NSF Award #OCE-9900310
NOAA # NA67RJ0152 AMEND #30

TransCom 3 Experimental Protocol

By Kevin Gurney, Rachel Law, Peter Rayner, and Scott Denning

Colorado State University

DEPARTMENT OF ATMOSPHERIC SCIENCE

PAPER NO.707

TransCom 3 Experimental Protocol

Kevin Gurney, Rachel Law,
Peter Rayner, Scott Denning

July 2000

Table of Contents

A. Overview:

The TransCom 3 experimental protocol describes the steps that participants must take in order to run the TransCom 3 experiment: an atmospheric carbon budget inversion. It also contains useful fortran code and a description of all the input data and format of the experimental output. The protocol has grown in size from the initial expectations. Some of this is due to elaboration of some of the experimental details while some is due to additions and adjustments. Please read the protocol *carefully*. Adhering to the details of the protocol will ensure truly comparitive results.

For background on this experiment and online text of this protocol, go to the TransCom website at: http://transcom.colostate.edu. Important items to review there are the TransCom 3 proposal and the outcome of the TransCom 3 kick-off meeting held in San Francisco December 1998.

This protocol is organized as follows:

- a description of the forward simulations
- a description of the input data
- details on reading the input data and regridding
- details on scaling input totals
- specifications of the required output

The experiment contains three different levels. The first level focuses on annual mean carbon sources and sinks and will require a limited number of tracers using supplied input fields of regional carbon exchange. The second level expands on the first by including an inverse calculation of the strength of the seasonal cycle in regional fluxes, but will require a much larger number of CTM tracers. ***Participants wishing to perform the level II experiment do not need to perform level I***. The third level invites participants to perform their own inversions allowing for a comparison of different inversion methods. Throughout the protocol, the requirements and procedures for these three levels are designated separately where it is appropriate.

All questions and comments should be sent to Kevin Gurney (kevin@atmos.colostate.edu).

Considerable thanks must be given to all those who attended the December 1998 TransCom 3 workshop in San Francisco. Extra thanks to those who contributed comments and suggestions while this document was being constructed.

B. The forward simulations:

Level I: Annual mean inversion (37 individual tracers)

The annual mean inversion consists of a 4 year forward simulation (365 days per year) containing 4 pre-subtracted tracers, 11 sulfur hexafluoride (SF_6) tracers, and 22 CO_2 tracers (11 terrestrial, 11 oceanic). A map of the basis regions is included in Appendix B.

1) Fossil-fuel carbon pre-subtraction

Run the supplied 1990 and 1995 fossil-fuel flux maps forward for 4 years *as two separate tracers.* Since these maps represent the annual mean flux, no time interpolation is required - the flux values at each grid cell are simply repeated every timestep for 4 years.

2) Neutral biosphere carbon pre-subtraction

Run the supplied mid-month NEP flux maps forward for 4 years. This requires running the provided maps in repetition for each of the 4 simulated years. These mid-month values must be interpolated to, at least, daily fluxes.

3) Oceanic carbon exchange pre-subtraction

Run the supplied mid-month net oceanic carbon exchange maps forward for 4 years. This requires running the provided maps in repetition for each of the 4 simulated years. These mid-month values must be interpolated to, at least, daily fluxes.

4) Terrestrial carbon basis functions

Run the supplied terrestrial carbon basis function flux maps forward for 4 years *as 11 separate tracers.* Since these maps contain no seasonality, no time interpolation is required - the flux values at each grid cell are simply repeated every timestep for 4 years.

5) Oceanic carbon basis functions

Run the supplied oceanic carbon basis function flux maps forward for 4 years *as 11 separate tracers.* Unlike the terrestrial carbon basis function flux maps, the oceanic maps contain seasonality (due to changing ice cover in polar regions) and have been supplied as 12 monthly maps for each oceanic region.[1] The impact of ice cover is small and only impacts four of the oceanic regions, therefore no time interpolation is required - flux values at each grid cell can be held at their month-specific value for each month period.

6) SF_6 basis functions

Run the supplied SF_6 basis function flux maps forward for 4 years *as 11 separate tracers.* Since these maps represent the annual mean flux, no time interpolation is required - the flux values at each grid cell are simply repeated every timestep for 4 years.

---

[1] The ice cover map used in the construction of the oceanic carbon basis functions may not match the ice cover map implicit in offline model wind fields or used explicitly within online models. Correcting this would be extremely difficult, so it has been left as part of the "model-to-model" difference.

Level II: Seasonal inversion (279 individual tracers)

The seasonal inversion consists of a 3 year forward simulation (365 days per year) containing 4 pre-subtracted tracers, 11 SF_6 tracers, and 22 CO_2 tracers (11 terrestrial, 11 oceanic). A map of the basis regions is included in Appendix B. The primary difference between level I and level II is the way in which the CO_2 tracers are simulated. This experiment will attempt to invert for the spatial and temporal pattern of the residual CO_2 sources and sinks.

1) Fossil-fuel carbon pre-subtraction

Run the supplied 1990 and 1995 fossil fuel CO_2 emissions maps forward for one year. At the beginning of the second year, shut off the emissions and continue to run forward for two more years. Since these maps represent the annual mean flux, no time interpolation is required - the flux values at each grid cell are simply repeated every timestep for one year.

Truncation: To limit the computational burden, you may truncate the forward integration prior to completing three years (for example, a two year forward integration). The output fields returned to the TransCom coordinators (see section F) must contain the full three years and will, therefore, require some form of extrapolation of this tracer (extrapolation method is at the modelers discretion).

2) Neutral biosphere carbon pre-subtraction

Run the supplied mid-month NEP flux maps forward for one year. At the beginning of the second year, shut off the emissions and continue to run forward for two more years. These mid-month values must be interpolated to, at least, daily fluxes.

Truncation: To limit the computational burden, you may truncate the forward integration prior to completing three years (for example, a two year forward integration). The output fields returned to the TransCom coordinators (see section F) must contain the full three years and will, therefore, require some form of extrapolation of this tracer (extrapolation method is at the modelers discretion).

3) Oceanic carbon exchange pre-subtraction

Run the supplied mid-month net oceanic carbon exchange maps forward for one year. At the beginning of the second year, shut off the emissions and continue to run forward for two more years. These mid-month values must be interpolated to, at least, daily fluxes.

Truncation: To limit the computational burden, you may truncate the forward integration prior to completing three years (for example, a two year forward integration). The output fields returned to the TransCom coordinators (see section F) must contain the full three years and will, therefore, require some form of extrapolation of this tracer (extrapolation method is at the modelers discretion).

4) Terrestrial carbon basis functions

Run the supplied terrestrial carbon basis function flux maps forward for 3 years as 132 separate tracers (11 regions x 12 months). These 132 separate tracers are run as pulsed emissions - that is, each region emits a flux for each of 12 months in the first year of the simulation. Each region/month combination is, therefore, a *separate* tracer. The emissions from each basis function/month are sustained for one month, then turned off. Transport of each of these 132 tracers is continued for 36 months as the spatial structure decays.

As per the explanation in Level I, these maps contain no seasonality. Therefore, no time interpolation is required - the flux values at each grid cell are simply repeated every timestep within the monthlong pulses.

*Truncation: To limit the computational burden, you may truncate the forward integration prior to completing three years. A minimum forward integration is **13 months** - the month of the pulse and the following twelve. The monthly mean output fields returned to the TransCom coordinators (see section F) must contain the full three years and will, therefore, require some form of extrapolation of this tracer (exponential decay is recommended).*

5) Oceanic carbon basis functions

Run the supplied oceanic carbon basis function flux maps forward for 3 years as 132 separate tracers (11 regions x 12 months). These 132 separate tracers are run as pulsed emissions - that is, each region emits a flux for each of 12 months in the first year of the simulation. Each region/month combination is, therefore, a *separate* tracer. The emissions from each basis function/month are sustained for one month, then turned off. Transport of each of these 132 tracers is continued for 36 months as spatial structure decays.

Unlike the terrestrial carbon basis function flux maps, the oceanic maps contain seasonality (due to changing ice cover in polar regions) and have been supplied as 12 monthly maps (see footnote 1). The impact of ice cover is small and only impacts four of the oceanic regions, therefore no time interpolation is required - flux values at each grid cell can be held at their month-specific value for each month period.

*Truncation: To limit the computational burden, you may truncate the forward integration prior to completing three years. A minimum forward integration is **13 months** - the month of the pulse and the following twelve. The monthly mean output fields returned to the TransCom coordinators (see section F) must contain the full three years and will, therefore, require some form of extrapolation of this tracer (exponential decay is recommended).*

6) SF_6 basis functions

Identical to Level I

NOTE: Level II requires running a large number of tracers. Some participants will run all tracers simultaneously within a single forward simulation. Others may want to subdivide the tracers into separate forward simulations and reassemble the output after all are complete.

Level 3: Open methodology

Level 3 was constructed to allow for a comparison of different inversion approaches (Kalman, adjoint, etc.). While level 3 is "open" some basic requirements are necessary.

- The same model as was used in level 2, must be used for level 3.
- Level 3 modellers must use the TransCom 3 basis regions or a combination of the TransCom 3 regions for the inverted fluxes, if possible.
- Pre-subtraction of the fossil fuel, neutral biosphere, and ocean exchange must be included using the fields supplied in TransCom 3, if possible.
- The measured CO_2 locations must coincide with those used in the level 1 and level 2 analyses. The exact choice of stations has yet to be determined.

Participants wishing to perform the level 3 experiment should contact Kevin Gurney (kevin@atmos.colostate.edu) about further details.

Initial conditions and conversion factors

350 ppmv for all CO_2 (equivalent to a mass mixing ratio of 5.31×10^{-4} g/g) 2.0 pptv for all SF_6 (equivalent to a mass mixing ratio of 1.01×10^{-11} g/g).

Please use 29 g/mole for the molecular weight of air, 146 grams/mole for the molecular weight of SF_6 and 12 for the weight of carbon.

C. The input dataset description:

The TransCom 3 input dataset ('input.dat') comprises all the data necessary to initiate the forward runs in the TransCom 3 experiment. It contains seven records, each of which is a distinct tracer category. The file is a binary format and all variables are double precision (sample code with which to read the input data is supplied in the next section). Because we need to ensure that all participants have precisely the same input fluxes, double precision is used for the input file so as to maintain precision in checking the global totals (see section E). The input data can be downloaded in compressed form as 'input.dat.Z' from the TransCom webpage (http://transcom.colostate.edu) or via the anonymous ftp site (dendrus.atmos.colostate.edu, then go to the "transcom/" subdirectory).

All the input arrays represent 0.5 x 0.5 degree surface maps in which the first grid cell is centered at 89.75 S latitude, 179.75 W longitude, proceeding in the longitudinal direction. The grid arrangement is as follows:

```
i  = 1 is centered at 179.75w, i increases eastward
j  = 1 is centered at  89.75s, j increases northward

  88.5s - | - - - | - - - | - - - | - -
          | (1,3) | (2,3) | (3,3) |
  89s   - | - - - | - - - | - - - | - -
          | (1,2) | (2,2) | (3,2) |
  89.5s - | - - - | - - - | - - - | - -
          | (1,1) | (2,1) | (3,1) |
  90s -   | - - - | - - - | - - - |
        180w   179.5w  179w    178.5w
```

The data within the file 'input.dat' is as follows:

1) Fossil-fuel pre-subtraction emission maps

The first two records are the fossil fuel pre-subtraction emission maps. They represent the annual mean emissions from fossil-fuel burning, hydraulic cement production and gas flaring in 1990 and 1995, respectively.

Description/Reference: The 1990 emissions map is derived from the data prepared by Andres, Marland, Fung, and Matthews and can be found (with detailed supporting information) at the CDIAC website: http://cdiac.esd.ornl.gov/ndps/ndp058.html.

The 1995 emissions map is derived from the data prepared by Antoinette Brenkert and can be found (with detailed supporting information) at the CDIAC website: http://cdiac.esd.ornl.gov/ndps/ndp058a.html

Specifications: The original 1 x 1 degree maps were subsampled to 0.5 x 0.5 degree maps in which the first grid cell is centered at 89.75 S latitude, 179.75 W longitude, proceeding in the longitudinal direction.

These first two records each contain arrays of dimension: 720x360

Units: kg C/m^2/second.

2) Neutral biosphere pre-subtraction maps

The next record contains the net ecosystem production (NEP) pre-subtraction carbon exchange flux maps, one for each month.

Description/Reference: The NEP maps come from a steady-state CASA model run. These maps were used in Randerson *et al.*, "The contribution of terrestrial sources and sinks to trends in the seasonal cycle of atmospheric carbon dioxide," *Global Biogeochemical Cycles*, **11**, 535-560, 1997.

Contact Jim Randerson, jimr@sequoia.atmos.berkeley.edu, for more information.

Specifications: The original 1 x 1 degree maps were subsampled to 0.5 x 0.5 degree maps in which the first grid cell is centered at 89.75 S latitude, 179.75 W longitude, proceeding in the longitudinal direction.

This record contains an array of dimension: 720x360x12.

Units: kg C/m^2/second.

3) Ocean exchange pre-subtraction maps

The next record contains the net ocean pre-subtraction carbon exchange flux maps, one for each month.

Reference: The net oceanic pre-subtraction carbon exchange maps were produced by Taro Takahashi, updated from T97. Detailed information about this data can be found in Appendix D.

Specifications: The original 4 x 5 degree maps were subsampled to 0.5 x 0.5 degree maps in which the first grid cell is centered at 89.75 S latitude, 179.75 W longitude, proceeding in the longitudinal direction.

This record contains an array of dimension: 720x360x12.

Units: kg C/m^2/second.

4) Terrestrial carbon basis functions

The next record contains the terrestrial carbon basis function flux maps, one for each terrestrial region.

Reference: The terrestrial carbon basis function flux maps represent the terrestrial portion of the normalized surface fluxes. These are the fluxes that are "adjusted" in the inverse portion of the TransCom 3 experiment in order to minimize the difference between the observed and simulated tracer concentrations. Each map represents a particular terrestrial region in which the annual summed flux over the region is equal to 1 Gt C/year. The flux from grid cells outside of a particular region is zero. The spatial distribution of the flux reflects annual mean NPP distribution as provided by a steady-state run of the model CASA (contact Jim Randerson, jimr@sequoia.atmos.berkeley.edu, for more information on the NPP map).

Specifications: The original NPP 1 x 1 degree maps were subsampled to 0.5 x 0.5 degree maps in order to create regional unit fluxes. The first grid cell is centered at 89.75 S latitude, 179.75 W longitude, proceeding in the longitudinal direction.

This record contains an array of dimensions: 720x360x11. *Note that the third dimension is the region index.*

Units: kg C/m^2/second.

5) Ocean carbon basis functions

The next record contains the ocean basis function flux maps, one for each land region and month.

Reference: The monthly ocean basis function flux maps represent the oceanic portion of the normalized surface fluxes. These are the fluxes that are "adjusted" in the inverse portion of the TransCom 3 experiment in order to minimize the difference between the observed and simulated tracer concentrations. Each map represents a particular ocean region and particular month combination in which the annual summed flux over a region is equal to 1 Gt C.

There is no spatial distribution to the flux with the exception of seasonal ice cover in those areas for which ice cover occurs (changing ice cover is the motivation for monthly maps). The sea ice cover data comes from the boundary conditions used by World Climate Research Programme, Working Group on Numerical Experimentation, Atmospheric Model Intercomparison Project

(AMIP). Reference: Taylor, K.E., D. Williamson and F. Zwiers, "The sea surface temperature and sea ice concentration boundary conditions for AMIP II simulations" PCMDI Report, in preparation. URL: http://www-pcmdi.llnl.gov/amip/AMIP2EXPDSN/BCS/amip2bcs.html

Specifications: The original ice cover 1 x 1 degree maps were subsampled to 0.5 x 0.5 degree maps in order to create regional/monthly unit fluxes. The first grid cell is centered at 89.75 S latitude, 179.75 W longitude, proceeding in the longitudinal direction.

This record contains an array of dimensions: 720x360x11x12. *Note that the third dimension is the region index, the fourth dimension is the month index.*

Units: kg C/m^2/second.

6) SF_6 basis functions

The next record contains the SF_6 basis function flux maps, one for each land region.

Reference: The SF_6 basis function flux maps will be used to invert for a flux whose global distribution is relatively well-known. The same 11 land basis function regions used for the unit carbon fluxes are utilized in the SF_6 flux maps. Each map represents a particular land region in which the annual summed flux over the region is equal to 1 Gg SF_6/year.

Specifications: The spatial distribution of the SF_6 flux is similar to that used for the TransCom 2 experiment. In this experiment, the spatial distribution has been scaled such that the flux in each basis function region sums to 1 Gg SF_6/year.

The spatial distribution of the SF_6 flux was constructed as described in Denning *et al.* 1999 (TransCom 2 paper - downloadable from TransCom website).

The first grid cell is centered at 89.75 S latitude, 179.75 W longitude, proceeding in the longitudinal direction.

This record contains an array of dimensions: 720x360x11. *Note that the third dimension is the region index.*

Units: kg SF_6/m^2/second.

D. Reading input dataset and regridding:

Sample code[2]

```
************************************************************************
* This program reads the TransCom 3 input variables out of the binary
* file, "input.dat".
************************************************************************

      Real*8, Dimension(720,360)              :: ff90,ff95
      Real*8, Dimension(720,360,11)           :: sf6,landunit
      Real*8, Dimension(720,360,12)           :: nep,ocean
      Real*8, Dimension(720,360,11,12)        :: oceanunit

      Open(unit=10,file='input.dat',form='unformatted')

      Read(10) ff90
      Read(10) ff95
      Read(10) nep
      Read(10) ocean
      Read(10) landunit
      Read(10) oceanunit
      Read(10) sf6

      Close(10)

Spatially aggregate to your model grid for forward runs

      Stop
      End
************************************************************************
```

NOTE: due to the size of the input arrays, you may exceed your system's stack limit. If increasing the stack limit does not work, array declarations can be put into a common block file or structured as allocatable arrays (a fortran 90 option) within the above program. If you have any difficulties, contact Kevin Gurney: keving@atmos.colostate.edu

Spatial aggregation

1) Land/sea boundary

Spatial aggregation of the input fields onto your model grid raises some particularly difficult problems at the land/sea boundary. Given a terrestrial input carbon flux field such as fossil fuel emissions, some of the smaller 0.5x0.5 degree gridcells may lie in what a particular model land/sea mask may designate as an ocean gridcell. The approach we took in TransCom2 to deal with this was to simply take the amount of emissions found outside of the designated land area, and allocate it evenly across the entire land area such that the global totals matched those computed with the 0.5x0.5 degree map. However, this can cause unfortunate spatial redistribution of input fluxes that have maxima near coastlines where large vertical transport gradients occur. We have decided upon a method to solve this problem that is easy and best maintains the spatial pattern of the input fluxes (thanks to helpful input from Roger D., Rachel M., and Peter R.).

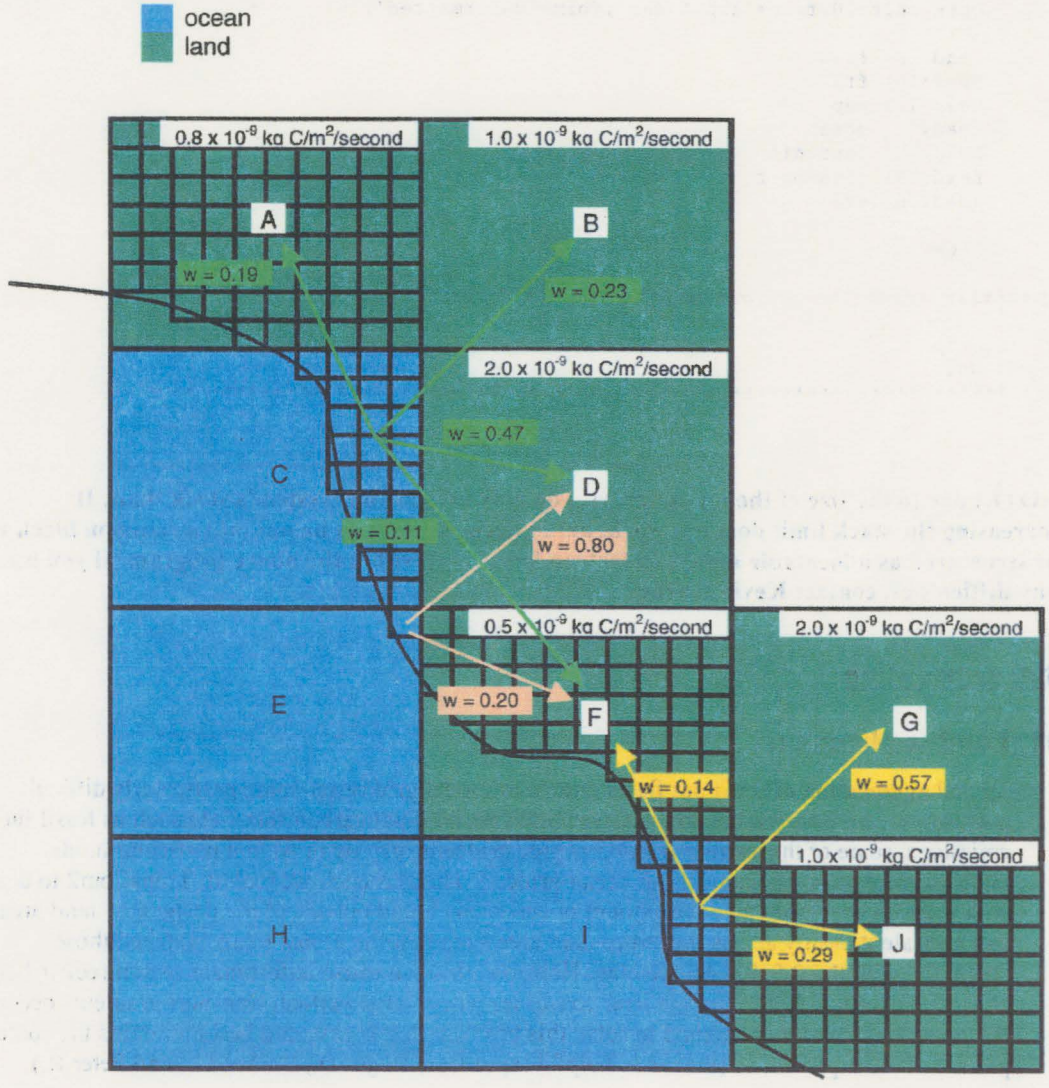
[2] Please read and carry the input data as double precision in order to match the global sums specified in section E. The precision with which each model carries out the forward integrations is an individual decision. However, output reporting will be required in single precision.

