac*'`
–	South Pacific Ocean	`'s* pac*'`
202	South Pacific (N. of Melbourne)	`'s* pac* 1*'`
203	South Pacific (N. of Cape Horn and S. of Melbourne)	`'s* pac* 2*'`
204	South Pacific (S. of Cape Horn)	`'s* pac* 3*'`
209	North Atlantic Ocean	`'n* atl*'`
–	South Atlantic Ocean	`'s* atl*'`
213	South Atlantic (N. of Cape of Good Hope)	`'s* atl* 1*'`
214	South Atlantic (S. of Cape of Good Hope)	`'s* atl* 2*'`
–	Indian Ocean	`'ind*'`
205	Indian Ocean (N. of S. Australia)	`'ind* 1*'`
206	Indian Ocean (S. of S. Australia)	`'ind* 2*'`
210	Arctic Ocean	`'arc*'`

Notes concerning Table 2:

- The character `'*'` is 'wild' meaning that it represents any string of characters, including the null string (but blanks are not permitted except as part of the group, `' & '`). For example, EzGet treats the following strings as equivalent: `'North America'`, `'n ame'`,

`'N. Amer.'`, etc.

- EzGet is case insensitive (at least in interpreting the geography strings) so, for example, the following strings are equivalent: `'North America'`, `'north america'`, `'NORTH AMERICA'`, etc.

- It is permissible to follow the name of an ocean or sea by the strings `' ocean'` or `' sea'`. For example, `'Mediterranean Sea'` is equivalent to `'Mediterranean'` and `'North Pacific Ocean'` is equivalent to `'North Pacific`. Note, however, that the numbered ocean basin subdomains (e.g., `South Pacific 1'`) should not be followed by either `' ocean'` or `' sea'`.

- The following equivalences are recognized:

  `'S Pac' = 'S Pac 1, S Pac 2, S Pac 3'`

  `'S Atl' = 'S Atl 1, S Atl 2'`

  `'Indian' = 'Indian 1, Indian 2'`

- When accessing the detailed geography data, you cannot select `'sea ice'` because the sea-ice distribution is not available from those files. All land and ocean regions can, however, be selected because the following equivalences are recognized:

  ```
  'land' = 'n ame, s ame, gre, afr, eur-asia, aus, ant, ind i'

  'ocean, sea ice' =
          'ame l, baf, asi l, med, n pac, s pac, n atl, s atl, ind, arc'
  ```

- The entry '–' appears in column one of the table because the region comprises two or more subregions. For example, `'South Atlantic'` selects all grid cells in either the `'South Atlantic 1'` or the `'South Atlantic 2'` regions (i.e., all grid cells that have been assigned either the value 213 or the value 214 in the geography file).

Figure 1, which follows on the next page, shows the geographical regions recognized by EzGet. The Indo-Pacific Islands, American Lakes, and Asian & African Lakes can also be selected as indicated in the table, but are not labeled on the figure.

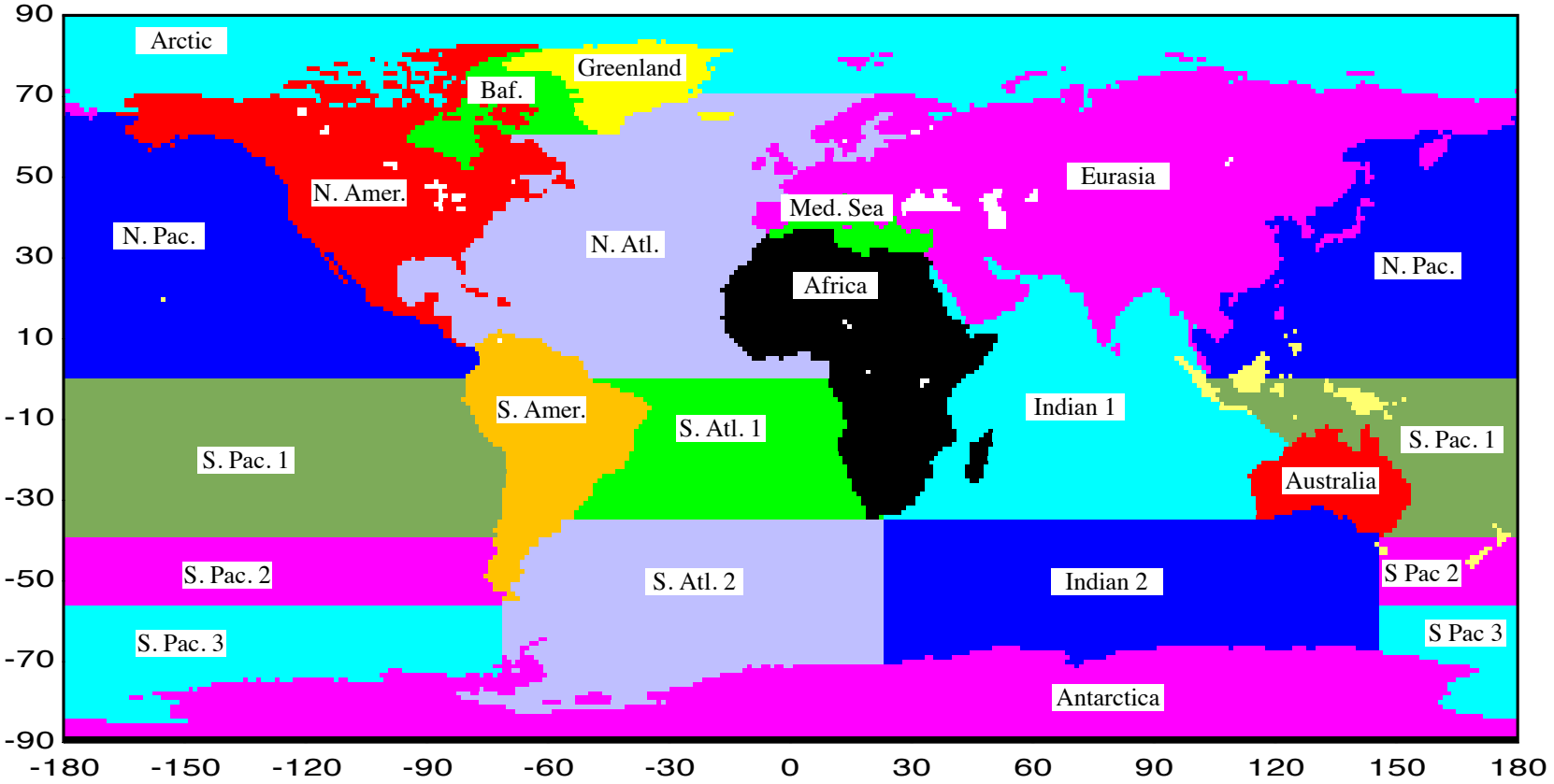


Figure 1: Geographical regions that can be selected through EZget. The Indo-Pacific Islands, American Lakes, and Asian & African Lakes can also be selected as indicated in Table 2, but are not labeled on this figure.