There are two steps to the procedure that can be performed for each input field reaggregation:

a) place all of the 0.5x0.5 degree cells from each input field within the confines of your (larger) model gridcells. For most modellers, the 0.5x0.5 degree size should fit evenly within your model gridcell (that is, most grids lie on whole or half degree lines). For those grids for which this is not the case, the 0.5x0.5 degree gridcells must be apportioned by area to the model gridcells that they straddle.

This allocation of the 0.5x0.5 degree gridcells to the model grid should be done while applying the land/sea mask that is routinely used with your model grid. So, all the gridcells that are considered "land" (in the case of the terrestrial pre-subtracted fields, for example) by your land/sea mask are filled by the 0.5x0.5 degree gridcells supplied in the input data. Refer to figure 1 for a schematic of a coastline example.

Figure 1



b) The next step is to go back over the two grids (your model grid and the 0.5x0.5 degree grid) and locate those model gridcells which contain emitting 0.5x0.5 degree gridcells but were allocated to ocean (while allocating a terrestrial pre-subtraction field, for example) or

land (while allocating the ocean pre-subtraction field, for example). In these cases the 0.5x0.5 degree gridcells in question should have their integrated flux added to the neighboring model gridcells *a la* weights which reflect the relative flux from those neighbors. In figure 1, 0.5x0.5 degree gridcells in model gridcell "C" are area integrated and added to the flux in the neighboring land model gridcells with the weights denoted by "w".[3]

The effect of this is to keep the reallocation of emissions *local*. This is most obvious for the case of fossil-fuel emissions which have strong maxima near the coastlines - instead of potentially removing this maxima and redistributing it across the entire globe or region, the maxima is shifted inland slightly. It is an imperfect solution in the sense that flux minima located on a coastline are essentially lost with the reshuffling procedure. However, for the purposes here, the loss of a small local flux is of comparitively little consequence.

This procedure should be performed for all of the input files except the ocean carbon basis functions. The ocean carbon basis function fields have essentially no spatial distribution (four of the basis functions have a bit of spatial structure due to seasonal sea ice cover) and therefore are best handled with the traditional approach. This is accomplished by eliminating those model gridcells that contain oceanic flux but which are considered land gridcells according to your land/sea mask. The region total is then scaled-up to match the annual, regional sum of 1 Gt C/year (see Section E).[4]

2) Region/region boundary

Another question concerning the spatial aggregation arises at boundaries between basis function regions. Unlike the land/ocean problem discussed above, there is no strict spatial mask to worry about (i.e. the land/sea mask of each model) and there are no obvious atmospheric transport gradients coinciding with any of the region/region boundaries. However, problems can arise since there is some significant spatial structure to the fluxes in the various regions (mainly the terrestrial basis functions and SF_6 basis functions).

I recommend a "soft" boundary approach. This will result in region/region boundaries overlapping by, on average, half of a model gridcell. The integrated flux in each region, however, will be maintained. Since the basis function arrays have been constructed such that each array contains values only in each region and the remainder of the array contains zeros, this makes the process very simple.

For each region (which are separate arrays in the input.dat file), place all of the 0.5x0.5 degree cells within the confines on your model gridcells. Model gridcells that contain even one 0.5x0.5 degree emitting gridcell should be considered part of the new regridded region defined by your model grid. This will maintain the total regional flux but spread the flux in the edge cells, on average, outward from the region by half of a model gridcell. Adjacent regions, will overlap in space slightly.

The other alternative might be referred to as a "hard" boundary in which there is no gridcell overlap: a model gridcell is in one region or another, not both. This moves the flux as with the "soft" approach but requires regional scaling and could cause significant alteration of the spatial structure of the flux were a region edge to contain high levels of flux.

Figure 2 shows these two approaches:

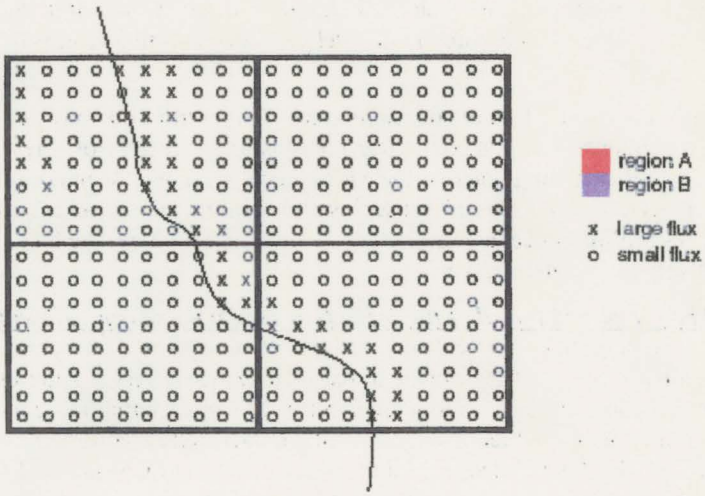
---

[3] Because many coastlines run roughly north-south, the order with which this shuffling scheme is performed does change the shuffling weights. For example, going from the south pole and working northward will have a slightly different shuffling effect than going from the North Pole and working southward. Though, the effect is small, we recommend determining the weights prior to reallocation - this eliminates the directional bias.

[4] Aside from the spatial flux pattern caused by sea ice cover (four regions only), further spatial patterning will occur due to the fact that some edge model gidcells will not be fully populated by 0.5x0.5 degree gridcells and, hence, will have less flux than interior gridcells.

Figure 2

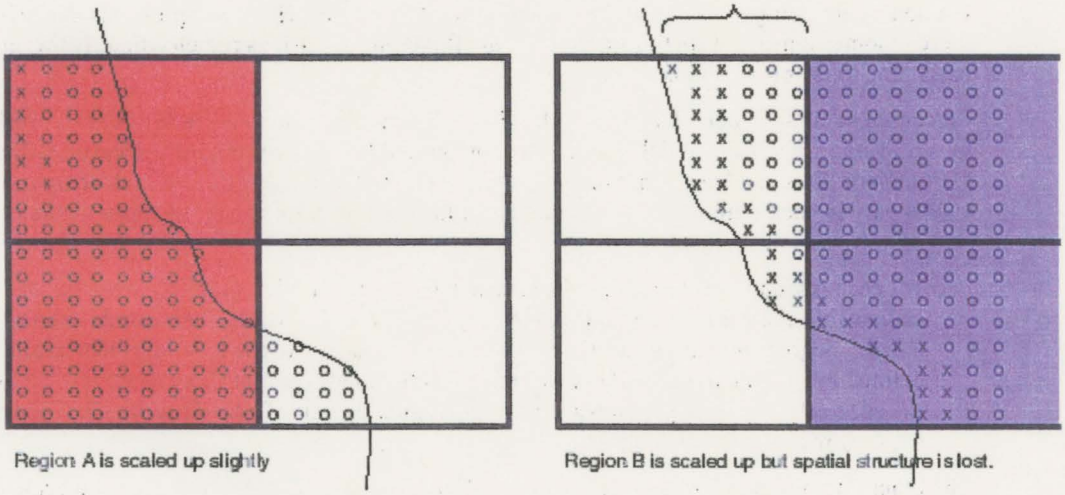
Imagine this as the nep flux in the vicinity of a regional boundary



The arrays in input.dat only define fluxes within a given region

HARD BOUNDARY

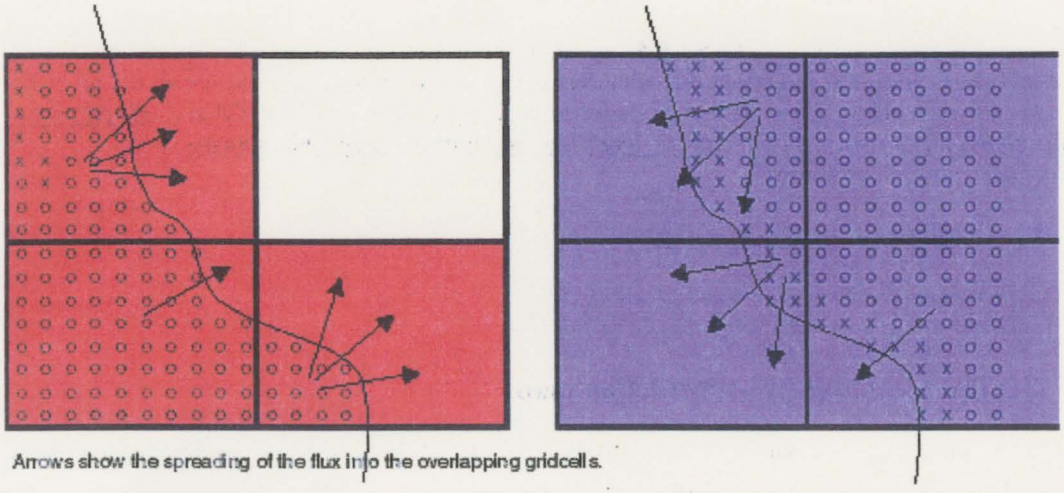
This flux that is not captured would be spread across region B according to the spatial distribution of the flux in the region.



Region A is scaled up slightly

Region B is scaled up but spatial structure is lost.

SOFT BOUNDARY



Arrows show the spreading of the flux into the overlapping gridcells.

E. Maintaining global totals:

All of the input data was generated at 0.5 x 0.5 degree to facilitate interpolation to the various model grids involved in TransCom 3. It is important that whatever spatial aggregation is performed does not alter the global sums provided below. After the input data is spatially aggregated to your model grid, ***please make sure your global totals match those below through regional/global scaling adjustments***. The global/regional sums are as follows[5]:

1) Fossil-fuel pre-subtraction maps

```
The total 1990 emissions are 5.811611 Gt
The total 1995 emissions are 6.172869 Gt
```

2) Neutral biosphere pre-subtraction maps

The following are global mid-month NEP values. After spatial aggregation to your model grid, ensure that your globally integrated mid-month fluxes match the following:[6]

```
Global NEP mid-month value for month 1  is:  242859.5 kg C/second
Global NEP mid-month value for month 2  is:  314486.3 kg C/second
Global NEP mid-month value for month 3  is:  211004.9 kg C/second
Global NEP mid-month value for month 4  is:  251455.6 kg C/second
Global NEP mid-month value for month 5  is: -148549.7 kg C/second
Global NEP mid-month value for month 6  is: -607653.2 kg C/second
Global NEP mid-month value for month 7  is: -755280.0 kg C/second
Global NEP mid-month value for month 8  is: -384896.5 kg C/second
Global NEP mid-month value for month 9  is:  180157.9 kg C/second
Global NEP mid-month value for month 10 is:  261082.3 kg C/second
Global NEP mid-month value for month 11 is:  233356.8 kg C/second
Global NEP mid-month value for month 12 is:  201913.8 kg C/second
```

Interpolate these mid-month fluxes to, at least, daily fluxes *using standard month lengths (31, 28, 31 days, etc.)*. After interpolation, the globally summed monthly totals should match the following:

```
Global NEP flux for month  1 is:  0.6644884 Gt/month
Global NEP flux for month  2 is:  0.7098920 Gt/month
Global NEP flux for month  3 is:  0.6139190 Gt/month
Global NEP flux for month  4 is:  0.5037655 Gt/month
Global NEP flux for month  5 is: -0.4368679 Gt/month
Global NEP flux for month  6 is: -1.4607660 Gt/month
Global NEP flux for month  7 is: -1.8092093 Gt/month
Global NEP flux for month  8 is: -0.9421531 Gt/month
Global NEP flux for month  9 is:  0.3264070 Gt/month
Global NEP flux for month 10 is:  0.6634400 Gt/month
Global NEP flux for month 11 is:  0.6024190 Gt/month
Global NEP flux for month 12 is:  0.5654337 Gt/month

Annual, global NEP flux is: 7.6815368E-4 Gt/year
```

[5] The global sums represent: (flux value*m^2/gridcell*seconds/year{or month}) summed over all grid cells.

[6] Assuming the fluxes represent mid-month values when initially derived as monthly means will lead to a different time integral. For the purposes of TransCom, however, the important requirements are that 1) everybody uses the same daily flux and 2) the global, annual flux in the mid-month daily interpolated version is not drastically different from the original monthly mean version. These should both be satisfied by adhering to the procedure and global totals listed.

3) Ocean exchange pre-subtraction maps

The following are global mid-month oceanic exchange values. After spatial aggregation to your model grid, ensure that your globally integrated mid-month fluxes match the following (see footnote 4):

```
Global ocean mid-month value for month  1 is: -78500.75 kg C/second
Global ocean mid-month value for month  2 is: -71361.29 kg C/second
Global ocean mid-month value for month  3 is: -75480.51 kg C/second
Global ocean mid-month value for month  4 is: -76880.54 kg C/second
Global ocean mid-month value for month  5 is: -77084.09 kg C/second
Global ocean mid-month value for month  6 is: -69717.96 kg C/second
Global ocean mid-month value for month  7 is: -47504.78 kg C/second
Global ocean mid-month value for month  8 is: -46194.30 kg C/second
Global ocean mid-month value for month  9 is: -53969.30 kg C/second
Global ocean mid-month value for month 10 is: -71638.83 kg C/second
Global ocean mid-month value for month 11 is: -79507.18 kg C/second
Global ocean mid-month value for month 12 is: -86656.08 kg C/second
```

Interpolate these mid-month fluxes to, at least, daily fluxes *using standard month lengths (31, 28, 31 days, etc.)*. After interpolation, the globally summed monthly totals should match the following:

```
Global ocean flux for month  1 is: -0.2101370 Gt/month
Global ocean flux for month  2 is: -0.1758068 Gt/month
Global ocean flux for month  3 is: -0.2013174 Gt/month
Global ocean flux for month  4 is: -0.1989271 Gt/month
Global ocean flux for month  5 is: -0.2037262 Gt/month
Global ocean flux for month  6 is: -0.1753489 Gt/month
Global ocean flux for month  7 is: -0.1338484 Gt/month
Global ocean flux for month  8 is: -0.1269569 Gt/month
Global ocean flux for month  9 is: -0.1435823 Gt/month
Global ocean flux for month 10 is: -0.1890994 Gt/month
Global ocean flux for month 11 is: -0.2061724 Gt/month
Global ocean flux for month 12 is: -0.2269120 Gt/month

Annual, global ocean flux is: -2.191835 Gt/year
```

4) Terrestrial carbon basis functions

```
The global total flux for each region is 1.0 Gt/year
```

5) Oceanic carbon basis functions

```
The global total flux for each region is 1.0 Gt/year
```

6) SF_6 basis functions

```
The global total flux for each region is 1.0 Gg/year
```

F. Output files

Output from the TransCom 3 experiment will be formatted as netCDF files. This will make central analysis simpler and allow all participants a common, accessible dataset for alternative analyses. Fortran code and specific instructions on how to write model output to netCDF files is provided in Appendix A.

Naming:

Each participant group will be submitting a number of data output files. Please give these files the name of your model or group (or group leader name) in place of "output" (which the netCDF writing routine I give you will produce at the beginning of the file name) whether you are turning in results for level I or level II. For example, were Martin Heimann to submit the files containing level II results using TM3, the file would be named, "TM3.heimann.XXX", where "XXX" is the remainder of the file name that the netCDF writing routine creates upon execution. An indication of the group or group leader is especially important for those models that are used by more than one participant.

Spatial aspects of output:

Three spatial forms will be used in the TransCom 3 experimental output: 3D (x, y, p), 2D (x, y) and single point reporting (wind speed and tracer concentration at a single location).

1) 3D fields:

3D fields will be reported according to the longitudinal and latitudinal dimensions of each participants model grid. In the vertical, this output must be reported on the following pressure (not sigma or other model coordinate levels) levels (in millibars).

100 millibars
200 millibars
300 millibars
400 millibars
500 millibars
700 millibars
850 millibars
925 millibars
1000 millibars

Note that *interpolation to pressure coordinates can be done after time averaging the wind and tracer fields.*[7] If your model does not extend to 100 mb, please report at all of the levels listed above that are within your model domain.

Any monthly mean values below the ground should be reported as *missing* (see the section on "terrain masking and missing values" below)

This means that the 3D output will come as arrays of dimension **im** x **jm** x **pm**, where **im** is the number of longitudinal grid cells in a participants model, **jm** is the number of latitudinal grid cell in a participants model, and **pm** is equal to 9 (or the maximum number of layers allowed by your model top), reflecting the pressure levels as designated above.

Please ensure that the first grid cell

- is at the dateline rather than at Greenwich
- is at the south pole rather than the north pole
- is at 1000 mb rather than 100 mb

[7] This has been done for simplicity and in recognition of the limited use of the 3D fields.

Strict adherence to these conventions will simplify the analysis process.

2) Maps

2D fields of the bottom two layers will be reported according to the longitudinal and latitudinal dimensions of each participants model grid. This means that the 2D field output will come as arrays of dimension **im** x **jm**, where **im** is the number of longitudinal grid cells and **jm** the number of latitudinal grid cell in a participant's model.

As with the 3D fields, please ensure that the first grid cell of the surface maps:

- is at the dateline rather than at Greenwich
- is at the south pole rather than the north pole

3) Single point reporting:

High frequency wind and tracer concentration will be reported at a collection of stations. The names and coordinates of these stations are listed in Appendix C. Elevation above sea level is provided at each of these stations. Use whatever interpolation scheme you deem appropriate to reflect the elevation.

Temporal aspects of output:

Two reporting time intervals will be used in TransCom 3: monthly means associated with the 2D and 3D fields and a time interval determined by each participant's model timestep, associated with the single point reporting. ***Please report in UTC rather than local time***.

1) 3D and 2D fields:

All 2D and 3D fields must be reported as monthly means. Please use real, non-leap year month lengths (31, 28, 31 days, etc.).

2) Single point reporting:

All single point fields should be reported at a 4 hour timestep or your model timestep if it is longer than 4 hours. ***Please report these as instantaneous values rather than 4 hour averages.***

Units:

Please report

- CO_2 concentration values as volumetric mixing ratios (parts per million by volume).
- SF_6 concentrations as volumetric mixing ratios (parts per trillion by volume).
- u and v winds in meters per second.
- ω wind in pascals/second.

Terrain masking and missing values:

Explicit surface terrain masking will not be required. However, all monthly mean values that reside on pressure surfaces below the ground (on pressure surfaces exceeding surface pressure) must be reported as missing for that month. ***For missing values, please use 1.0×10^{36} !!***

NOTE: monthly mean β values are part of the required output - if you have not included terrain masking, just send an array filled with zeros to the output subroutines outlined in Appendix A.

For those who want to include explicit surface terrain masking, the following procedure should be followed:

At many locations and at many times of the year, one or more of the reporting pressure levels are below the surface of the Earth. In order to properly report 3D fields at these locations, terrain masking is employed. As shown by Boer (1982), this can be accomplished by keeping careful track of the points above ground using a terrain mask, β, and by carefully defining averages only over points above ground. Define β on pressure surfaces as

$\beta = 1$ for $p < p_s$

$\beta = 0$ for $p \geq p_s$

The representative monthly mean average of a quantity X is now defined by

$$\overline{X} = \frac{\overline{\beta X}}{\overline{\beta}}$$

If every timestep of a given month is below the ground, designate this monthly mean quantity as "missing". ***For missing values, please use 1.0 x 10^{36} !!***

Required output - Level I

1) Pre-subtracted tracers:

Monthly mean 3D volumetric mixing ratio of the four pre-subtracted tracers from the last simulation year (1990 and 1995 fossil fuel, neutral biosphere, and net oceanic exchange). This represents 48 (4 CO_2 tracers x 12 months) 3D fields. ~6 MB

Monthly mean volumetric mixing ratio maps (two lowest model layers) of the four pre-subtracted tracers from the last simulation year (1990 and 1995 fossil fuel, neutral biosphere, and net oceanic exchange). This represents 96 (4 CO_2 tracers x 12 months x 2 layers) map fields. ~2 MB

2) Terrestrial and ocean exchange basis functions:

Monthly mean 3D volumetric mixing ratio of CO_2 for each basis function from the last simulation year for both the oceans and terrestrial basis functions. This represents 264 (22 regions x 12 months) 3D fields. ~35 MB

Monthly mean volumetric mixing ratio maps (two lowest model layers) of CO_2 for each basis function from the last simulation year for both the oceans and terrestrial basis functions. This represents 528 (22 regions x 12 months x 2 layers) surface map fields. ~7 MB

3) SF_6 tracers:

Monthly mean 3D volumetric mixing ratio of SF_6 from the last simulation year for each of 11 terrestrial basis functions. This represents 132 (11 regions x 12 months) 3D fields. ~20 MB

Monthly mean volumetric mixing ratio maps (two lowest model layers) of SF_6 from the last simulation year for each of 11 terrestrial basis functions. This represents 264 (11 regions x 12 months x 2 layers) surface map fields. ~5 MB

4) Winds:

Monthly mean 3D fields of u, v, and ω for the last simulation year. This represents 36 (3 winds x 12 months) 3D fields. ~5 MB

Monthly mean surface map fields of u and v from the last simulation year. This represents 24 (2 winds x 12 months) surface map fields. ~0.4 MB

5) Single Point Reporting:

High frequency station location reporting for CO_2 surface volumetric mixing ratio, u, and v from the last simulation year. This data will be placed into two arrays, the first containing u and v, and the second containing the CO_2 mixing ratio for all the separate simulations. Please use a 4 hour timestep for reporting this data.[8] The first array comes to (228 stations x 2190 timesteps x 2 winds) ~ 4.1 MB. The second array comes to to (228 stations x 2190 timesteps x 26 CO_2 tracers[9]) ~51 MB or a total of ~55 MB.

Please submit a text file containing the latitude, longitude, altitude, model level (or levels interpolated between) and the surface type for each site (see Appendix C).

6) $\overline{\beta}$: (for those computing an explicit surface terrain mask)

Monthly mean 3D $\overline{\beta}$ values from the last simulation year. This represents 12 (1 β x 12 months) 3D fields. ~1 MB

7) Land/Sea mask

This represents 1 surface map field. ~0.02 MB

Total ~130 MB

Required output - Level II

1) Pre-subtracted tracers:

Monthly mean 3D volumetric mixing ratio of the four pre-subtracted tracers for all three years (1990 and 1995 fossil fuel, neutral biosphere, and net oceanic exchange). This represents 192 (4 CO_2 tracers x 36 months) 3D fields. ~16 MB

Monthly mean volumetric mixing ratio maps (two lowest model layers) of the four pre-subtracted tracers for all three years (1990 and 1995 fossil fuel, neutral biosphere, and net oceanic exchange). This represents 384 (4 CO_2 tracers x 36 months x 2 layers) map fields. ~4 MB

2) Terrestrial and Ocean exchange basis functions:

Monthly mean 3D volumetric mixing ratio of CO_2 for both the oceans and terrestrial regions for all three years. This represents 12,672 (22 regions x 12 month-pulses x 36 months) 3D fields. ~1.1 GB

Monthly mean volumetric mixing ratio maps (two lowest model layers) of CO_2 for both the oceans and terrestrial regions for all three years. This represents 25,344 (22 regions x 12 month-pulses x 36 months x 2 layers) surface map fields. ~240 MB

3) SF_6 tracer:

Monthly mean 3D volumetric mixing ratio of SF_6 from the last simulation year for each of 11 terrestrial basis functions. This represents 528 (11 regions x 12 months) 3D fields. ~15 MB

Monthly mean volumetric mixing ratio maps (two lowest model layers) of SF_6 from the last simulation year for each of 11 terrestrial basis functions. This represents 1056 (11 regions x 12 months x 2 layers) surface map fields. ~3.3 MB

---

[8] If your model runs at a timestep longer than four hours, report at that timestep.

[9] This represents the four pre-subtracted tracers (1990 fossil fuel, 1995 fossil fuel, neutral biosphere and net oceanic exchange) plus the 22 basis function (11 terrestrial and 11 ocean).

4) Winds:

Monthly mean 3D fields of u, v, and ω for all three years. This represents 144 (3 winds x 36 months) 3D fields (if using a single year of winds, simply repeat). ~12 MB.

Monthly mean surface map fields of u and v for all three years. This represents 96 (2 winds x 36 months) surface map fields (if using a single year of winds, simply repeat). ~1 MB

5) Single Point Reporting:

High frequency station location reporting for CO_2 surface volumetric mixing ratio, u, and v. This data will be placed into four arrays, the first containing u and v for the entire three year period, the second containing the pre-subtracted tracer CO_2 mixing ratios for the entire three year period, the third containing the terrestrial basis function CO_2 mixing ratios for the twelve months following the pulsed emission (11 tracers, 13 reported months each), and the fourth containing the oceanic basis function CO_2 mixing ratios for the twelve months following the pulsed emission (11 tracers, 13 reported months each). The first two arrays (the winds and the pre-subtracted tracers) will have a time dimension of three years. For the basis function tracers, the time dimension is one year, starting at the time of the particular pulsed emission and then reporting the year that follows. Please use a 4 hour timestep for reporting this data (see footnote 8) and start reporting at the first timestep rather than zero. The first array comes to (228 stations x 6570 timesteps x 2 winds) ~12 MB. The second array comes to (228 stations x 6570 timesteps x 4 CO_2 tracers) ~24 MB. The third array comes to (228 stations x ~2376 timesteps[10] x 11 regions x 12 month-pulses) ~270 MB. The fourth array comes to (228 stations x ~2376 timesteps x 11 regions x 12 month-pulses) ~270 MB or a total of ~576 MB

Please submit a text file containing the latitude, longitude, altitude, model level (or levels interpolated between) and the surface type for each site (see Appendix C).

6) $\overline{\beta}$: (for those computing an explicit surface terrain mask)

Monthly mean 3D $\overline{\beta}$ values for all three years. This represents 36 (1 β x 36 months) 3D fields. ~3 MB

7) Land/Sea mask

This represents 1 surface map field. ~0.02 MB

Total ~2 GB

Required output - Level III

There is no prescription for level III output. For purposes of comparison, some description of the resulting source/sink estimates would be necessary. If you are planning on performing level III, please contact Kevin Gurney (kevin@atmos.colostate.edu)

Output file format and structure

Appendix A contains code that will place your output into a netCDF file for transference to the central coordinator.

[10] 2376 timesteps consists of a 12 month year plus one 31 day month. In order to make the netCDF file construction simple, please report 2376 timesteps even if the last month contains less than 31 days.

G. Timetable

Please have your final level I and level II output submitted to us by March 2000. All submissions can be made to our anonymous ftp site (dendrus.atmos.colostate.edu, then go to the "transcom/" subdirectory) or by another medium such as CD-ROM.

We are planning to have the analysis of levels I and II complete by March 2001. The level III collation, and the TransCom 3 publications and experimental wrap-up are planned to be complete by December of the year 2001.

Meeting dates:

1) **May 18th and 19th 2000: Paris, France**
2) **March 2001: Australia (tentative)**

Appendix A. Model output to netCDF format

The following code will perform direct creation of a netCDF output data file. I have structured this as a subroutine (*write_cdf*). It is up to the individual groups on how exactly they want to perform the subroutine calls - that is, from within you forward simulations or from an off-line piece of code (for example, you may already have output algorithms that send data to binary files - these could then be reopened and the subroutine below used to transfer this to netCDF). There are two different subroutines listed here, one for the level I experiment and one for the level II experiment. The only difference between these two is the size of a few of the output arrays.

In order to perform the netCDF writing, you need to have the netCDF libraries installed on your system. The URL for a site from which you can download the netCDF binaries is:

http://unidata.ucar.edu/packages/netcdf/index.html

If you are interested in viewing the output files, there are a large number of applications that have been written for this. A list with links can be found at:

http://unidata.ucar.edu/packages/netcdf/utilities.html

We have been using Ferret and find it quite useful for quick and easy viewing. Documentation and code are at:

http://ferret.wrc.noaa.gov/Ferret/

For level I

```
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cO                                                                                    OO
cO This program writes the TransCom 3 Level 1 output data to a single OO
cO netCDF file.                                                                       OO
cO                                                                                    OO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

cOOOOOOOOO begin notes OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
C
C TRANSCOM MODELERS!
C
C PLEASE USE THE FOLLOWING CONVENTIONS WHEN FILLING THE LEVEL 1 ARRAYS:
C (all the array descriptions are provided below)
C
C PLEASE WRITE THE GRIDCELL CENTERS ("LONVECT" AND "LATVECT" VECTORS)
C IN UNITS OF DEGREES EAST (ie. -175, -170.......0, 5.....170, 175) AND
C DEGREES NORTH (ie. -85, 80......0, 5......80, 85)
C
C PLEASE ENSURE THAT THE FIRST GRIDCELL IN 2D MAPS AND 3D FIELDS IS
C     - AT THE DATELINE RATHER THAN AT GREENWICH
C     - AT THE SOUTH POLE RATHER THAN THE NORTH POLE
C     - AT 1000 MB RATHER THAN 100 MB (FOR 3D FIELDS)
C
C THESE ARRAYS WILL THEN BE WRITTEN MOVING EAST (FOR LONGITUDINAL
C DIRECTION), MOVING NORTH (FOR LATITUDINAL DIRECTION) AND MOVING UP
C (FOR VERTICAL DIRECTION).
C
C PLEASE MAINTAIN THE BASIS FUNCTION REGION NUMBERING (ie. REGION 1 =
C NORTH AMERICAN BOREAL, 2 = NORTH AMERICAN TEMPERATE......)
C
C UNITS:
C     - CO2 CONCENTRATION VALUES AS VOLUMETRIC PARTS PER MILLION (PPMV)
```

```
C     - SF6 CONCENTRATION VALUES AS VOLUMETRIC PARTS PER TRILLION (PPTV)
C     - U AND V WINDS IN METERS PER SECOND
C     - OMEGA WIND IN PASCALS/SECOND
C
C MISSING VALUES ARE REPORTED AS 1.0 x 10^36
C
C LAND/SEA MASK ("LSMASK") AND TERRAIN MASK ("BETA") REPORT AS UNITLESS
C FOLLOWING THE SAME GRIDDING RULES LISTED ABOVE.
C
C
C                          ALLOCATABLE ARRAYS
C
C THIS PROGRAM HAS BEEN WRITTEN USING ALLOCATABLE ARRAYS. IF A
C SUBROUTINE IS USED TO FILL THE OUTPUT ARRAYS, CODE MUST BE WRITTEN IN
C THE INTERFACE SECTION (this is noted below). IF THE ARRAYS ARE FILLED
C IN THE MAIN BODY OF THE PROGRAM, NO INTERFACE CODE IS REQUIRED.
C
C
C To compile on Dendrus (SGI O2K) at Colorado State University:
C
C f90 make.output.11.f -o make.11 -I/usr/local/netcdf/include
C -L/usr/local/lib -lnetcdf
C
cOOOOOOOOO end notes OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO



      Program makeout

cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin variable declarations OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Implicit none
      include 'netcdf.inc'

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccc T3 modelers: adjust the following integer parameters!! cccccc

      integer, parameter                        :: im=72,jm=44,pm=9,
     ^                                              timestep=6,
     ^                                              tlen=365*timestep,
     ^                                              statcnt=228
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer                                   :: c
      integer, parameter                        :: mtot   =12,lreg=11,
     ^                                              windnum=2,tractot=26

      real, dimension(:),          allocatable :: lonvect,latvect,
     ^                                              presvect,mvect,lvect,
     ^                                              svect,tvect,trvect,
     ^                                              wvect

      real, dimension(:,:),        allocatable :: lsmask

      real, dimension(:,:,:),      allocatable :: ff90_s,  ff90_sm1,
     ^                                              ff95_s,  ff95_sm1,
     ^                                              bios_s,  bios_sm1,
     ^                                              ocean_s,ocean_sm1,
     ^                                              u_s,v_s,
     ^                                              statco2,statwind

      real, dimension(:,:,:,:),    allocatable :: ff90,ff95,bios,ocean,
     ^                                              u,v,omega,beta,
     ^                                              sf6_s,sf6_sm1,
     ^                                              landunit_s,
     ^                                              landunit_sm1,
     ^                                              oceanunit_s,
     ^                                              oceanunit_sm1
```

```
      real, dimension(:,:,:,:,:),  allocatable :: landunit,oceanunit,sf6

c000000000 end variable declarations 0000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin main program variable description 0000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c
c im            : The number of longitudinal grid cells reported.
c jm            : The number of latitudinal grid cells reported.
c pm            : The number of pressure levels reported.
c timestep      : The number of timesteps per day reported for the high
c                 frequency data reporting.
c tlen          : The total number of timesteps reported for the high
c                 frequency data (timestep*365)
c statcnt       : The number of CO2 observational stations
c mtot          : The number of months written: 12
c lreg          : The number of terrestrial basis function regions: 11
c windnum       : The number of wind directions reported in the high
c                 frequency data: 2
c tractot       : The total number of tracers (number of basis functions
c                 + 4 pre-subtracted tracers): 26
c lonvect       : A vector containing the coordinates of the reported
c                 longitudinal grid centers. First element starts near
c                 the dateline and moves East. Units: degrees east
c latvect       : A vector containing the coordinates of the reported
c                 latitudinal grid centers. First element starts near the
c                 South Pole. Units: degrees north
c presvect      : A vector containing the coordinates of the reported
c                 vertical grid centers in millibars. First element
c                 starts at 1000 mb.
c mvect         : A vector of the month index: 1 through 12
c lvect         : A vector of the region index: 1 through 11
c svect         : A vector of the station index: 1 through 228
c tvect         : A vector of the high frequency timestep index: 1
c                 through 2190
c trvect        : A vector of the tracer index: 1 through 26
c wvect         : A vector of the wind direction index: 1 through 2
c lsmask        : Land/sea mask. This is a 2D array in which both
c                 dimensions represent space.
c ff90          : 1990 pre-subtracted fossil-fuel CO2 mixing ratio. This
c                 is a 4D array in which the first 3 dimensions are space
c                 and the last is time.
c ff95          : 1995 pre-subtracted fossil-fuel CO2 mixing ratio. This
c                 is a 4D array in which the first 3 dimensions are space
c                 and the last is time.
c bios          : Neutral biosphere pre-subtracted CO2 mixing ratio. This
c                 is a 4D array in which the first 3 dimensions are space
c                 and the last is time.
c ocean         : Ocean exchange pre-subtracted CO2 mixing ratio. This is
c                 a 4D array in which the first 3 dimensions are space
c                 and the last is time.
c ff90_s        : 1990 pre-subtracted fossil-fuel CO2 surface layer
c                 mixing ratio. This is a 3D array in which the first 2
c                 dimensions are space and the last is time.
c ff90_sm1      : 1990 pre-subtracted fossil-fuel CO2 mixing ratio in
c                 layer above the surface layer. This is a 3D array in
c                 which the first 2 dimensions are space, last is time
c ff95_s        : 1995 pre-subtracted fossil-fuel CO2 surface layer
c                 mixing ratio. This is a 3D array in which the first 2
c                 dimensions are space and the last is time.
c ff95_sm1      : 1995 pre-subtracted fossil-fuel CO2 mixing ratio in
c                 layer above the surface layer. This is a 3D array in
c                 which the first 2 dimensions are space, last is time.
c bios_s        : Neutral biosphere pre-subtracted CO2 surface layer
c                 mixing ratio. This is a 3D array in which the first 2
c                 dimensions are space and the last is time.
c bios_sm1      : Neutral biosphere pre-subtracted CO2 mixing ratio in
c                 layer above the surface layer. This is a 3D array in
c                 which the first 2 dimensions are space, last is time.
```

```
c ocean_s       : Ocean exchange pre-subtracted CO2 surface layer mixing
c                 ratio. This is a 3D array in which the first 2
c                 dimensions are space and the last is time.
c ocean_sml     : Ocean exchange pre-subtracted CO2 mixing ratio in layer
c                 above the surface layer. This is a 3D array in which
c                 the first 2 dimensions are space and the last is time.
c landunit      : Terrestrial carbon basis function CO2 mixing ratio.
c                 This is a 5D array in which the first 3 dimensions are
c                 space, the fourth indexes the basis function region,
c                 and the last dimension is time.
c oceanunit     : Oceanic carbon basis function CO2 mixing ratio. This
c                 is a 5D array in which the first 3 dimensions are
c                 space, the fourth indexes the basis function region,
c                 and the last dimension is time.
c landunit_s    : Terrestrial carbon basis function surface layer CO2
c                 mixing ratio. This is a 4D array in which the first 2
c                 dimensions are space, the 3rd dimension indexes the
c                 basis function region, and the last dimension is time.
c landunit_sml  : Terrestrial carbon basis function CO2 mixing ratio in
c                 layer above the surface layer. This is a 4D array in
c                 which the first 2 dimensions are space, the 3rd
c                 dimension indexes the basis function region, and the
c                 last dimension is time.
c oceanunit_s   : Oceanic carbon basis function surface layer CO2 mixing
c                 ratio. This is a 4D array in which the first 2
c                 dimensions are space, the 3rd dimension indexes the
c                 basis function region, and the last dimension is time.
c oceanunit_sml: Oceanic carbon basis function CO2 mixing ratio in layer
c                 above the surface layer. This is a 4D array in which
c                 the first 2 dimensions are space, the 3rd dimension
c                 indexes the basis function region, and the last
c                 dimension is time.
c sf6           : SF6 mixing ratio. This is a 5D array in which the first
c                 3 dimensions are space, the fourth indexes the basis
c                 function region, and the last dimension is time.
c sf6_s         : Surface layer SF6 mixing ratio. This is a 4D array in
c                 which the first 2 dimensions are space, the 3rd
c                 dimension indexes the basis function region, and the
c                 last dimension is time.
c sf6_sml       : SF6 mixing ratio in layer above the surface layer. This
c                 is a 4D array in which the first 2 dimensions are
c                 space, the 3rd dimension indexes the basis function
c                 region, and the last dimension is time.
c u             : Longitudinal wind velocity. This is a 4D array in which
c                 the first three dimensions are space, the last is time.
c v             : Latitudinal wind velocity. This is a 4D array in which
c                 the first three dimensions are space, the last is time.
c omega         : vertical pressure velocity. This is a 4D array in which
c                 the first three dimensions are space, the last is time.
c u_s           : Longitudinal surface wind velocity. This is a 3D array:
c                 the first two dimensions are space, the last is time.
c v_s           : Meridional surface wind velocity. This is a 3D array:
c                 the first two dimensions are space, the last is time.
c statco2       : A 3D array containing the CO2 single point reporting
c                 output. The first dimension indexes the station
c                 location (please use the order listed in the TransCom
c                 protocol), the second indexes the tracer, and the last
c                 indexes time. Please use the following order for the
c                 tracer dimension: the pre-subtracted 1990 fossil fuel,
c                 the pre-subtracted 1995 fossil fuel, the neutral
c                 biosphere, the pre-subtracted oceanic exchange, the
c                 terrestrial basis functions (1 through 11), and the
c                 oceanic basis functions (1 through 11).
c statwind      : A 3D array containing the wind single point reporting
c                 output. The first dimension indexes the station
c                 location, the second indexes the reported wind
c                 direction: u and v, and the last indexes time.
c beta          : Terrain mask. This is a 4D array in which the
c                 first three dimensions are space and the last is time.
c
cOOOOOOOOO end main program variable description OOOOOOOOOOOOOOOOOOOOOO
```

```
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin interface module OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Interface

COOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
C
C TRANSCOM MODELERS!
C
C IF YOU ARE CALLING A SUBROUTINE TO FILL THE OUTPUT ARRAYS, AN
C INTERFACE MODULE WILL BE REQUIRED HERE (due to the use of allocatable
C arrays).
C
COOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

        Subroutine write_cdf_1(im,jm,pm,mtot,lreg,statcnt,tlen,
     ^                         tractot,windnum,
     ^                         lonvect,latvect,presvect,mvect,lvect,
     ^                         svect,tvect,trvect,wvect,
     ^                         ff90,ff90_s,ff90_sm1,ff95,ff95_s,ff95_sm1,
     ^                         bios,bios_s,bios_sm1,
     ^                         ocean,ocean_s,ocean_sm1,
     ^                         landunit,landunit_s,landunit_sm1,
     ^                         oceanunit,oceanunit_s,oceanunit_sm1,
     ^                         sf6,sf6_s,sf6_sm1,u,u_s,v,v_s,omega,
     ^                         statco2,statwind,lsmask,beta)

          integer                                 :: im,jm,pm,mtot,lreg,
     ^                                               statcnt,tlen,
     ^                                               tractot,windnum

          real, dimension(:),          intent(in):: lonvect,latvect,
     ^                                               presvect,mvect,lvect,
     ^                                               svect,tvect,trvect,
     ^                                               wvect

          real, dimension(:,:),        intent(in):: lsmask

          real, dimension(:,:,:),      intent(in):: ff90_s,  ff95_sm1,
     ^                                               ff95_s,  ff90_sm1,
     ^                                               bios_s,  bios_sm1,
     ^                                               ocean_s,ocean_sm1,
     ^                                               u_s,v_s,
     ^                                               statco2,statwind

          real, dimension(:,:,:,:),    intent(in):: ff90,ff95,bios,ocean,
     ^                                               u,v,omega,beta,
     ^                                               sf6_s,sf6_sm1,
     ^                                               landunit_s,
     ^                                               landunit_sm1,
     ^                                               oceanunit_s,
     ^                                               oceanunit_sm1

          real, dimension(:,:,:,:,:),intent(in):: landunit,oceanunit,sf6

        End Subroutine write_cdf_1

      End interface

cOOOOOOOOO end interface module OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin allocation of the TransCom3 level 1 arrays OOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Allocate(mvect (mtot   )); mvect  = (/(c,c=1,mtot)   /)
```

```
      Allocate(lvect (lreg   )); lvect  = (/(c,c=1,lreg)   /)
      Allocate(svect (statcnt)); svect  = (/(c,c=1,statcnt)/)
      Allocate(tvect (tlen   )); tvect  = (/(c,c=1,tlen)   /)
      Allocate(trvect(tractot)); trvect = (/(c,c=1,tractot)/)
      Allocate(wvect (windnum)); wvect  = (/(c,c=1,windnum)/)

      Allocate(lonvect      (im))
      Allocate(latvect      (jm))
      Allocate(presvect     (pm))
      Allocate(lsmask       (im,jm))
      Allocate(ff90_s       (im,jm,mtot))
      Allocate(ff90_sm1     (im,jm,mtot))
      Allocate(ff95_s       (im,jm,mtot))
      Allocate(ff95_sm1     (im,jm,mtot))
      Allocate(bios_s       (im,jm,mtot))
      Allocate(bios_sm1     (im,jm,mtot))
      Allocate(ocean_s      (im,jm,mtot))
      Allocate(ocean_sm1    (im,jm,mtot))
      Allocate(u_s          (im,jm,mtot))
      Allocate(v_s          (im,jm,mtot))
      Allocate(ff90         (im,jm,pm,mtot))
      Allocate(ff95         (im,jm,pm,mtot))
      Allocate(bios         (im,jm,pm,mtot))
      Allocate(ocean        (im,jm,pm,mtot))
      Allocate(u            (im,jm,pm,mtot))
      Allocate(v            (im,jm,pm,mtot))
      Allocate(omega        (im,jm,pm,mtot))
      Allocate(beta         (im,jm,pm,mtot))
      Allocate(sf6_s        (im,jm,lreg,mtot))
      Allocate(sf6_sm1      (im,jm,lreg,mtot))
      Allocate(landunit_s   (im,jm,lreg,mtot))
      Allocate(landunit_sm1 (im,jm,lreg,mtot))
      Allocate(oceanunit_s  (im,jm,lreg,mtot))
      Allocate(oceanunit_sm1(im,jm,lreg,mtot))
      Allocate(sf6          (im,jm,pm,lreg,mtot))
      Allocate(landunit     (im,jm,pm,lreg,mtot))
      Allocate(oceanunit    (im,jm,pm,lreg,mtot))
      Allocate(statco2      (statcnt,tractot,tlen))
      Allocate(statwind     (statcnt,windnum,tlen))

c000000000 end allocation of the TransCom3 level 1 arrays 00000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin call to filler subroutine 0000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000000
C
C TRANSCOM MODELERS!
C
C FILL THE OUTPUT ARRAYS HERE. THIS CAN BE DONE WITH EITHER A SUBROUTINE
C CALL (which requires code in the interface module above) OR PERFORMED
C DIRECTLY IN THIS PORTION OF THE PROGRAM'S MAIN BODY.
C
c000000000 end call to filler subroutine 00000C0000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin call to write_cdf subroutine 0000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000000

      Call write_cdf_1(im,jm,pm,mtot,lreg,statcnt,tlen,
     ^                 tractot,windnum,
     ^                 lonvect,latvect,presvect,mvect,lvect,
     ^                 svect,tvect,trvect,wvect,
     ^                 ff90,ff90_s,ff90_sm1,ff95,ff95_s,ff95_sm1,
     ^                 bios,bios_s,bios_sm1,
     ^                 ocean,ocean_s,ocean_sm1,
     ^                 landunit,landunit_s,landunit_sm1,
     ^                 oceanunit,oceanunit_s,oceanunit_sm1,
     ^                 sf6,sf6_s,sf6_sm1,u,u_s,v,v_s,omega,
     ^                 statco2,statwind,lsmask,beta)
```

```
c000000000 end call to write_cdf subroutine 000000000000000000000000000

      Write(*,*) 'output.L1.nc written'

      Stop
      End Program makeout

c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0                      WRITE_CDF_1 SUBROUTINE                              00
c0                                                                          00
c0 This subroutine writes all of the TransCom 3 level I output to a         00
c0 netCDF file called "output.L1.nc"                                        00
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      Subroutine write_cdf_1(im,jm,pm,mtot,lreg,statcnt,tlen,
     ^                       tractot,windnum,
     ^                       lonvect,latvect,presvect,mvect,lvect,
     ^                       svect,tvect,trvect,wvect,
     ^                       ff90,ff90_s,ff90_sm1,ff95,ff95_s,ff95_sm1,
     ^                       bios,bios_s,bios_sm1,
     ^                       ocean,ocean_s,ocean_sm1,
     ^                       landunit,landunit_s,landunit_sm1,
     ^                       oceanunit,oceanunit_s,oceanunit_sm1,
     ^                       sf6,sf6_s,sf6_sm1,u,u_s,v,v_s,omega,
     ^                       statco2,statwind,lsmask,beta)


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable declarations 000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      implicit none

      include 'netcdf.inc'

      integer                                  :: im,jm,pm,mtot,lreg,
     ^                                            statcnt,tlen,
     ^                                                 tractot,windnum,r

      real, parameter                          :: misval=1.0e+36

      real, dimension(:),           intent(in) :: lonvect,latvect,
     ^                                            presvect,mvect,lvect,
     ^                                            svect,tvect,trvect,
     ^                                            wvect

      real, dimension(:,:),         intent(in) :: lsmask

      real, dimension(:,:,:),       intent(in) :: ff90_s,  ff90_sm1,
     ^                                            ff95_s,  ff95_sm1,
     ^                                            bios_s,  bios_sm1,
     ^                                            ocean_s,ocean_sm1,
     ^                                            u_s,v_s,
     ^                                            statco2,statwind

      real, dimension(:,:,:,:),     intent(in) :: ff90,ff95,bios,ocean,
     ^                                            u,v,omega,beta,
     ^                                            sf6_s,sf6_sm1,
     ^                                            landunit_s,
```

```
     ^                                          landunit_sml,
     ^                                          oceanunit_s,
     ^                                          oceanunit_sml

      real, dimension(:,:,:,:,:),   intent(in) :: landunit,oceanunit,sf6

      character (len=10), dimension(lreg)      :: sf6name
      character (len=80)                       :: regn
      character (len=2),  dimension(lreg)      :: num
      character (len=15), dimension(lreg)      :: landname
      character (len=16), dimension(lreg)      :: oceanname

ccccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccc

      integer                                  :: status,ncid,imid,jmid,
     ^                                            pmid,mtotid,lregid,
     ^                                            statcntid,tlenid,
     ^                                            tractotid,windnumid,
     ^                                            lonid,latid,presid,
     ^                                            mid,lid,sid,tid,trid,
     ^                                            wid,ff90id,ff90_sid,
     ^                                            ff90_sm1id,ff95id,
     ^                                            ff95_sid,ff95_sm1id,
     ^                                            biosid,bios_sid,
     ^                                            bios_sm1id,oceanid,
     ^                                            ocean_sid,ocean_sm1id,
     ^                                            landunit_regid(11),
     ^                                            landunit_sid,
     ^                                            landunit_sm1id,
     ^                                            oceanunit_regid(11),
     ^                                            oceanunit_sid,
     ^                                            oceanunit_sm1id,
     ^                                            sf6_regid(11),
     ^                                            sf6_sid,sf6_sm1id,
     ^                                            uid,u_sid,vid,v_sid,
     ^                                            omegaid,
     ^                                            statCO2id,statwindid,
     ^                                            lsmaskid,betaid,
     ^                                            vdims2(2),vdims3(3),
     ^                                            vdims4(4),
     ^                                            start3(3),count3(3),
     ^                                            start4(4),count4(4)

c000000000 end variable declarations 000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable description 000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000
c
ccccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccccc
c
c misval        : The designated TransCom missing value: 1.0e+36.
c r             : Loop index. Loops over lreg.
c sf6name       : A character array containing the variable names for
c                 the series of netCDF arrays associated with the SF6
c                 basis function region 3D concentrations.
c regn          : A character variable used for the 'long_name' netCDF
c                 attribute.
c num           : A character array containing the basis function region
c                 number: 01 through 11.
c landname      : A character array containing the variable names for
c                 the series of netCDF arrays associated with the CO2
c                 terrestrial basis function region 3D concentrations.
c oceanname     : A character array containing the variable names for
c                 the series of netCDF arrays associated with the CO2
c                 oceanic basis function region 3D concentrations.
c
ccccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccccc
c
c ncid          : The netCDF ID representing the open file.
```

```
c XXXXXid       : A series of ID's that correspond to those variables
c                 names passed into the write_cdf subroutine. There are
c                 three exceptions to this scheme. They are:
c      landunit_reg# : These ID's reduce the incoming landunit 5D array
c                      into 4D arrays. There are 11 of these ID's, one
c                      one for each terrestrial basis function region.
c      oceanunit_reg#: These ID's reduce the incoming oceanunit 5D array
c                      into 4D arrays. There are 11 of these ID's, one
c                      one for each oceanic basis function region.
c      sf6_reg#      : These ID's reduce the incoming sf6 5D array
c                      into 4D arrays. There are 11 of these ID's, one
c                      one for each terrestrial basis function region.
c vdims#        : These are netCDF integer vectors whose elements
c                 denote the dimensions of the netCDF arrays.
c start#        : These are netCDF integer vectors whose elements denote
c                 where in the netCDF array the first element of a given
c                 write statement should start.
c count#        : These are netCDF integer vectors whose elements denote
c                 the edge lengths along each dimension of a given write
c                 statement.
c
c000000000 end variable description 0000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable initialization 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000

      sf6name(1)  = 'sf6_reg_01'; num(1)  = '01'
      sf6name(2)  = 'sf6_reg_02'; num(2)  = '02'
      sf6name(3)  = 'sf6_reg_03'; num(3)  = '03'
      sf6name(4)  = 'sf6_reg_04'; num(4)  = '04'
      sf6name(5)  = 'sf6_reg_05'; num(5)  = '05'
      sf6name(6)  = 'sf6_reg_06'; num(6)  = '06'
      sf6name(7)  = 'sf6_reg_07'; num(7)  = '07'
      sf6name(8)  = 'sf6_reg_08'; num(8)  = '08'
      sf6name(9)  = 'sf6_reg_09'; num(9)  = '09'
      sf6name(10) = 'sf6_reg_10'; num(10) = '10'
      sf6name(11) = 'sf6_reg_11'; num(11) = '11'

      landname(1)  = 'landunit_reg_01'
      landname(2)  = 'landunit_reg_02'
      landname(3)  = 'landunit_reg_03'
      landname(4)  = 'landunit_reg_04'
      landname(5)  = 'landunit_reg_05'
      landname(6)  = 'landunit_reg_06'
      landname(7)  = 'landunit_reg_07'
      landname(8)  = 'landunit_reg_08'
      landname(9)  = 'landunit_reg_09'
      landname(10) = 'landunit_reg_10'
      landname(11) = 'landunit_reg_11'

      oceanname(1)  = 'oceanunit_reg_01'
      oceanname(2)  = 'oceanunit_reg_02'
      oceanname(3)  = 'oceanunit_reg_03'
      oceanname(4)  = 'oceanunit_reg_04'
      oceanname(5)  = 'oceanunit_reg_05'
      oceanname(6)  = 'oceanunit_reg_06'
      oceanname(7)  = 'oceanunit_reg_07'
      oceanname(8)  = 'oceanunit_reg_08'
      oceanname(9)  = 'oceanunit_reg_09'
      oceanname(10) = 'oceanunit_reg_10'
      oceanname(11) = 'oceanunit_reg_11'

c000000000 end variable initialization 00000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 open netCDF file and define the dimensions 0000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
      status = NF_CREATE('output.L1.nc',0,ncid)
      write(*,*) 'output.L1.nc created'

      status = NF_DEF_DIM(ncid,'longitude',im,imid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'latitude',jm,jmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'height',pm,pmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'time',mtot,mtotid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'land_region',lreg,lregid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'station_count',statcnt,statcntid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'tlen',tlen,tlenid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'tracer_total',tractot,tractotid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'wind_dir',windnum,windnumid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end open of netCDF file and dimension definition 000000000000


c0000000000C0000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 0000000000000000000000000000000000000000000000
c0000000000C0000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (latitude,longitude,pressure,time,regions) ccccccccccc

      status = NF_DEF_VAR(ncid,'longitude',NF_FLOAT,1,imid,lonid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'latitude',NF_FLOAT,1,jmid,latid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'height',NF_FLOAT,1,pmid,presid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'time',NF_FLOAT,1,mtotid,mid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'land_region',NF_FLOAT,1,lregid,lid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'station_count',NF_FLOAT,1,statcntid,sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'tlen',NF_FLOAT,1,tlenid,tid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'tracer_total',NF_FLOAT,1,tractotid,trid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'wind_dir',NF_FLOAT,1,windnumid,wid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc land/ocean mask (lsmask) ccccccccccccccccccccccccccccccccccc

      vdims2(1) = imid; vdims2(2) = jmid

      status = NF_DEF_VAR(ncid,'lsmask',NF_FLOAT,2,vdims2,lsmaskid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted carbon surface maps ccccccccccccccccccccccccc

c ff90_s,ff90_sm1,ff95_s,ff95_sm1,bios_s,bios_sm1,ocean_s,ocean_sm1

      vdims3(1) = imid; vdims3(2) = jmid; vdims3(3) = mtotid

      status = NF_DEF_VAR(ncid,'ff90_s',NF_FLOAT,3,vdims3,ff90_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff90_sm1',NF_FLOAT,3,vdims3,ff90_smlid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff95_s',NF_FLOAT,3,vdims3,ff95_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff95_sm1',NF_FLOAT,3,vdims3,ff95_smlid)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
      status = NF_DEF_VAR(ncid,'bios_s',NF_FLOAT,3,vdims3,bios_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'bios_sm1',NF_FLOAT,3,vdims3,bios_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ocean_s',NF_FLOAT,3,vdims3,ocean_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status =NF_DEF_VAR(ncid,'ocean_sm1',NF_FLOAT,3,vdims3,ocean_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc surface wind maps (u_s,v_s) ccccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'u_s',NF_FLOAT,3,vdims3,u_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'v_s',NF_FLOAT,3,vdims3,v_sid)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc pre-subtracted 3D carbon fields (ff90,ff95,bios,ocean) cccccc

      vdims4(1)=imid; vdims4(2)=jmid; vdims4(3)=pmid; vdims4(4)=mtotid

      status = NF_DEF_VAR(ncid,'ff90',NF_FLOAT,4,vdims4,ff90id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff95',NF_FLOAT,4,vdims4,ff95id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'bios',NF_FLOAT,4,vdims4,biosid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ocean',NF_FLOAT,4,vdims4,oceanid)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc terrain mask (beta) ccccccccccccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'beta',NF_FLOAT,4,vdims4,betaid)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc 3D wind fields (u,v,omega) ccccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'u',NF_FLOAT,4,vdims4,uid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'v',NF_FLOAT,4,vdims4,vid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'omega',NF_FLOAT,4,vdims4,omegaid)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc 3D SF6 fields (sf6_reg1,sf6_reg2......) ccccccccccccccccccc

      Do r = 1, lreg
         status = NF_DEF_VAR(ncid,sf6name(r),NF_FLOAT,4,vdims4,
     ^                       sf6_regid(r))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

cccccccccc SF6 surface maps (sf6_s,sf6_sm1) ccccccccccccccccccccccccccc

      vdims4(3) = lregid

      status = NF_DEF_VAR(ncid,'sf6_s',NF_FLOAT,4,vdims4,sf6_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'sf6_sm1',NF_FLOAT,4,vdims4,sf6_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc basis function surface maps ccccccccccccccccccccccccccccccc

c landunit_s,landunit_sm1,oceanunit_s,oceanunit_sm1

      status = NF_DEF_VAR(ncid,'landunit_s',NF_FLOAT,4,vdims4,
     ^                    landunit_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'landunit_sm1',NF_FLOAT,4,vdims4,
     ^                    landunit_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
      status = NF_DEF_VAR(ncid,'oceanunit_s',NF_FLOAT,4,vdims4,
     ^                    oceanunit_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'oceanunit_sm1',NF_FLOAT,4,vdims4,
     ^                    oceanunit_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccccc
c
c landunit_reg1,landunit_reg2...., oceanunit_reg1,oceanunit_reg2.......

      vdims4(3) = pmid

      Do r = 1, lreg
         status = NF_DEF_VAR(ncid,landname(r),NF_FLOAT,4,vdims4,
     ^                       landunit_regid(r))
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_DEF_VAR(ncid,oceanname(r),NF_FLOAT,4,vdims4,
     ^                       oceanunit_regid(r))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

ccccccccccc time series (statCO2,statwind) ccccccccccccccccccccccccccccc

      vdims3(1) = statcntid; vdims3(2) = tractotid; vdims3(3) = tlenid

      status =NF_DEF_VAR(ncid,'station_CO2',NF_FLOAT,3,vdims3,statCO2id)
      if (status .ne. nf_noerr) call handle_err(status)

      vdims3(1) = statcntid; vdims3(2) = windnumid; vdims3(3) = tlenid

      status = NF_DEF_VAR(ncid,'station_wind',NF_FLOAT,3,vdims3,
     ^                    statwindid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end define variables 0000000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin apply attributes 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc The vectors (latitude,longitude,pressure,time,regions) cccccc

      status = NF_PUT_ATT_TEXT(ncid,lonid,'long_name',29,
     ^                         'Longitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lonid,'units',12,'degrees_east')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,latid,'long_name',28,
     ^                         'Latitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,latid,'units',13,'degrees_north')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,presid,'long_name',25,
     ^                         'Vertical gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,presid,'units',9,'millibars')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,mid,'long_name',5,'month')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,mid,'units',5,'month')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,lid,'long_name',13,'region number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lid,'units',6,'region')
      if (status .ne. nf_noerr) call handle_err(status)
```

```
      status = NF_PUT_ATT_TEXT(ncid,sid,'long_name',14,'station number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,sid,'units',7,'station')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,tid,'long_name',31,
     ^                         'elapsed hours in one year span')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,tid,'units',5,'hours')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,trid,'long_name',13,'tracer number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,trid,'units',6,'tracer')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,wid,'long_name',21,
     ^                         'wind direction number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,wid,'units',14,'wind direction')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc land/ocean mask (lsmask) ccccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,lsmaskid,'long_name',13,
     ^                         'Land/sea mask')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lsmaskid,'units',8,'unitless')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,lsmaskid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted carbon surface maps ccccccccccccccccccccccccc

c ff90_s,ff90_sm1,ff95_s,ff95_sm1,bios_s,bios_sm1,ocean_s,ocean_sm1

      status = NF_PUT_ATT_TEXT(ncid,ff90_sid,'long_name',47,
     ^                 'Pre-subtracted 1990 fossil fuel CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff90_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff90_sid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff95_sid,'long_name',47,
     ^                'Pre-subtracted 1995 fossil fuel CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff95_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff95_sid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,bios_sid,'long_name',40,
     ^                   'Pre-subtracted biosphere CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,bios_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,bios_sid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ocean_sid,'long_name',38,
     ^                         'Pre-subtracted ocean CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ocean_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ocean_sid,'missing_value',NF_FLOAT,
```

```
     ^                          1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff90_sm1id,'long_name',60,
     ^    'Pre-subtracted 1990 fossil fuel CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff90_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff90_sm1id,'missing_value',NF_FLOAT,
     ^                          1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff95_sm1id,'long_name',60,
     ^    'Pre-subtracted 1995 fossil fuel CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff95_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff95_sm1id,'missing_value',NF_FLOAT,
     ^                          1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,bios_sm1id,'long_name',53,
     ^         'Pre-subtracted biosphere CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,bios_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,bios_sm1id,'missing_value',NF_FLOAT,
     ^                          1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ocean_sm1id,'long_name',51,
     ^           'Pre-subtracted oceanic CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ocean_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status =NF_PUT_ATT_REAL(ncid,ocean_sm1id,'missing_value',NF_FLOAT,
     ^                          1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc surface wind maps (u_s,v_s)ccccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,u_sid,'long_name',34,
     ^                        'Longitudinal surface wind velocity')
      if (status .ne. nf_noerr) call handle_err(status)
      status =NF_PUT_ATT_TEXT(ncid,u_sid,'units',17,'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,u_sid,'missing_value',NF_FLOAT,1,
     ^                          misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,v_sid,'long_name',32,
     ^                       'Meridional surface wind velocity')
      if (status .ne. nf_noerr) call handle_err(status)
      status =NF_PUT_ATT_TEXT(ncid,v_sid,'units',17,'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,v_sid,'missing_value',NF_FLOAT,1,
     ^                          misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted 3D carbon fields (ff90,ff95,bios,ocean) cccccc

      status = NF_PUT_ATT_TEXT(ncid,ff90id,'long_name',44,
     ^                  'Pre-subtracted 1990 fossil fuel CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff90id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff90id,'missing_value',NF_FLOAT,1,
     ^                          misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff95id,'long_name',44,
```

```
     ^                    'Pre-subtracted 1995 fossil fuel CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff95id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff95id,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,biosid,'long_name',37,
     ^                        'Pre-subtracted biosphere CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,biosid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,biosid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,oceanid,'long_name',35,
     ^                       'Pre-subtracted oceanic CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,oceanid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,oceanid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc terrain mask (beta) ccccccccccccccccccccccccccccccccccccccc

      status =NF_PUT_ATT_TEXT(ncid,betaid,'long_name',12,'Terrain mask')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D wind fields (u,v,omega) ccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,uid,'long_name',26,
     ^                       'Longitudinal 3D wind field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,uid,'units',17,'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status=NF_PUT_ATT_REAL(ncid,uid,'missing_value',NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,vid,'long_name',24,
     ^                       'Meridional 3D wind field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,vid,'units',17,'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status=NF_PUT_ATT_REAL(ncid,vid,'missing_value',NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,omegaid,'long_name',22,
     ^                       'Vertical 3D wind field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,omegaid,'units',18,
     ^                        'pascals per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,omegaid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D SF6 fields (sf6_reg1,sf6_reg2,sf6_reg3.......) ccccccccccc

      Do r = 1, lreg
         regn   = 'SF6 basis function 3D concentration field: region '
     ^             //num(r)
         status = NF_PUT_ATT_TEXT(ncid,sf6_regid(r),'long_name',52,regn)
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_TEXT(ncid,sf6_regid(r),'units',4,'pptv')
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_REAL(ncid,sf6_regid(r),'missing_value',
     ^                            NF_FLOAT,1,misval)
         if (status .ne. nf_noerr) call handle_err(status)
```

```
      End do

ccccccccccc SF6 surface maps (sf6_s,sf6_sm1) ccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,sf6_sid,'long_name',30,
     ^                         'SF6 concentration map: surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,sf6_sid,'units',4,'pptv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,sf6_sid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,sf6_sm1id,'long_name',42,
     ^                   'SF6 concentration map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,sf6_sm1id,'units',4,'pptv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,sf6_sm1id,'missing_value',NF_FLOAT,
     ^                         1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc basis function surface maps ccccccccccccccccccccccccccccccc

c landunit_s,landunit_sm1,oceanunit_s,oceanunit_sm1

      status = NF_PUT_ATT_TEXT(ncid,landunit_sid,'long_name',38,
     ^                         'Terrestrial basis function surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,landunit_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,landunit_sid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,oceanunit_sid,'long_name',34,
     ^                         'Oceanic basis function surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,oceanunit_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,oceanunit_sid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,landunit_sm1id,'long_name',50,
     ^            'Terrestrial basis function layer above surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,landunit_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,landunit_sm1id,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,oceanunit_sm1id,'long_name',46,
     ^                'Oceanic basis function layer above surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,oceanunit_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,oceanunit_sm1id,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccc

c landunit_reg1,landunit_reg2...., oceanunit_reg1,oceanunit_reg2.......

      Do r = 1, lreg
         regn = '3D terrestrial basis function field: region '//num(r)
         status = NF_PUT_ATT_TEXT(ncid,landunit_regid(r),'long_name',46,
     ^                            regn)
         if (status .ne. nf_noerr) call handle_err(status)
```

```
          status=NF_PUT_ATT_TEXT(ncid,landunit_regid(r),'units',4,'ppmv')
          if (status .ne. nf_noerr) call handle_err(status)
          status = NF_PUT_ATT_REAL(ncid,landunit_regid(r),
     ^                             'missing_value',NF_FLOAT,1,misval)
          if (status .ne. nf_noerr) call handle_err(status)

          regn = '3D oceanic basis function field: region '//num(r)
          status = NF_PUT_ATT_TEXT(ncid,oceanunit_regid(r),'long_name',
     ^                            42,regn)
          if (status .ne. nf_noerr) call handle_err(status)
          status = NF_PUT_ATT_TEXT(ncid,oceanunit_regid(r),'units',4,
     ^                             'ppmv')
          if (status .ne. nf_noerr) call handle_err(status)
          status = NF_PUT_ATT_REAL(ncid,oceanunit_regid(r),
     ^                             'missing_value',NF_FLOAT,1,misval)
          if (status .ne. nf_noerr) call handle_err(status)

        End do

ccccccccccc time series (statCO2,statwind) ccccccccccccccccccccccccccccc

        status = NF_PUT_ATT_TEXT(ncid,statCO2id,'long_name',38,
     ^                       'High frequency station CO2 time series')
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_ATT_TEXT(ncid,statCO2id,'units',4,'ppmv')
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_ATT_REAL(ncid,statCO2id,'missing_value',NF_FLOAT,
     ^                           1,misval)
        if (status .ne. nf_noerr) call handle_err(status)

        status = NF_PUT_ATT_TEXT(ncid,statwindid,'long_name',39,
     ^                       'High frequency station wind time series')
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_ATT_TEXT(ncid,statwindid,'units',17,
     ^                         'meters per second')
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_ATT_REAL(ncid,statwindid,'missing_value',NF_FLOAT,
     ^                           1,misval)
        if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccccc

        status = NF_ENDDEF(ncid)
        if (status .ne. nf_noerr) call handle_err(status)

cOOOOOOOOO end apply attributes OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin filling variables OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

        status = NF_PUT_VAR_REAL(ncid,lonid,lonvect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,latid,latvect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,presid,presvect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,mid,mvect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,lid,lvect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,sid,svect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,tid,tvect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,trid,trvect)
        if (status .ne. nf_noerr) call handle_err(status)
        status = NF_PUT_VAR_REAL(ncid,wid,wvect)
        if (status .ne. nf_noerr) call handle_err(status)
```

```
ccccccccccc land/ocean mask (lsmask) ccccccccccccccccccccccccccccccccc

      status = NF_PUT_VAR_REAL(ncid,lsmaskid,lsmask)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted carbon surface maps ccccccccccccccccccccccccc

c ff90_s,ff90_sm1,ff95_s,ff95_sm1,bios_s,bios_sm1,ocean_s,ocean_sm1

      start3(1) = 1; start3(2) = 1; start3(3) = 1
      count3(1) = im; count3(2) = jm; count3(3) = mtot

      status = NF_PUT_VARA_REAL(ncid,ff90_sid,start3,count3,ff90_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff95_sid,start3,count3,ff95_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,bios_sid,start3,count3,bios_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ocean_sid,start3,count3,ocean_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff90_sm1id,start3,count3,ff90_sm1)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff95_sm1id,start3,count3,ff95_sm1)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,bios_sm1id,start3,count3,bios_sm1)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ocean_sm1id,start3,count3,
     ^                          ocean_sm1)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc surface wind maps (u_s,v_s)ccccccccccccccccccccccccccccccccc

      status = NF_PUT_VARA_REAL(ncid,u_sid,start3,count3,u_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,v_sid,start3,count3,v_s)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted 3D carbon fields (ff90,ff95,bios,ocean) cccccc

      start4(1) = 1; start4(2) = 1; start4(3) = 1; start4(4) = 1
      count4(1) = im; count4(2) = jm; count4(3) = pm; count4(4) = mtot

      status = NF_PUT_VARA_REAL(ncid,ff90id,start4,count4,ff90)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff95id,start4,count4,ff95)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,biosid,start4,count4,bios)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,oceanid,start4,count4,ocean)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc terrain mask (beta) ccccccccccccccccccccccccccccccccccccccccc

      status = NF_PUT_VARA_REAL(ncid,betaid,start4,count4,beta)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D wind fields (u,v,omega) ccccccccccccccccccccccccccccccccc

      status = NF_PUT_VARA_REAL(ncid,uid,start4,count4,u)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,vid,start4,count4,v)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,omegaid,start4,count4,omega)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D SF6 fields (sf6_reg1,sf6_reg2,sf6_reg3.......) ccccccccccc

      Do r = 1, lreg
         status = NF_PUT_VARA_REAL(ncid,sf6_regid(r),start4,count4,
     ^                             sf6(:,:,:,r,:))
         if (status .ne. nf_noerr) call handle_err(status)
```

```
      End do

ccccccccc SF6 surface maps (sf6_s,sf6_sm1) ccccccccccccccccccccccccccc

      count4(3) = lreg

      status = NF_PUT_VARA_REAL(ncid,sf6_sid,start4,count4,sf6_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,sf6_sm1id,start4,count4,sf6_sm1)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc basis function surface maps ccccccccccccccccccccccccccccc

c landunit_s,landunit_sm1,oceanunit_s,oceanunit_sm1

      start4(1) = 1;  start4(2) = 1;  start4(3) = 1; start4(4) = 1
      count4(1) = im; count4(2) = jm; count4(3) = lreg;
      count4(4) = mtot

      status = NF_PUT_VARA_REAL(ncid,landunit_sid,start4,count4,
     ^                          landunit_s)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_VARA_REAL(ncid,oceanunit_sid,start4,count4,
     ^                          oceanunit_s)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_VARA_REAL(ncid,landunit_sm1id,start4,count4,
     ^                          landunit_sm1)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_VARA_REAL(ncid,oceanunit_sm1id,start4,count4,
     ^                          oceanunit_sm1)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccc

c landunit_reg1,landunit_reg2...., oceanunit_reg1,oceanunit_reg2.......

      count4(3) = pm

      Do r = 1, lreg
         status = NF_PUT_VARA_REAL(ncid,landunit_regid(r),start4,count4,
     ^                             landunit(:,:,:,r,:))
         if (status .ne. nf_noerr) call handle_err(status)

         status =NF_PUT_VARA_REAL(ncid,oceanunit_regid(r),start4,count4,
     ^                            oceanunit(:,:,:,r,:))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

ccccccccc time series (statCO2,statwind) ccccccccccccccccccccccccccc

      count3(1) = statcnt; count3(2) = tractot; count3(3) = tlen

      status = NF_PUT_VARA_REAL(ncid,statco2id,start3,count3,statco2)
      if (status .ne. nf_noerr) call handle_err(status)

      count3(1) = statcnt; count3(2) = windnum; count3(3) = tlen

      status = NF_PUT_VARA_REAL(ncid,statwindid,start3,count3,statwind)
      if (status .ne. nf_noerr) call handle_err(status)

cOOOOOOOOO end filling variables OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOO close file and end subroutine OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      status = NF_CLOSE(ncid)
```

```
      if (status .ne. nf_noerr) call handle_err(status)

      Return
      End Subroutine write_cdf_1


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c00000000000000000000000000000000000000000000000000000000000000000000000
c00000000000000000000000000000000000000000000000000000000000000000000000
c0                     Error-flagging subroutine                      00
c00000000000000000000000000000000000000000000000000000000000000000000000
c00000000000000000000000000000000000000000000000000000000000000000000000

      Subroutine Handle_err(status)


      Implicit none
      include 'netcdf.inc'
      integer                                  :: status

      If (status .ne. nf_noerr) then
        Write(*,*) NF_STRERROR(status)
        Stop
      End if

      Return
      End subroutine handle_err
```

For level II

```
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cO                                                                          OO
cO This program writes the TransCom 3 Level 2 output data to a series OO
cO of netCDF files. To search for what changes need to be made to      OO
cO this code search for the phrase: 'Transcom modelers'                OO
cO                                                                          OO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

cOOOOOOOOO Begin notes OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
C
C TRANSCOM MODELERS!
C
C PLEASE USE THE FOLLOWING CONVENTIONS WHEN FILLING LEVEL 2 ARRAYS:
C (all the array desctiptions are provided below)
C
C PLEASE WRITE THE GRIDCELL CENTERS ("LONVECT" AND "LATVECT" VECTORS)
C IN UNITS OF DEGREES EAST (ie. -175, -170.......0, 5.....170, 175) AND
C DEGREES NORTH (ie. -85, 80......0, 5......80, 85)
C
C PLEASE ENSURE THAT THE FIRST GRIDCELL IN 2D MAPS AND 3D FIELDS IS
C    - AT THE DATELINE RATHER THAN AT GREENWICH
C    - AT THE SOUTH POLE RATHER THAN THE NORTH POLE
C    - AT 1000 MB RATHER THAN 100 MB (FOR 3D FIELDS)
C
C THESE ARRAYS WILL THEN BE WRITTEN MOVING EAST (FOR LONGITUDINAL
C DIRECTION), MOVING NORTH (FOR LATITUDINAL DIRECTION) AND MOVING UP
C (FOR VERTICAL DIRECTION)
C
C PLEASE MAINTAIN THE BASIS FUNCTION REGION NUMBERING AS INDICATED IN
C THE PROTOCOL AND BASIS FUNCTION REGION MAP (ie. REGION 1 = NORTH
C AMERICAN BOREAL, 2 = NORTH AMERICAN TEMPERATE......)
C
C UNITS:
C    - CO2 CONCENTRATION VALUES AS VOLUMETRIC PARTS PER MILLION (PPMV)
C    - SF6 CONCENTRATION VALUES AS VOLUMETRIC PARTS PER TRILLION (PPTV)
C    - U AND V WINDS IN METERS PER SECOND
C    - OMEGA WIND IN PASCALS/SECOND
C
C MISSING VALUES ARE REPORTED AS 1.0 x 10^36
C
C LAND/SEA MASK ("LSMASK") AND TERRAIN MASK ("BETA") REPORT AS UNITLESS,
C FOLLOWING THE SAME GRIDDING RULES LISTED ABOVE.
C
C PLEASE NOTE THAT THERE ARE FOUR TIME-SERIES TO BE
C WRITTEN. THE FIRST REPRESENTS THE HIGH-FREQUENCY OUTPUT FOR THE
C PRE-SUBTRACTED FIELDS. THE SECOND IS FOR THE U AND V WIND, THE THIRD
C IS FOR THE TERRESTRIAL BASIS FUNCTIONS AND THE FOURTH IS FOR THE
C OCEANIC BASIS FUNCTIONS. PLEASE BEGIN ALL THESE TIME SERIES AT YOUR
C FIRST REPORTED TIMESTEP RATHER THAN ZERO.
C
C
C QUESTIONS? CONTACT KEVIN - keving@atmos.colostate.edu
C
C To compile on Dendrus (SGI O2K) at Colorado State University:
C
C f90 make.output.l2.f -o make.l2 -I/usr/local/netcdf/include
C -L/usr/local/lib -lnetcdf
C
COOOOOOOOO end notes OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
```

```
      Program makeout

cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin variable declarations OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Implicit none
      include 'netcdf.inc'

CMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELER
C
C TRANSCOM MODELERS: adjust the following integer parameters!!
C
      integer, parameter                          :: im=72,jm=44,pm=9,
     ^                                               timestep=6,
     ^                                               tlen1=(365+31)*
     ^                                               timestep,
     ^                                               tlen2=365*timestep*3
C
CMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELER

      integer                                     :: m,c,r,s,t,o,cnt
      integer, parameter                          :: mtot   =12,lreg=11,
     ^                                               elapm  =36,
     ^                                               statcnt=228,
     ^                                               presub =4,windnum=2

      real, dimension(:),           allocatable :: lonvect,latvect,
     ^                                               presvect,mvect,
     ^                                               lvect,evect,svect,
     ^                                               pvect,wvect,
     ^                                               t1vect,t2vect

      real, dimension(:,:),         allocatable :: lsmask,svects

      real, dimension(:,:,:),       allocatable :: ff90_s,  ff90_sm1,
     ^                                               ff95_s,  ff95_sm1,
     ^                                               bios_s,  bios_sm1,
     ^                                               ocean_s,ocean_sm1,
     ^                                               u_s,v_s,
     ^                                               statpco2,statwind

      real, dimension(:,:,:,:),     allocatable :: ff90,ff95,bios,
     ^                                               ocean,u,v,omega,
     ^                                               sf6_s,sf6_sm1,beta,
     ^                                               statlco2,statoco2

      real, dimension(:,:,:,:,:),   allocatable :: bfmap,sf6,
     ^                                               landunit,oceanunit

      character (len=8), dimension(:),allocatable:: monpulse
      character (len=1), dimension(:),allocatable:: it

cOOOOOOOOO end variable declarations OOOOOOOOOCOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin main program variable description OOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
C
c im          : The number of longitudinal grid cells reported.
c jm          : The number of latitudinal grid cells reported.
c pm          : The number of pressure levels reported.
c timestep    : The number of timesteps per day reported for the high
c               frequency data reporting.
c tlen1       : The total number of timesteps reported for a single
c               year of high frequency data (timestep*(365+31)).
c tlen2       : The total number of timesteps reported for three
c               years of high frequency data (tlen1*3).
c statcnt     : The number of CO2 observational stations: 228
c cnt         : A counter.
```

```
c mtot          : The number of months in a year: 12
c lreg          : The number of terrestrial basis function regions: 11
c windnum       : The number of wind directions reported in the high
c                 frequency data: 2
c elapm         : The number of months written for the CO2 tracers: 36
c presub        : The number of pre-subtracted tracers: 4
c monpulse      : A character array containing the month-pulse names.
c it            : A character array containing an iteration letter.
c lonvect       : A vector containing the coordinates of the reported
c                 longitudinal grid centers. First element starts near
c                 the dateline and moves East. Units: degrees east.
c latvect       : A vector containing the coordinates of the reported
c                 latitudinal grid centers. First element starts near the
c                 South Pole. Units: degrees north.
c presvect      : A vector containing the coordinates of the reported
c                 vertical grid centers in millibars. First element
c                 starts at 1000 mb.
c mvect         : A vector of the month index: 1 through 12
c lvect         : A vector of the region index: 1 through 11
c evect         : A vector of the elapsed month index: 1 through 36
c svect         : Vector of the full station index: 1-228
c svects        : An array containing the three subportions of the
c                 station index: 1-76, 77-152, 153-228
c t1vect        : A vector of the single year high frequency timestep
c                 index: 1 through 2376
c t2vect        : A vector of the three year high frequency timestep
c                 index: 1 through 2376
c pvect         : A vector of the pre-subtracted tracers index: 1 thru 4
c wvect         : A vector of the wind direction index: 1 through 2
c lsmask        : Land/sea mask. This is a 2D array in which both
c                 dimensions index space.
c ff90          : 1990 pre-subtracted fossil-fuel CO2 mixing ratio. This
c                 is a 4D array in which the first three dimensions index
c                 space and the last indexes time.
c ff95          : 1995 pre-subtracted fossil-fuel CO2 mixing ratio. This
c                 is a 4D array in which the first three dimensions index
c                 space and the last indexes time.
c bios          : Neutral biosphere pre-subtracted CO2 mixing ratio. This
c                 is a 4D array in which the first three dimensions index
c                 space and the last indexes time.
c ocean         : Ocean exchange pre-subtracted CO2 mixing ratio. This is
c                 a 4D array in which the first three dimensions index
c                 space and the last indexes time.
c ff90_s        : 1990 pre-subtracted fossil-fuel CO2 surface layer
c                 mixing ratio. This is a 3D array in which the first two
c                 dimensions index space and the last indexes time.
c ff90_sm1      : 1990 pre-subtracted fossil-fuel CO2 mixing ratio in
c                 layer above the surface layer. This is a 3D array in
c                 which the first two dimensions index space and the last
c                 indexes time.
c ff95_s        : 1995 pre-subtracted fossil-fuel CO2 surface layer
c                 mixing ratio. This is a 3D array in which the first two
c                 dimensions index space and the last indexes time.
c ff95_sm1      : 1995 pre-subtracted fossil-fuel CO2 mixing ratio in
c                 layer above the surface layer. This is a 3D array in
c                 which the first two dimensions index space and the last
c                 indexes time.
c bios_s        : Neutral biosphere pre-subtracted CO2 surface layer
c                 mixing ratio. This is a 3D array in which the first two
c                 dimensions index space and the last indexes time.
c bios_sm1      : Neutral biosphere pre-subtracted CO2 mixing ratio in
c                 layer above the surface layer. This is a 3D array in
c                 which the first two dimensions index space and the last
c                 indexes time.
c ocean_s       : Ocean exchange pre-subtracted CO2 surface layer mixing
c                 ratio. This is a 3D array in which the first two
c                 dimensions index space and the last indexes time.
c ocean_sm1     : Ocean exchange pre-subtracted CO2 mixing ratio in layer
c                 above the surface layer. This is a 3D array in which
c                 the first two dimensions index space and the last time.
c landunit      : Terrestrial carbon 3D basis function CO2 mixing ratio.
```

```
c                This is a 5D array in which the first three dimensions
c                index space, the fourth indexes the terrestrial basis
c                function region, and the fifth indexes time.
c oceanunit     : Oceanic carbon 3D basis function CO2 mixing ratio. This
c                is a 5D array in which the first three dimensions index
c                space, the fourth indexes the oceanic basis function
c                region, and the fifth indexes time.
c bfmap         : Basis function 2D concentration maps. A 5D array
c                in which the first two dimensions index space, the
c                third indexes the basis function region, the fourth
c                indexes the pulse-month, and the last indexes time.
c                There are four types: terrestrial surface, terrestrial
c                layer above surface, oceanic surface, oceanic layer
c                above surface.
c sf6           : SF6 mixing ratio. This is a 4D array in which the first
c                three dimensions index space, the fourth indexes the
c                basis function region and the last indexes time (since
c                this is not run as a pulsed emission, the final
c                dimension is 'mtot' instead of 'elapm').
c sf6_s         : Surface layer SF6 mixing ratio. This is a 4D array in
c                which the first two dimensions index space, the third
c                indexes the basis function region and the last time
c                (since this is not run as a pulsed emission, the final
c                dimension is 'mtot' instead of 'elapm').
c sf6_sml       : SF6 mixing ratio in layer above the surface layer. This
c                is a 4D array in which the first two dimensions index
c                space, the third indexes the basis function region and
c                the last indexes time (since this is not run as a
c                pulsed emission, the final dimension is 'mtot' instead
c                of 'elapm').
c u             : Longitudinal wind velocity. This is a 4D array in which
c                the first three dimensions index space and the last
c                indexes time.
c v             : Latitudinal wind velocity. This is a 4D array in which
c                the first three dimensions index space and the last
c                indexes time.
c omega         : vertical pressure velocity. This is a 4D array in which
c                the first three dimensions index space and the last
c                indexes time.
c u_s           : Longitudinal surface wind velocity. This is a 3D array
c                in which the first two dimensions index space and the
c                last indexes time.
c v_s           : Meridional surface wind velocity. This is a 3D array in
c                which the first two dimensions index space and the last
c                indexes time.
c statlco2      : A 4D array containing the CO2 single point reporting
c                output for the terrestrial basis function regions. The
c                first dimension indexes the station location, the
c                second indexes the basis function region, the third
c                indexes the pulse-month, and the last indexes time.
c statocc2      : A 4D array containing the CO2 single point reporting
c                output for the oceanic basis function regions. The
c                first dimension indexes the station location, the
c                second indexes the basis function region, the third
c                indexes the pulse-month, and the last indexes time.
c statpco2      : A 3D array containing the CO2 single point reporting
c                output for the pre-subtracted tracers. The first
c                dimension indexes the station location, the second
c                indexes the pre-subtraced tracer (order as ff90,ff95,
c                bios,ocean), and the last indexes time.
c statwind      : A 3D array containing the wind single point reporting
c                output. The first dimension indexes the station
c                location, the second indexes the reported wind
c                direction: u and v, and the last indexes time.
c beta          : Surface terrain mask. This is a 4D array in which the
c                first three dimensions index space and the last time.
c
c000000000 end main program variable description 0000000000000000000000

c0000000000000000000000000000000000000000000000c0000000000000000000000000000
```

```
cOOOOOOOOO begin interface module OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Interface

CMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELER
C
C TRANSCOM MODELERS!
C
C IF YOU ARE CALLING A SUBROUTINE TO FILL THE OUTPUT ARRAYS, SIX
C INTERFACE MODULES WILL BE REQUIRED HERE (due to the use of allocatable
C arrays).
C
CMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELERSMODELER

        Subroutine write_cdf_2_1(im,jm,pm,mtot,lreg,elapm,lonvect,
     ^                           latvect,presvect,mvect,lvect,evect,
     ^                           ff90,ff90_s,ff90_sm1,ff95,ff95_s,
     ^                           ff95_sm1,bios,bios_s,bios_sm1,sf6_s,
     ^                           sf6_sm1,sf6,ocean,ocean_s,ocean_sm1,
     ^                           u,u_s,v,v_s,omega,lsmask,beta)
          integer                                  :: im,jm,pm,lreg,mtot,
     ^                                               elapm

          real, dimension(:),         intent(in) :: lonvect,latvect,
     ^                                               presvect,mvect,
     ^                                               lvect,evect
          real, dimension(:,:),       intent(in) :: lsmask
          real, dimension(:,:,:),     intent(in) :: ff90_s,  ff95_sm1,
     ^                                               ff95_s,  ff90_sm1,
     ^                                               bios_s,  bios_sm1,
     ^                                               ocean_s,ocean_sm1,
     ^                                               u_s,v_s
          real, dimension(:,:,:,:),   intent(in) :: ff90,ff95,bios,
     ^                                               ocean,u,v,omega,
     ^                                               sf6_s,sf6_sm1,beta
          real, dimension(:,:,:,:,:), intent(in) :: sf6
        End Subroutine write_cdf_2_1


        Subroutine write_cdf_2_2(mtot,lreg,statcnt,tlen1,mvect,lvect,
     ^                           svect,t1vect,statlco2,it)
          integer                                  :: mtot,lreg,statcnt,
     ^                                               tlen1
          real, dimension(:),         intent(in) :: mvect,lvect,svect,
     ^                                               t1vect
          real, dimension(:,:,:,:),   intent(in) :: statlco2
          character(len=1)                         :: it
        End subroutine write_cdf_2_2


        Subroutine write_cdf_2_2d(statcnt,tlen2,presub,windnum,svect,
     ^                            t2vect,pvect,wvect,statpco2,statwind)
          integer                                  :: statcnt,tlen2,
     ^                                               presub,windnum
          real, dimension(:),         intent(in) :: svect,t2vect,pvect,
     ^                                               wvect
          real, dimension(:,:,:),     intent(in) :: statpco2,statwind
        End subroutine write_cdf_2_2d


        Subroutine write_cdf_2_3(mtot,lreg,statcnt,tlen1,mvect,lvect,
     ^                           svect,t1vect,statoco2,it)
          integer                                  :: mtot,lreg,statcnt,
     ^                                               tlen1
          real, dimension(:),         intent(in) :: mvect,lvect,svect,
     ^                                               t1vect
          real, dimension(:,:,:,:),   intent(in) :: statoco2
          character(len=1)                         :: it
        End subroutine write_cdf_2_3
```

```
      Subroutine write_cdf_2_4(im,jm,mtot,lreg,elapm,lonvect,latvect,
     ^                         mvect,evect,bfmap,it,cnt)
        integer                                  :: im,jm,cnt,
     ^                                              mtot,lreg,elapm
        real, dimension(:),         intent(in)   :: lonvect,latvect,
     ^                                              mvect,evect
        real, dimension(:,:,:,:,:), intent(in)   :: bfmap
        character(len=1)                         :: it
      End subroutine write_cdf_2_4


      Subroutine write_cdf_2_5(m,im,jm,pm,mtot,lreg,elapm,
     ^                         lonvect,latvect,presvect,mvect,lvect,
     ^                         evect,landunit,monpulse)
        integer                                  :: m,im,jm,pm,
     ^                                              mtot,lreg,
     ^                                              elapm
        real, dimension(:),         intent(in)   :: lonvect,latvect,
     ^                                              presvect,mvect,
     ^                                              lvect,evect
        real, dimension(:,:,:,:,:), intent(in)   :: landunit
        character (len=8), dimension(mtot)       :: monpulse
      End subroutine write_cdf_2_5


      Subroutine write_cdf_2_6(m,im,jm,pm,mtot,lreg,elapm,
     ^                         lonvect,latvect,presvect,mvect,lvect,
     ^                         evect,oceanunit,monpulse)
        integer                                  :: m,im,jm,pm,
     ^                                              mtot,lreg,
     ^                                              elapm
        real, dimension(:),         intent(in)   :: lonvect,latvect,
     ^                                              presvect,mvect,
     ^                                              lvect,evect
        real, dimension(:,:,:,:,:), intent(in)   :: oceanunit
        character (len=8), dimension(mtot)       :: monpulse
      End subroutine write_cdf_2_6

    End interface

c000000000 end interface module 0000000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin main program vector fill 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000

    Allocate(mvect (mtot   )); mvect  = (/(c,c=1,mtot)   /)
    Allocate(lvect (lreg   )); lvect  = (/(c,c=1,lreg)   /)
    Allocate(evect (elapm  )); evect  = (/(c,c=1,elapm)  /)

    Allocate(svect(statcnt))
    svect = (/(c,c=1,statcnt)/)

    Allocate(svects(3,statcnt/3))
    svects(1,:) = (/(c,c=1,statcnt/3)/)
    svects(2,:) = (/(c,c=statcnt/3+1,2*statcnt/3)/)
    svects(3,:) = (/(c,c=2*statcnt/3+1,statcnt)/)

    Allocate(it(4))
    it(1) = 'a'
    it(2) = 'b'
    it(3) = 'c'
    it(4) = 'd'

    Allocate(t1vect(tlen1  )); t1vect = (/(c,c=1,tlen1)  /)
    Allocate(t2vect(tlen2  )); t2vect = (/(c,c=1,tlen2)  /)
    Allocate(pvect (presub )); pvect  = (/(c,c=1,presub) /)
    Allocate(wvect (windnum)); wvect  = (/(c,c=1,windnum)/)
```

```
      Allocate(monpulse(mtot))

      monpulse(1)  = 'janpulse'; monpulse(2)  = 'febpulse'
      monpulse(3)  = 'marpulse'; monpulse(4)  = 'aprpulse'
      monpulse(5)  = 'maypulse'; monpulse(6)  = 'junpulse'
      monpulse(7)  = 'julpulse'; monpulse(8)  = 'augpulse'
      monpulse(9)  = 'seppulse'; monpulse(10) = 'octpulse'
      monpulse(11) = 'novpulse'; monpulse(12) = 'decpulse'

c000000000 end main program vector fill 0000000000000000000000000000000

cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
c000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin allocation and 2_1 array filling 00000000000000000000
c000000000000000000000000000000000000000000000000000000000000000000000000

      Allocate(lonvect   (im))
      Allocate(latvect   (jm))
      Allocate(presvect  (pm))
      Allocate(lsmask    (im,jm))
      Allocate(ff90_s    (im,jm,elapm))
      Allocate(ff90_sm1  (im,jm,elapm))
      Allocate(ff95_s    (im,jm,elapm))
      Allocate(ff95_sm1  (im,jm,elapm))
      Allocate(bios_s    (im,jm,elapm))
      Allocate(bios_sm1  (im,jm,elapm))
      Allocate(ocean_s   (im,jm,elapm))
      Allocate(ocean_sm1 (im,jm,elapm))
      Allocate(u_s       (im,jm,elapm))
      Allocate(v_s       (im,jm,elapm))
      Allocate(ff90      (im,jm,pm,elapm))
      Allocate(ff95      (im,jm,pm,elapm))
      Allocate(bios      (im,jm,pm,elapm))
      Allocate(ocean     (im,jm,pm,elapm))
      Allocate(u         (im,jm,pm,elapm))
      Allocate(v         (im,jm,pm,elapm))
      Allocate(omega     (im,jm,pm,elapm))
      Allocate(beta      (im,jm,pm,elapm))
      Allocate(sf6_s     (im,jm,lreg,mtot))
      Allocate(sf6_sm1   (im,jm,lreg,mtot))
      Allocate(sf6       (im,jm,pm,lreg,mtot))

C TRANSCOM MODELERS!
C
C FILL THE JUST-ALLOCATED ARRAYS HERE. THIS CAN BE DONE WITH EITHER A
C SUBROUTINE CALL (which requires code in the interface module above) OR
C PERFORMED DIRECTLY IN THIS PORTION OF THE PROGRAM'S MAIN BODY (no
C interface module required).
C
c000000000 end allocation and 2_1 array filling 0000000000000000000000

c000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin call to write_cdf_2_1 subroutine and deallocate 0000000
c000000000000000000000000000000000000000000000000000000000000000000000000

      Write(*,*) 'Calling the first netCDF writing subroutine'
      Write(*,*) ' '

      Call write_cdf_2_1(im,jm,pm,mtot,lreg,elapm,lonvect,
     ^                   latvect,presvect,mvect,lvect,evect,
     ^                   ff90,ff90_s,ff90_sm1,ff95,ff95_s,ff95_sm1,
     ^                   bios,bios_s,bios_sm1,sf6_s,sf6_sm1,sf6,
     ^                   ocean,ocean_s,ocean_sm1,u,u_s,v,v_s,omega,
     ^                   lsmask,beta)

      Write(*,*) 'output.L2.1.nc written and closed'
      Write(*,*) ' '

      Deallocate(ff90,ff90_s,ff90_sm1,ff95,ff95_s,ff95_sm1,bios,bios_s)
```

```
      Deallocate(bios_sm1,ocean,ocean_s,ocean_sm1,sf6_s,sf6_sm1,sf6)
      Deallocate(u,u_s,v,v_s,omega,lsmask,beta)

cOOOOOOOOO end call to write_cdf_2_1 subroutine and deallocate OOOOOOOOO


cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin allocation and 2_2 array filling OOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Do o = 1, 3
         Allocate(statlco2(statcnt/3,lreg,mtot,tlen1))

C TRANSCOM MODELERS!
C
C FILL THE THREE PORTIONS OF THE STATLCO2 ARRAY HERE; A THIRD EACH TIME
C THROUGH THIS O-LOOP. THEY ARE DIVIDED INTO EQUAL THIRDS BY THE
C STATION DIMENSION AS AND SHOULD BE FILLED IN THE FOLLOWING ORDER:
C
C 1) STATIONS 1-76,
C 2) STATIONS 77-152,
C 3) STATIONS 153-228.
C
C THIS ARRAY FILLING CAN BE DONE WITH EITHER A SUBROUTINE CALL (which
C requires code in the interface module above)OR PERFORMED DIRECTLY IN
C THIS PORTION OF THE PROGRAM'S MAIN BODY (no interface module
C required).
C
cOOOOOOOOO end allocation and array filling OOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin call to write_cdf_2_2 subroutine and deallocate OOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

         Write(*,*) 'Calling the netCDF writing subroutine 2_2'//it(o)
         Write(*,*) ' '

         Call write_cdf_2_2(mtot,lreg,statcnt/3,tlen1,mvect,lvect,
     ^                      svects(o,:),tlvect,statlco2,it(o))

         Write(*,*) 'output.L2.2'//it(o)//'.nc written and closed'
         Write(*,*) ' '
         Deallocate(statlco2)
      End do

cOOOOOOOOO end call to write_cdf_2_2 subroutine and deallocate OOOOOOOOO


cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin allocation and 2_2d array filling OOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Allocate(statpco2(statcnt,presub, tlen2))
      Allocate(statwind(statcnt,windnum,tlen2))

C TRANSCOM MODELERS!
C
C FILL THE JUST-ALLOCATED ARRAYS HERE. THIS CAN BE DONE WITH EITHER A
C SUBROUTINE CALL (which requires code in the interface module above) OR
C PERFORMED DIRECTLY IN THIS PORTION OF THE PROGRAM'S MAIN BODY (no
C interface module required).
C
cOOOOOOOOO end allocation and array filling OOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin call to write_cdf_2_2d subroutine and deallocate OOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
```

```
      Write(*,*) 'Calling the netCDF writing subroutine 2_2d'
      Write(*,*) ' '

      Call write_cdf_2_2d(statcnt,tlen2,presub,windnum,
     ^                    svect,t2vect,pvect,wvect,statpco2,statwind)

      Write(*,*) 'output.L2.2d.nc written and closed'
      Write(*,*) ' '

      Deallocate(statpco2,statwind)

c000000000 end call to write_cdf_2_2d subroutine and deallocate 00000000


cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
c0000000000000000000000000000C0000000000000000000000000000000000000000000000
c000000000 begin allocation and 2_3 array filling 000000000000000000000000
c0000000000000000000000000000C0000000000000000000000000000000000000000000000

      Do o = 1, 3
         Allocate(statoco2(statcnt/3,lreg,mtot,tlen1))

C TRANSCOM MODELERS!
C
C FILL THE THREE PORTIONS OF THE STATOCO2 ARRAY HERE; A THIRD EACH TIME
C THROUGH THIS O-LOOP. THEY ARE DIVIDED INTO EQUAL THIRDS BY THE
C STATION DIMENSION AS AND SHOULD BE FILLED IN THE FOLLOWING ORDER:
C
C 1) STATIONS 1-76,
C 2) STATIONS 77-152,
C 3) STATIONS 153-228
C
C THIS ARRAY FILLING CAN BE DONE WITH EITHER A SUBROUTINE CALL (which
C requires code in the interface module above)OR PERFORMED DIRECTLY IN
C THIS PORTION OF THE PROGRAM'S MAIN BODY (no interface module
C required).
C
c000000000 end allocation and 2_3 array filling 000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin call to write_cdf_2_3 subroutine and deallocate 0000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

         Write(*,*) 'Calling the netCDF writing subroutine 2_3'//it(o)
         Write(*,*) ' '

         Call write_cdf_2_3(mtot,lreg,statcnt/3,tlen1,mvect,lvect,
     ^                      svects(o,:),tlvect,statoco2,it(o))

         Write(*,*) 'output.L2.3'//it(o)//'.nc written and closed'
         Write(*,*) ' '
         Deallocate(statoco2)
      End do

c000000000 end call to write_cdf_2_3 subroutine and deallocate 000000000


cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin allocation and 2_4 array filling 000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      cnt = 0
      Do o = 1, 4
         Allocate(bfmap(im,jm,lreg,mtot,elapm))
         cnt = cnt + 1

C TRANSCOM MODELERS!
C
```

```
C FILL THE JUST-ALLOCATED ARRAY HERE. THIS IS DONE FOUR TIMES, ONCE EACH
C TIME THROUGH THIS O-LOOP. THIS ARRAY HOLDS THE BASIS FUNCTION MAP
C FIELDS AND SHOULD BE FILLED IN THE FOLLOWING ORDER:
C
C 1) TERRESTRIAL BASIS FUNCTION SURFACE CONCENTRATION,
C 2) TERRESTRIAL BASIS FUNCTION CONCENTRATION ABOVE THE SURFACE LAYER,
C 3) OCEANIC BASIS FUNCTION SURFACE CONCENTRATION,
C 4) OCEANIC BASIS FUNCTION CONCENTRATION ABOVE THE SURFACE LAYER.
C
C THIS ARRAY FILLING CAN BE DONE WITH EITHER A SUBROUTINE CALL (which
C requires code in the interface module above) OR PERFORMED DIRECTLY
C IN THIS PORTION OF THE PROGRAM'S MAIN BODY (no interface module
C required).
C
cOOOOOOOOOO end allocation and 2_4 array filling OOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOO begin call to write_cdf_2_4 subroutine and deallocate OOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

         Write(*,*) 'Calling the netCDF writing subroutine 2.4'//it(o)
         Write(*,*) ' '

         Call write_cdf_2_4(im,jm,mtot,lreg,elapm,lonvect,latvect,
     ^                      mvect,evect,bfmap,it(o),cnt)

         Write(*,*) 'output.L2.4'//it(o)//'.nc written'
         Write(*,*) ' '
         Deallocate(bfmap)
       End do

cOOOOOOOOO end call to write_cdf_2_4 subroutine and deallocate OOOOOOOOO


cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin allocation and 2_5 array filling OOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

       Allocate(landunit(im,jm,pm,lreg,elapm))

       Do m = 1, mtot

C TRANSCOM MODELERS!
C
C FILL THE JUST-ALLOCATED ARRAYS HERE. THIS CAN BE DONE WITH EITHER A
C SUBROUTINE CALL (which requires code in the interface module above) OR
C PERFORMED DIRECTLY IN THIS PORTION OF THE PROGRAM'S MAIN BODY (no
C interface module required). NOTE THAT THIS IS DONE WITHIN A LOOP
C THROUGH "MTOT" (THE VALUE IS 12). SO, THE LANDUNIT ARRAY IS FILLED
C ONCE FOR EVERY MONTH-PULSE.
C
cOOOOOOOOO end allocation and array 2_5 filling OOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin call to write_cdf_2_5 subroutine and deallocate OOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

         Write(*,*) 'Calling the fifth netCDF writing subroutine'
         Write(*,*) ' '

         Call write_cdf_2_5(m,im,jm,pm,mtot,lreg,elapm,
     ^                      lonvect,latvect,presvect,mvect,lvect,evect,
     ^                      landunit,monpulse)
         Write(*,*) 'output.L2.5_'//monpulse(m)//'.nc written, closed'
         Write(*,*) ' '
       End do
       Write(*,*) ' '
```

```
      Deallocate(landunit)

cOOOOOOOOO end call to write_cdf_5 subroutine and deallocate OOOOOOOOOO


cXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin allocation and 2_6 array filling OOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Allocate(oceanunit(im,jm,pm,lreg,elapm))

      Do m = 1, mtot

C TRANSCOM MODELERS!
C
C FILL THE JUST-ALLOCATED ARRAYS HERE. THIS CAN BE DONE WITH EITHER A
C SUBROUTINE CALL (which requires code in the interface module above) OR
C PERFORMED DIRECTLY IN THIS PORTION OF THE PROGRAM'S MAIN BODY (no
C interface module required). NOTE THAT THIS IS DONE WITHIN A LOOP
C THROUGH "MTOT" (THE VALUE IS 12). SO, THE LANDUNIT ARRAY IS FILLED
C ONCE FOR EVERY MONTH-PULSE.
C
cOOOOOOOOO end allocation and 2_6 array filling OOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin call to write_cdf_2_6 subroutine and deallocate OOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

         Write(*,*) 'Calling the sixth netCDF writing subroutine'
         Write(*,*) ' '


         Call write_cdf_2_6(m,im,jm,pm,mtot,lreg,elapm,
     ^                      lonvect,latvect,presvect,mvect,lvect,evect,
     ^                      oceanunit,monpulse)
         Write(*,*) 'output.L2.6_'//monpulse(m)//'.nc written, closed'
         Write(*,*) ' '
       End do
       Write(*,*) ' '

      Deallocate(oceanunit)

cOOOOOOOOO end call to write_cdf_6 subroutine and deallocate OOOOOOOOOO


      Stop
      End Program makeout

cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO



!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cO                     WRITE_CDF_2_1 SUBROUTINE                          OO
cO                                                                       OO
cO This subroutine writes the non-regional TransCom 3 level 2 output     OO
cO to a netcdf file called "output.L2.1.nc"                              OO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Subroutine write_cdf_2_1(im,jm,pm,mtot,lreg,elapm,lonvect,
     ^                         latvect,presvect,mvect,lvect,evect,
     ^                         ff90,ff90_s,ff90_sm1,ff95,ff95_s,ff95_sm1,
     ^                         bios,bios_s,bios_sm1,sf6_s,sf6_sm1,sf6,
```

```
     ^                        ocean,ocean_s,ocean_sml,u,u_s,v,v_s,omega,
     ^                        lsmask,beta)

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable declarations 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      implicit none
      include 'netcdf.inc'

      integer                                  :: im,jm,pm,mtot,lreg,
     ^                                            elapm,r

      real, parameter                          :: misval=1.0e+36

      real, dimension(:),           intent(in) :: lonvect,latvect,
     ^                                            presvect,mvect,lvect,
     ^                                            evect

      real, dimension(:,:),         intent(in) :: lsmask

      real, dimension(:,:,:),       intent(in) :: ff90_s,  ff90_sml,
     ^                                            ff95_s,  ff95_sml,
     ^                                            bios_s,  bios_sml,
     ^                                            ocean_s,ocean_sml,
     ^                                            u_s,v_s

      real, dimension(:,:,:,:),     intent(in) :: ff90,ff95,bios,ocean,
     ^                                            u,v,omega,
     ^                                            sf6_s,sf6_sml,beta

      real, dimension(:,:,:,:,:),   intent(in) :: sf6

      character (len=80)                       :: regn
      character (len=2), dimension(lreg)       :: num
      character (len=10), dimension(lreg)      :: sf6name

ccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccc

      integer                                  :: status,ncid,imid,
     ^                                            jmid,pmid,mtotid,
     ^                                            lregid,elapmid,lonid,
     ^                                            latid,presid,mid,lid,
     ^                                            tid,ff90id,ff90_sid,
     ^                                            ff90_smlid,ff95id,
     ^                                            ff95_sid,ff95_smlid,
     ^                                            biosid,bios_sid,
     ^                                            bios_smlid,oceanid,
     ^                                            ocean_sid,
     ^                                            ocean_smlid,sf6_sid,
     ^                                            sf6_smlid,
     ^                                            sf6_regid(lreg),
     ^                                            uid,vid,
     ^                                            u_sid,v_sid,omegaid,
     ^                                            lsmaskid,betaid,
     ^                                            vdims2(2),vdims3(3),
     ^                                            vdims4(4),
     ^                                            start2(2),count2(2),
     ^                                            start3(3),count3(3),
     ^                                            start4(4),count4(4)

c000000000 end variable declarations 0000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable description 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c
ccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccc
c
c misval        : The designated TransCom missing value: 1.0e+36.
```

```
c r             : An index for looping over the terrestrial regions.
c sf6name       : A character array containing the variable names for
c                 the series of netCDF arrays associated with the SF6
c                 basis function region 3D concentrations.
c regn          : A character variable used for the 'long_name' netCDF
c                 attribute.
c num           : A character array containing the basis function region
c                 number: 01 through 11.
c
ccccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccccccc
c
c ncid          : The netCDF ID representing the open file.
c XXXXXid       : A series of ID's that correspond to those variables
c                 names passed into the write_cdf subroutine. There are
c                 three exceptions to this scheme. They are:
c      sf6_reg#      : These ID's reduce the incoming sf6 5D array
c                      into 4D arrays. There are 11 of these ID's, one
c                      one for each terrestrial basis function region.
c vdims#        : These are netCDF integer vectors whose elements
c                 denote the dimensions of the netCDF arrays.
c start#        : These are netCDF integer vectors whose elements denote
c                 where in the netCDF array the first element of a given
c                 write statement should start.
c count#        : These are netCDF integer vectors whose elements denote
c                 the edge lengths along each dimension of a given write
c                 statement.
c
c000000000 end variable description 0000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable initialization 00000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      sf6name(1)  = 'sf6_reg_01'; num(1)  = '01'
      sf6name(2)  = 'sf6_reg_02'; num(2)  = '02'
      sf6name(3)  = 'sf6_reg_03'; num(3)  = '03'
      sf6name(4)  = 'sf6_reg_04'; num(4)  = '04'
      sf6name(5)  = 'sf6_reg_05'; num(5)  = '05'
      sf6name(6)  = 'sf6_reg_06'; num(6)  = '06'
      sf6name(7)  = 'sf6_reg_07'; num(7)  = '07'
      sf6name(8)  = 'sf6_reg_08'; num(8)  = '08'
      sf6name(9)  = 'sf6_reg_09'; num(9)  = '09'
      sf6name(10) = 'sf6_reg_10'; num(10) = '10'
      sf6name(11) = 'sf6_reg_11'; num(11) = '11'

c000000000 end variable initialization 00000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 open netCDF file and define the dimensions 0000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CREATE('output.L2.1.nc',0,ncid)
      write(*,*) 'output.L2.1.nc created'
      write(*,*) ' '

      status = NF_DEF_DIM(ncid,'longitude',im,imid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'latitude',jm,jmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'height',pm,pmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'month',mtot,mtotid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'land_region',lreg,lregid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'time',elapm,elapmid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end open of netCDF file and dimension definition 000000000000
```

```
c000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 000000000000000000000000000000000000000000
c000000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,pressure,time,regions) ccccccccccc

      status = NF_DEF_VAR(ncid,'longitude',NF_FLOAT,1,imid,lonid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'latitude',NF_FLOAT,1,jmid,latid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'height',NF_FLOAT,1,pmid,presid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'month',NF_FLOAT,1,mtotid,mid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'land_region',NF_FLOAT,1,lregid,lid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'time',NF_FLOAT,1,elapmid,tid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc land/ocean mask (lsmask) ccccccccccccccccccccccccccccccccccc

      vdims2(1) = imid; vdims2(2) = jmid

      status = NF_DEF_VAR(ncid,'lsmask',NF_FLOAT,2,vdims2,lsmaskid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted carbon surface maps ccccccccccccccccccccccccc

c ff90_s,ff90_sm1,ff95_s,ff95_sm1,bios_s,bios_sm1,ocean_s,ocean_sm1

      vdims3(1) = imid; vdims3(2) = jmid; vdims3(3) = elapmid

      status = NF_DEF_VAR(ncid,'ff90_s',NF_FLOAT,3,vdims3,ff90_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff90_sm1',NF_FLOAT,3,vdims3,ff90_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff95_s',NF_FLOAT,3,vdims3,ff95_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff95_sm1',NF_FLOAT,3,vdims3,ff95_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'bios_s',NF_FLOAT,3,vdims3,bios_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'bios_sm1',NF_FLOAT,3,vdims3,bios_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ocean_s',NF_FLOAT,3,vdims3,ocean_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status =NF_DEF_VAR(ncid,'ocean_sm1',NF_FLOAT,3,vdims3,ocean_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc surface wind maps (u_s,v_s) ccccccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'u_s',NF_FLOAT,3,vdims3,u_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'v_s',NF_FLOAT,3,vdims3,v_sid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted 3D carbon fields (ff90,ff95,bios,ocean) cccccc

      vdims4(1)=imid; vdims4(2)=jmid; vdims4(3)=pmid; vdims4(4)=elapmid

      status = NF_DEF_VAR(ncid,'ff90',NF_FLOAT,4,vdims4,ff90id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ff95',NF_FLOAT,4,vdims4,ff95id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'bios',NF_FLOAT,4,vdims4,biosid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'ocean',NF_FLOAT,4,vdims4,oceanid)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
ccccccccc 3D wind fields (u,v,omega) ccccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'u',NF_FLOAT,4,vdims4,uid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'v',NF_FLOAT,4,vdims4,vid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'omega',NF_FLOAT,4,vdims4,omegaid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc terrain mask (beta) ccccccccccccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'beta',NF_FLOAT,4,vdims4,betaid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D SF6 fields (sf6_reg1,sf6_reg2.....) ccccccccccccccccccc

      vdims4(1)=imid; vdims4(2)=jmid; vdims4(3)=pmid; vdims4(4)=mtotid

      Do r = 1, lreg
         status = NF_DEF_VAR(ncid,sf6name(r),NF_FLOAT,4,vdims4,
     ^                       sf6_regid(r))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

ccccccccc SF6 surface maps (sf6_s,sf6_sm1) ccccccccccccccccccccccccc

      vdims4(1)=imid; vdims4(2)=jmid; vdims4(3)=lregid; vdims4(4)=mtotid

      status = NF_DEF_VAR(ncid,'sf6_s',NF_FLOAT,4,vdims4,sf6_sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'sf6_sm1',NF_FLOAT,4,vdims4,sf6_sm1id)
      if (status .ne. nf_noerr) call handle_err(status)

cOOOOOOOOO end define variables OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin apply attributes OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

ccccccccc vectors (longitude,latitude,pressure,time,regions) ccccccccc

      status = NF_PUT_ATT_TEXT(ncid,lonid,'long_name',29,
     ^                         'Longitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lonid,'units',12,'degrees_east')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,latid,'long_name',28,
     ^                         'Latitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,latid,'units',13,'degrees_north')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,presid,'long_name',25,
     ^                         'Vertical gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,presid,'units',9,'millibars')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,mid,'long_name',5,'month')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,mid,'units',5,'month')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,lid,'long_name',13,'region number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lid,'units',11,'land region')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,tid,'long_name',13,'elapsed month')
```

```
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,tid,'units',13,'elapsed month')
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc land/ocean mask (lsmask) ccccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,lsmaskid,'long_name',13,
     ^                         'Land/sea mask')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lsmaskid,'units',8,'unitless')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,lsmaskid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc pre-subtracted carbon surface maps ccccccccccccccccccccccc

c ff90_s,ff90_sml,ff95_s,ff95_sml,bios_s,bios_sml,ocean_s,ocean_sml

      status = NF_PUT_ATT_TEXT(ncid,ff90_sid,'long_name',47,
     ^               'Pre-subtracted 1990 fossil fuel CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff90_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff90_sid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff95_sid,'long_name',47,
     ^              'Pre-subtracted 1995 fossil fuel CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff95_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff95_sid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,bios_sid,'long_name',40,
     ^                        'Pre-subtracted biosphere CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,bios_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,bios_sid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ocean_sid,'long_name',38,
     ^                         'Pre-subtracted oceanic CO2 surface map')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ocean_sid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ocean_sid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff90_smlid,'long_name',60,
     ^   'Pre-subtracted 1990 fossil fuel CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff90_smlid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff90_smlid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff95_smlid,'long_name',60,
     ^   'Pre-subtracted 1995 fossil fuel CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff95_smlid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff95_smlid,'missing_value',
     ^                         NF_FLOAT,1,misval)
```

```
      if (status .ne. nf_ncerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,bios_sm1id,'long_name',53,
     ^         'Pre-subtracted biosphere CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,bios_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,bios_sm1id,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ocean_sm1id,'long_name',51,
     ^             'Pre-subtracted oceanic CO2 map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ocean_sm1id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status =NF_PUT_ATT_REAL(ncid,ocean_sm1id,'missing_value',
     ^                        NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc surface wind maps (u_s,v_s)ccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,u_sid,'long_name',34,
     ^                         'Longitudinal surface wind velocity')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,u_sid,'units',17,
     ^                         'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,u_sid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,v_sid,'long_name',32,
     ^                         'Meridional surface wind velocity')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,v_sid,'units',17,
     ^                         'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,v_sid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

cccccccccc pre-subtracted 3D carbon fields (ff90,ff95,bios,ocean) cccccc

      status = NF_PUT_ATT_TEXT(ncid,ff90id,'long_name',44,
     ^                  'Pre-subtracted 1990 fossil fuel CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff90id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff90id,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,ff95id,'long_name',44,
     ^                  'Pre-subtracted 1995 fossil fuel CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,ff95id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,ff95id,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,biosid,'long_name',37,
     ^                         'Pre-subtracted biosphere CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,biosid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,biosid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
      status = NF_PUT_ATT_TEXT(ncid,oceanid,'long_name',35,
     ^                         'Pre-subtracted oceanic CO2 3D field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,oceanid,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,oceanid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D wind fields (u,v,omega) ccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,uid,'long_name',26,
     ^                         'Longitudinal 3D wind field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,uid,'units',17,
     ^                         'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,uid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,vid,'long_name',24,
     ^                         'Meridional 3D wind field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,vid,'units',17,
     ^                         'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,vid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,omegaid,'long_name',22,
     ^                         'Vertical 3D wind field')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,omegaid,'units',18,
     ^                         'pascals per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,omegaid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc terrain mask (beta) ccccccccccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,betaid,'long_name',20,
     ^                         'Surface terrain mask')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,betaid,'units',8,'unitless')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,betaid,'missing_value',NF_FLOAT,1,
     ^                         misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D SF6 fields (sf6_reg1,sf6_reg2,sf6_reg3.......) ccccccccccc

      Do r = 1, lreg
         regn = 'SF6 basis function 3D concentration field: terrestrial
     ^region '//num(r)
         status = NF_PUT_ATT_TEXT(ncid,sf6_regid(r),'long_name',64,regn)
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_TEXT(ncid,sf6_regid(r),'units',4,'pptv')
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_REAL(ncid,sf6_regid(r),'missing_value',
     ^                            NF_FLOAT,1,misval)
         if (status .ne. nf_noerr) call handle_err(status)
      End do

ccccccccc SF6 surface maps (sf6_s,sf6_sml) ccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,sf6_sid,'long_name',30,
     ^                         'SF6 concentration map: surface')
      if (status .ne. nf_noerr) call handle_err(status)
```

```
      status = NF_PUT_ATT_TEXT(ncid,sf6_sid,'units',4,'pptv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,sf6_sid,'missing_value',NF_FLOAT,
     ^                         1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,sf6_sm1id,'long_name',42,
     ^                   'SF6 concentration map: layer above surface')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,sf6_sm1id,'units',4,'pptv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,sf6_sm1id,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccc

      status = NF_ENDDEF(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end apply attributes 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin filling variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccc vectors (longitude,latitude,pressure,time,regions) ccccccccc

      status = NF_PUT_VAR_REAL(ncid,lonid,lonvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,latid,latvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,presid,presvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,mid,mvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,lid,lvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,tid,evect)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc land/ocean mask (lsmask) ccccccccccccccccccccccccccccccccc

      status = NF_PUT_VAR_REAL(ncid,lsmaskid,lsmask)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc pre-subtracted carbon surface maps ccccccccccccccccccccccc

c ff90_s,ff90_sm1,ff95_s,ff95_sm1,bios_s,bios_sm1,ocean_s,ocean_sm1

      start3(1) = 1;  start3(2) = 1;  start3(3) = 1
      count3(1) = im; count3(2) = jm; count3(3) = elapm

      status = NF_PUT_VARA_REAL(ncid,ff90_sid,start3,count3,ff90_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff95_sid,start3,count3,ff95_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,bios_sid,start3,count3,bios_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ocean_sid,start3,count3,ocean_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff90_sm1id,start3,count3,ff90_sm1)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff95_sm1id,start3,count3,ff95_sm1)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,bios_sm1id,start3,count3,bios_sm1)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ocean_sm1id,start3,count3,
     ^                          ocean_sm1)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
ccccccccccc surface wind maps (u_s,v_s)ccccccccccccccccccccccccccccccccc

      status = NF_PUT_VARA_REAL(ncid,u_sid,start3,count3,u_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,v_sid,start3,count3,v_s)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc pre-subtracted 3D carbon fields (ff90,ff95,bios,ocean) cccccc

      start4(1) = 1; start4(2) = 1; start4(3) = 1; start4(4) = 1
      count4(1) = im; count4(2) = jm; count4(3) = pm; count4(4) = elapm

      status = NF_PUT_VARA_REAL(ncid,ff90id,start4,count4,ff90)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,ff95id,start4,count4,ff95)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,biosid,start4,count4,bios)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,oceanid,start4,count4,ocean)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D wind fields (u,v,omega) ccccccccccccccccccccccccccccccccc

      status = NF_PUT_VARA_REAL(ncid,uid,start4,count4,u)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,vid,start4,count4,v)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,omegaid,start4,count4,omega)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc terrain mask (beta) ccccccccccccccccccccccccccccccccccccccccc

      status = NF_PUT_VARA_REAL(ncid,betaid,start4,count4,beta)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D SF6 fields (sf6_reg1,sf6_reg2,sf6_reg3.......) ccccccccccc

      count4(1) = im; count4(2) = jm; count4(3) = pm; count4(4) = mtot

      Do r = 1, lreg
         status=NF_PUT_VARA_REAL(ncid,sf6_regid(r),start4,count4,
     ^                           sf6(:,:,:,r,:))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

ccccccccccc SF6 surface maps (sf6_s,sf6_sm1) ccccccccccccccccccccccccccc

      count4(3) = lreg

      status = NF_PUT_VARA_REAL(ncid,sf6_sid,start4,count4,sf6_s)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VARA_REAL(ncid,sf6_sm1id,start4,count4,sf6_sm1)
      if (status .ne. nf_noerr) call handle_err(status)

cOOOOOOOOO end filling variables OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOO close file and end subroutine OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      status = NF_CLOSE(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

      Return
      End Subroutine write_cdf_2_1


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cO                      WRITE_CDF_2_2X SUBROUTINE                              OO
cO                                                                             OO
cO This subroutine writes TransCom 3 level 2 output to a netCDF file           OO
cO called "output.L2.2X.nc"                                                    OO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      Subroutine write_cdf_2_2(mtot,lreg,statcnt,tlen1,
     ^                         mvect,lvect,svect,tlvect,statlco2,it)

cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin variable declarations OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

      implicit none
      include 'netcdf.inc'

      integer                                      :: mtot,lreg,statcnt,
     ^                                                tlen1
      real, parameter                              :: misval=1.0e+36
      real, dimension(:),             intent(in)   :: mvect,lvect,svect,
     ^                                                tlvect
      real, dimension(:,:,:,:),       intent(in)   :: statlco2
      character(len=1)                             :: it

ccccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccc

      integer                                      :: status,ncid,mtotid,
     ^                                                lregid,statcntid,
     ^                                                tlenlid,
     ^                                                mid,lid,sid,tlid,
     ^                                                statlco2id,
     ^                                                vdims3(3),vdims4(4),
     ^                                                start3(3),count3(3),
     ^                                                start4(4),count4(4)

cOOOOOOOOO end variable declarations OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin variable description OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
c
ccccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccccccccc
c
c misval        : The designated TransCom missing value: 1.0e+36.
c
ccccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccccccccc
c
c ncid          : The netCDF ID representing the open file.
c XXXXXid       : A series of ID's that correspond to those variables
c                 names passed into the write_cdf subroutine. There are
c                 three exceptions to this scheme. They are:
c vdims#        : These are netCDF integer vectors whose elements
c                 denote the dimensions of the netCDF arrays.
c start#        : These are netCDF integer vectors whose elements denote
c                 where in the netCDF array the first element of a given
c                 write statement should start.
c count#        : These are netCDF integer vectors whose elements denote
c                 the edge lengths along each dimension of a given write
c                 statement.
c
cOOOOOOOOO end variable description OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO open netCDF file and define the dimensions OOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
```

```
      status = NF_CREATE('output.L2.2'//it//'.nc',0,ncid)
      write(*,*) 'output.L2.2'//it//'.nc created'

      status = NF_DEF_DIM(ncid,'month',mtot,mtotid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'land_region',lreg,lregid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'station_count',statcnt,statcntid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'tlen1',tlen1,tlen1id)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end open of netCDF file and dimension definition 000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'month',NF_FLOAT,1,mtotid,mid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'land_region',NF_FLOAT,1,lregid,lid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'station_count',NF_FLOAT,1,statcntid,sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'tlen1',NF_FLOAT,1,tlen1id,t1id)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc time series (statlco2X) ccccccccccccccccccccccccccccccccccccc

      vdims4(1) = statcntid; vdims4(2) = lregid; vdims4(3) = mtotid
      vdims4(4) = tlen1id

      status =NF_DEF_VAR(ncid,'stat_L_CO2'//it,NF_FLOAT,4,vdims4,
     ^                   statlco2id)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end define variables 0000000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin apply attributes 000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,mid,'long_name',5,'month')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,mid,'units',11,'pulse-month')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,lid,'long_name',13,'region number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lid,'units',11,'land region')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,sid,'long_name',14,'station number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,sid,'units',7,'station')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,t1id,'long_name',34,
     ^                         'elapsed timesteps in one year span')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,t1id,'units',8,'timestep')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc time series (statlco2X) ccccccccccccccccccccccccccccccccccccc
```

```
      status = NF_PUT_ATT_TEXT(ncid,statlco2id,'long_name',58,
     ^    'High frequency land basis function station CO2 time series')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,statlco2id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,statlco2id,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccccc

      status = NF_ENDDEF(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end apply attributes 0000000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin filling variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_PUT_VAR_REAL(ncid,mid,mvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,lid,lvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,sid,svect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,tlid,tlvect)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc time series (statlco2X) ccccccccccccccccccccccccccccccccccccc

      start4(1) = 1; start4(2) = 1; start4(3) = 1; start4(4) = 1
      count4(1)=statcnt;count4(2)=lreg;count4(3)=mtot;count4(4)=tlen1

      status = NF_PUT_VARA_REAL(ncid,statlco2id,start4,count4,statlco2)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end filling variables 0000000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c00000000 close file and end subroutine 000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CLOSE(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

      Return
      End Subroutine write_cdf_2_2


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0                     WRITE_CDF_2_2d SUBROUTINE                          00
c0                                                                        00
c0 This subroutine writes TransCom 3 level 2 output to a netCDF file      00
c0 called "output.L2.2d.nc"                                               00
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      Subroutine write_cdf_2_2d(statcnt,tlen2,presub,windnum,svect,
     ^                          t2vect,pvect,wvect,statpco2,statwind)


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable declarations 0000000000000000000000000000000000
```

```
c0000000000000000000000000000000000000000000000000000000000000000000000000000000

      implicit none
      include 'netcdf.inc'

      integer                                    :: statcnt,tlen2,
     ^                                              presub,windnum
      real, parameter                            :: misval=1.0e+36
      real, dimension(:),             intent(in) :: svect,t2vect,pvect,
     ^                                              wvect
      real, dimension(:,:,:),         intent(in) :: statpco2,statwind

ccccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccccc

      integer                                    :: status,ncid,
     ^                                              statcntid,
     ^                                              tlen2id,
     ^                                              presubid,windnumid,
     ^                                              sid,
     ^                                              t2id,pid,wid,
     ^                                              statpco2id,
     ^                                              statwindid,
     ^                                              vdims3(3),vdims4(4),
     ^                                              start3(3),count3(3),
     ^                                              start4(4),count4(4)

c000000000 end variable declarations 0000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable description 0000C000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c
ccccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccccccc
c
c misval        : The designated TransCom missing value: 1.0e+36.
c
ccccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccccccc
c
c ncid          : The netCDF ID representing the open file.
c XXXXXid       : A series of ID's that correspond to those variables
c                 names passed into the write_cdf subroutine. There are
c                 three exceptions to this scheme. They are:
c vdims#        : These are netCDF integer vectors whose elements
c                 denote the dimensions of the netCDF arrays.
c start#        : These are netCDF integer vectors whose elements denote
c                 where in the netCDF array the first element of a given
c                 write statement should start.
c count#        : These are netCDF integer vectors whose elements denote
c                 the edge lengths along each dimension of a given write
c                 statement.
c
c000000000 end variable description 0000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 open netCDF file and define the dimensions 000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CREATE('output.L2.2d.nc',0,ncid)
      write(*,*) 'output.L2.2d.nc created'

      status = NF_DEF_DIM(ncid,'station_count',statcnt,statcntid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'tlen2',tlen2,tlen2id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'presub_number',presub,presubid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'wind_dir',windnum,windnumid)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
c000000000 end open of netCDF file and dimension definition 000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'station_count',NF_FLOAT,1,statcntid,sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'tlen2',NF_FLOAT,1,tlen2id,t2id)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'presub_number',NF_FLOAT,1,presubid,pid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'wind_dir',NF_FLOAT,1,windnumid,wid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc time series (statpco2,statwind) ccccccccccccccccccccccccccc

      vdims3(1) = statcntid; vdims3(2) = presubid; vdims3(3) = tlen2id

      status =NF_DEF_VAR(ncid,'stat_p_CO2',NF_FLOAT,3,vdims3,statpco2id)
      if (status .ne. nf_noerr) call handle_err(status)

      vdims3(1) = statcntid; vdims3(2) = windnumid; vdims3(3) = tlen2id

      status = NF_DEF_VAR(ncid,'stat_wind',NF_FLOAT,3,vdims3,statwindid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end define variables 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000c begin apply attributes 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,sid,'long_name',14,'station number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,sid,'units',7,'station')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,t2id,'long_name',35,
     ^                         'elapsed timesteps in four year span')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,t2id,'units',8,'timestep')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,pid,'long_name',13,'tracer number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,pid,'units',14,'pre-sub tracer')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,wid,'long_name',21,
     ^                         'wind direction number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,wid,'units',14,'wind direction')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc time series (statpco2,statwind) ccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,statpco2id,'long_name',53,
     ^      'High frequency pre-subtracted station CO2 time series')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,statpco2id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,statpco2id,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
      status = NF_PUT_ATT_TEXT(ncid,statwindid,'long_name',39,
     ^                 'High frequency station wind time series')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,statwindid,'units',17,
     ^                         'meters per second')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,statwindid,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccc

      status = NF_ENDDEF(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end apply attributes 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin filling variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_PUT_VAR_REAL(ncid,sid,svect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,t2id,t2vect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,pid,pvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,wid,wvect)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc time series (statpco2,statwind) ccccccccccccccccccccccccccc

      start3(1) = 1; start3(2) = 1; start3(3) = 1
      count3(1) = statcnt; count3(2) = presub; count3(3) = tlen2

      status = NF_PUT_VARA_REAL(ncid,statpco2id,start3,count3,statpco2)
      if (status .ne. nf_noerr) call handle_err(status)

      count3(1) = statcnt; count3(2) = windnum; count3(3) = tlen2

      status = NF_PUT_VARA_REAL(ncid,statwindid,start3,count3,statwind)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end filling variables 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c00000000 close file and end subroutine 000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CLOSE(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

      Return
      End Subroutine write_cdf_2_2d


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
cO                     WRITE_CDF_2_3X SUBROUTINE                          OO
cO                                                                        OO
cO This subroutine writes TransCom 3 level 2 output to a netCDF file      OO
cO called "output.L2.3X.nc"                                               OO
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
```

```
      Subroutine write_cdf_2_3(mtot,lreg,statcnt,tlen1,
     ^                         mvect,lvect,svect,tlvect,statoco2,it)

c0000000000000000000000000000000c0000000000000000000000000000000000000000000000000
c000000000 begin variable declarations 0000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000000

      implicit none
      include 'netcdf.inc'

      integer                                    :: mtot,lreg,statcnt,
     ^                                              tlen1
      real, parameter                            :: misval=1.0e+36
      real, dimension(:),           intent(in)   :: mvect,lvect,svect,
     ^                                              tlvect
      real, dimension(:,:,:,:),     intent(in)   :: statoco2
      character(len=1)                           :: it

ccccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccc

      integer                                    :: status,ncid,mtotid,
     ^                                              lregid,statcntid,
     ^                                              tlen1id,
     ^                                              mid,lid,sid,tlid,
     ^                                              statoco2id,
     ^                                              vdims4(4),
     ^                                              start4(4),count4(4)

c000000000 end variable declarations 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable description 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c
ccccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccccccccc
c
c misval        : The designated TransCom missing value: 1.0e+36.
c
ccccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccccccccc
c
c ncid          : The netCDF ID representing the open file.
c XXXXXid       : A series of ID's that correspond to those variables
c                 names passed into the write_cdf subroutine. There are
c                 three exceptions to this scheme. They are:
c vdims#        : These are netCDF integer vectors whose elements
c                 denote the dimensions of the netCDF arrays.
c start#        : These are netCDF integer vectors whose elements denote
c                 where in the netCDF array the first element of a given
c                 write statement should start.
c count#        : These are netCDF integer vectors whose elements denote
c                 the edge lengths along each dimension of a given write
c                 statement.
c
c000000000 end variable description 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 open netCDF file and define the dimensions 00000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CREATE('output.L2.3'//it//'.nc',0,ncid)
      write(*,*) 'output.L2.3'//it//'.nc created'

      status = NF_DEF_DIM(ncid,'month',mtot,mtotid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'land_region',lreg,lregid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'station_count',statcnt,statcntid)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
      status = NF_DEF_DIM(ncid,'tlen1',tlen1,tlenlid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end open of netCDF file and dimension definition 000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'month',NF_FLOAT,1,mtotid,mid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'land_regicn',NF_FLOAT,1,lregid,lid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'station_count',NF_FLOAT,1,statcntid,sid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'tlen1',NF_FLOAT,1,tlenlid,tlid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc time series (statoco2) ccccccccccccccccccccccccccccccccccccccc

      vdims4(1) = statcntid; vdims4(2) = lregid; vdims4(3) = mtotid
      vdims4(4) = tlenlid

      status =NF_DEF_VAR(ncid,'stat_O_CO2'//it,NF_FLOAT,4,vdims4,
     ^                   statoco2id)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end define variables 000000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin apply attributes 000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000C000000

ccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,mid,'long_name',5,'month')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,mid,'units',11,'pulse-month')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,lid,'long_name',13,'region number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lid,'units',12,'ocean region')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,sid,'long_name',14,'station number')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,sid,'units',7,'station')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,tlid,'long_name',34,
     ^                         'elapsed timesteps in one year span')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,tlid,'units',8,'timestep')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc time series (statoco2X) ccccccccccccccccccccccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,statoco2id,'long_name',61,
     ^  'High frequency oceanic basis function station CO2 time series')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,statoco2id,'units',4,'ppmv')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_REAL(ncid,statoco2id,'missing_value',
     ^                         NF_FLOAT,1,misval)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
ccccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccc

      status = NF_ENDDEF(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end apply attributes 0000000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin filling variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (regions,time,numbers) ccccccccccccccccccccccccccccc

      status = NF_PUT_VAR_REAL(ncid,mid,mvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,lid,lvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,sid,svect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,tlid,tlvect)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc time series (statoco2X) ccccccccccccccccccccccccccccccccccccc

      start4(1) = 1; start4(2) = 1; start4(3) = 1; start4(4) = 1
      count4(1)=statcnt;count4(2)=lreg;count4(3)=mtot;count4(4)=tlen1

      status = NF_PUT_VARA_REAL(ncid,statoco2id,start4,count4,
     ^                          statoco2)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end filling variables 0000000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c00000000 close file and end subroutine 000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CLOSE(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

      Return
      End Subroutine write_cdf_2_3


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
cO                     WRITE_CDF_2_4X SUBROUTINE                          OO
cO                                                                        OO
cO This subroutine writes TransCom 3 level 2 output to a netCDF file      OO
cO called "output.L2.4X.nc"                                               OO
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      Subroutine write_cdf_2_4(im,jm,mtot,lreg,elapm,lonvect,latvect,
     ^                         mvect,evect,bfmap,it,cnt)

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable declarations 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      implicit none
      include 'netcdf.inc'

      integer                                  :: im,jm,r,cnt,
     ^                                            mtot,lreg,elapm
      real, parameter                          :: misval=1.0e+36
      real, dimension(:),           intent(in) :: lonvect,latvect,
```

```
     ^                                                  mvect,evect
      real, dimension(:,:,:,:,:),     intent(in) :: bfmap
      character (len=80)                         :: regn
      character (len=2),  dimension(lreg)        :: num
      character (len=20), dimension(lreg,4)      :: bfname
      character (len=1)                          :: it

ccccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccc

      integer                                  :: status,ncid,imid,jmid,
     ^                                            mtotid,elapmid,
     ^                                            lonid,latid,mid,tid,
     ^                                            bfmapid(lreg),
     ^                                            vdims4(4),
     ^                                            start4(4),count4(4)

c000000000 end variable declarations 0000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable description 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000
c
ccccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccc
c
c misval         : The designated TransCom missing value: 1.0e+36.
c r              : Loop index. Loops over lreg.
c regn           : A character variable used for the 'long_name' netCDF
c                  attribute.
c num            : A character array containing the basis function region
c                  number: 01 through 11.
c bfname         : A character array containing the variable names for
c                  the series of netCDF arrays associated with the 2D CO2
c                  basis function region concentrations: terrestrial
c                  surface, terrestrial layer above surface, oceanic
c                  surface, oceanic layer above surface.
c
ccccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccccc
c
c ncid           : The netCDF ID representing the open file.
c XXXXXid        : A series of ID's that correspond to those variables
c                  names passed into the write_cdf subroutine. There are
c                  three exceptions to this scheme. They are:
c vdims#         : These are netCDF integer vectors whose elements
c                  denote the dimensions of the netCDF arrays.
c start#         : These are netCDF integer vectors whose elements denote
c                  where in the netCDF array the first element of a given
c                  write statement should start.
c count#         : These are netCDF integer vectors whose elements denote
c                  the edge lengths along each dimension of a given write
c                  statement.
c
c000000000 end variable description 0000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable initialization 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000

      bfname(1,1)  = 'landunit_s_reg_01'; num(1)  = '01'
      bfname(2,1)  = 'landunit_s_reg_02'; num(2)  = '02'
      bfname(3,1)  = 'landunit_s_reg_03'; num(3)  = '03'
      bfname(4,1)  = 'landunit_s_reg_04'; num(4)  = '04'
      bfname(5,1)  = 'landunit_s_reg_05'; num(5)  = '05'
      bfname(6,1)  = 'landunit_s_reg_06'; num(6)  = '06'
      bfname(7,1)  = 'landunit_s_reg_07'; num(7)  = '07'
      bfname(8,1)  = 'landunit_s_reg_08'; num(8)  = '08'
      bfname(9,1)  = 'landunit_s_reg_09'; num(9)  = '09'
      bfname(10,1) = 'landunit_s_reg_10'; num(10) = '10'
      bfname(11,1) = 'landunit_s_reg_11'; num(11) = '11'

      bfname(1,2)  = 'landunit_sml_reg_01'
```

```
      bfname(2,2)  = 'landunit_sm1_reg_02'
      bfname(3,2)  = 'landunit_sm1_reg_03'
      bfname(4,2)  = 'landunit_sm1_reg_04'
      bfname(5,2)  = 'landunit_sm1_reg_05'
      bfname(6,2)  = 'landunit_sm1_reg_06'
      bfname(7,2)  = 'landunit_sm1_reg_07'
      bfname(8,2)  = 'landunit_sm1_reg_08'
      bfname(9,2)  = 'landunit_sm1_reg_09'
      bfname(10,2) = 'landunit_sm1_reg_10'
      bfname(11,2) = 'landunit_sm1_reg_11'

      bfname(1,3)  = 'oceanunit_s_reg_01'
      bfname(2,3)  = 'oceanunit_s_reg_02'
      bfname(3,3)  = 'oceanunit_s_reg_03'
      bfname(4,3)  = 'oceanunit_s_reg_04'
      bfname(5,3)  = 'oceanunit_s_reg_05'
      bfname(6,3)  = 'oceanunit_s_reg_06'
      bfname(7,3)  = 'oceanunit_s_reg_07'
      bfname(8,3)  = 'oceanunit_s_reg_08'
      bfname(9,3)  = 'oceanunit_s_reg_09'
      bfname(10,3) = 'oceanunit_s_reg_10'
      bfname(11,3) = 'oceanunit_s_reg_11'

      bfname(1,4)  = 'oceanunit_sm1_reg_01'
      bfname(2,4)  = 'oceanunit_sm1_reg_02'
      bfname(3,4)  = 'oceanunit_sm1_reg_03'
      bfname(4,4)  = 'oceanunit_sm1_reg_04'
      bfname(5,4)  = 'oceanunit_sm1_reg_05'
      bfname(6,4)  = 'oceanunit_sm1_reg_06'
      bfname(7,4)  = 'oceanunit_sm1_reg_07'
      bfname(8,4)  = 'oceanunit_sm1_reg_08'
      bfname(9,4)  = 'oceanunit_sm1_reg_09'
      bfname(10,4) = 'oceanunit_sm1_reg_10'
      bfname(11,4) = 'oceanunit_sm1_reg_11'

c000000000 end variable initialization 0000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 open netCDF file and define the dimensions 0000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CREATE('output.L2.4'//it//'.nc',0,ncid)
      write(*,*) 'output.L2.4'//it//'.nc created'

      status = NF_DEF_DIM(ncid,'longitude',im,imid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'latitude',jm,jmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'pulse',mtot,mtotid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'time',elapm,elapmid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end open of netCDF file and dimension definition 000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,regions,time) ccccccccccccccccccc

      status = NF_DEF_VAR(ncid,'longitude',NF_FLOAT,1,imid,lonid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'latitude',NF_FLOAT,1,jmid,latid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'pulse',NF_FLOAT,1,mtotid,mid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'time',NF_FLOAT,1,elapmid,tid)
      if (status .ne. nf_noerr) call handle_err(status)
```

```
ccccccccccc basis function surface maps ccccccccccccccccccccccccccccccccc

c bfmap (landunit_s,landunit_sml,oceanunit_s,oceanunit_sml)

      vdims4(1) = imid; vdims4(2) = jmid; vdims4(3) = mtotid
      vdims4(4) = elapmid

      Do r = 1, lreg
         status = NF_DEF_VAR(ncid,bfname(r,cnt),NF_FLOAT,4,vdims4,
     ^                       bfmapid(r))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

c000000000 end define variables 0000000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin apply attributes 000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,regions,time) ccccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,lonid,'long_name',29,
     ^                         'Longitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lonid,'units',12,'degrees_east')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,latid,'long_name',28,
     ^                         'Latitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,latid,'units',13,'degrees_north')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,mid,'long_name',5,'pulse')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,mid,'units',5,'pulse')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,tid,'long_name',13,'elapsed month')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,tid,'units',13,'elapsed month')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc basis function surface maps ccccccccccccccccccccccccccccccccc

c bfmap (landunit_s,landunit_sml,oceanunit_s,oceanunit_sml)

      Do r = 1, lreg
         If (cnt==1) then
            regn='Terrestrial basis function, surface layer: region '
     ^           //num(r)
         Elseif (cnt==2) then
         regn='Terrestrial basis function, layer above surface: region '
     ^           //num(r)
         Elseif (cnt==3) then
            regn='Oceanic basis function, surface layer: region '
     ^           //num(r)
         Elseif (cnt==4) then
            regn='Oceanic basis function, layer above surface: region '
     ^           //num(r)
         Endif
         status = NF_PUT_ATT_TEXT(ncid,bfmapid(r),'long_name',58,regn)
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_TEXT(ncid,bfmapid(r),'units',4,'ppmv')
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_REAL(ncid,bfmapid(r),'missing_value',
     ^                            NF_FLOAT,1,misval)
         if (status .ne. nf_noerr) call handle_err(status)
      End do
```

```
ccccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccccccc

      status = NF_ENDDEF(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

c0000000000 end apply attributes 0000000000000000000000000000000000000000000

c00000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000 begin filling variables 0000000000000000000000000000000000000000
c00000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,time) ccccccccccccccccccccccccccc

      status = NF_PUT_VAR_REAL(ncid,lonid,lonvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,latid,latvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,mid,mvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,tid,evect)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc basis function surface maps ccccccccccccccccccccccccccccccc

c bfmap (landunit_s,landunit_sml,oceanunit_s,oceanunit_sml)

      start4(1) = 1;  start4(2)= 1; start4(3) = 1; start4(4) = 1
      count4(1) = im; count4(2)= jm; count4(3) = mtot; count4(4) = elapm

      Do r = 1, lreg
         status = NF_PUT_VARA_REAL(ncid,bfmapid(r),start4,count4,
     ^                             bfmap(:,:,r,:,:))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

c0000000000 end filling variables 0000000000000000000000000000000000000000000

c00000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000 close file and end subroutine 0000000000000000000000000000000000
c00000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CLOSE(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

      Return
      End Subroutine write_cdf_2_4


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c00000000000000000000000000000000000000000000000000000000000000000000000000
c00000000000000000000000000000000000000000000000000000000000000000000000000
c0                     WRITE_CDF_2_5 SUBROUTINE                           00
c0                                                                        00
c0 This subroutine writes TransCom 3 level 2 output to a netCDF file      00
c0 called "output.L2.5.nc"                                                00
c00000000000000000000000000000000000000000000000000000000000000000000000000
c00000000000000000000000000000000000000000000000000000000000000000000000000

      Subroutine write_cdf_2_5(m,im,jm,pm,mtot,lreg,elapm,
     ^                         lonvect,latvect,presvect,mvect,lvect,
     ^                         evect,landunit,monpulse)

c00000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000 begin variable declarations 0000000000000000000000000000000000
c00000000000000000000000000000000000000000000000000000000000000000000000000

      implicit none
```

```
      include 'netcdf.inc'
      integer                                    :: m,im,jm,pm,r,
     ^                                              mtot,lreg,elapm

      real, parameter                            :: misval=1.0e+36

      real, dimension(:),              intent(in) :: lonvect,latvect,
     ^                                              presvect,mvect,
     ^                                              lvect,evect

      real, dimension(:,:,:,:,:),      intent(in) :: landunit

      character (len=80)                         :: regn
      character (len=2),  dimension(lreg)        :: num
      character (len=15), dimension(lreg)        :: landname
      character (len=8),  dimension(mtot)        :: monpulse

ccccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccc

      integer                                   :: status,ncid,imid,
     ^                                             jmid,pmid,elapmid,
     ^                                             lonid,latid,presid,
     ^                                             mid,tid,
     ^                                             landunit_regid(lreg),
     ^                                             vdims4(4),
     ^                                             start4(4),count4(4)

c000000000 end variable declarations 00000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000C000000
c000000000 begin variable description 00000C0000000000000000000000000000000
c0000000000000000000000000000000000000000000C0000000000000000000000000000000
c
ccccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccccccc
c
c misval        : The designated TransCom missing value: 1.0e+36.
c r             : Loop index. Loops over lreg.
c regn          : A character variable used for the 'long_name' netCDF
c                 attribute.
c num           : A character array containing the basis function region
c                 number: 01 through 11.
c landname      : A character array containing the variable names for
c                 the series of netCDF arrays associated with the CO2
c                 terrestrial basis function region 3D concentrations.
c monpulse      : A character array containing the month-pulse names.
c
ccccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccccccc
c
c ncid          : The netCDF ID representing the open file.
c XXXXXid       : A series of ID's that correspond to those variables
c                 names passed into the write_cdf subroutine. There are
c                 three exceptions to this scheme. They are:
c      landunit_reg# : These ID's reduce the incoming landunit 5D array
c                      into 4D arrays. There are 11 of these ID's, one
c                      one for each terrestrial basis function region.
c vdims#        : These are netCDF integer vectors whose elements
c                 denote the dimensions of the netCDF arrays.
c start#        : These are netCDF integer vectors whose elements denote
c                 where in the netCDF array the first element of a given
c                 write statement should start.
c count#        : These are netCDF integer vectors whose elements denote
c                 the edge lengths along each dimension of a given write
c                 statement.
c
c000000000 end variable description 00000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000C000
c000000000 begin variable initialization 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000C0000000000000000000000000000000
```

```
      landname(1)  = 'landunit_reg_01'; num(1)  = '01'
      landname(2)  = 'landunit_reg_02'; num(2)  = '02'
      landname(3)  = 'landunit_reg_03'; num(3)  = '03'
      landname(4)  = 'landunit_reg_04'; num(4)  = '04'
      landname(5)  = 'landunit_reg_05'; num(5)  = '05'
      landname(6)  = 'landunit_reg_06'; num(6)  = '06'
      landname(7)  = 'landunit_reg_07'; num(7)  = '07'
      landname(8)  = 'landunit_reg_08'; num(8)  = '08'
      landname(9)  = 'landunit_reg_09'; num(9)  = '09'
      landname(10) = 'landunit_reg_10'; num(10) = '10'
      landname(11) = 'landunit_reg_11'; num(11) = '11'

c000000000 end variable initialization 0000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 open netCDF file and define the dimensions 0000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CREATE('output.L2.5_'//monpulse(m)//'.nc',0,ncid)
      write(*,*) 'output.L2.5_'//monpulse(m)//'.nc created'

      status = NF_DEF_DIM(ncid,'longitude',im,imid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'latitude',jm,jmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'height',pm,pmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'time',elapm,elapmid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end open of netCDF file and dimension definition 000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,pressure,time) ccccccccccccccccc

      status = NF_DEF_VAR(ncid,'longitude',NF_FLOAT,1,imid,lonid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'latitude',NF_FLOAT,1,jmid,latid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'height',NF_FLOAT,1,pmid,presid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'time',NF_FLOAT,1,elapmid,tid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccccc
c
c landunit_reg1,landunit_reg2....

      vdims4(1)=imid; vdims4(2)=jmid; vdims4(3)=pmid; vdims4(4)=elapmid

      Do r = 1, lreg
         status = NF_DEF_VAR(ncid,landname(r),NF_FLOAT,4,vdims4,
     ^                       landunit_regid(r))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

c000000000 end define variables 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin apply attributes 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,pressure,time) ccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,lonid,'long_name',29,
```

```
     ^                        'Longitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lonid,'units',12,'degrees_east')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,latid,'long_name',28,
     ^                        'Latitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,latid,'units',13,'degrees_north')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,presid,'long_name',25,
     ^                        'Vertical gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,presid,'units',9,'millibars')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,tid,'long_name',13,'elapsed month')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,tid,'units',13,'elapsed month')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccc

c landunit_reg1,landunit_reg2....

      Do r = 1, lreg
         regn = '3D terrestrial basis function field: region '//num(r)//
     ^', '//monpulse(m)
         status = NF_PUT_ATT_TEXT(ncid,landunit_regid(r),'long_name',57,
     ^                           regn)
         if (status .ne. nf_noerr) call handle_err(status)
         status=NF_PUT_ATT_TEXT(ncid,landunit_regid(r),'units',4,'ppmv')
         if (status .ne. nf_noerr) call handle_err(status)
         status=NF_PUT_ATT_REAL(ncid,landunit_regid(r),'missing_value',
     ^                          NF_FLOAT,1,misval)
         if (status .ne. nf_noerr) call handle_err(status)
      End do

ccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccc

      status = NF_ENDDEF(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

cOOOOOOOOO end apply attributes OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO


cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOO begin filling variables OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
cOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

ccccccccc vectors (longitude,latitude,pressure,time) ccccccccccccccccc

      status = NF_PUT_VAR_REAL(ncid,lonid,lonvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,latid,latvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,presid,presvect)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_VAR_REAL(ncid,tid,evect)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccc
c
c landunit_reg1,landunit_reg2....

      start4(1) = 1;  start4(2) = 1;  start4(3) = 1;  start4(4) = 1
      count4(1) = im; count4(2) = jm; count4(3) = pm; count4(4) = elapm

      Do r = 1, lreg
         status = NF_PUT_VARA_REAL(ncid,landunit_regid(r),start4,count4,
```

```
     ^                              landunit(:,:,:,r,:))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

c000000000 end filling variables 0000000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c00000000 close file and end subroutine 00000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CLOSE(ncid)
      if (status .ne. nf_noerr) call handle_err(status)

      Return
      End Subroutine write_cdf_2_5


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0                     WRITE_CDF_2_6 SUBROUTINE                           00
c0                                                                        00
c0 This subroutine writes TransCom 3 level 2 output to a netCDF file      00
c0 called "output.L2.6.nc"                                                00
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      Subroutine write_cdf_2_6(m,im,jm,pm,mtot,lreg,elapm,
     ^                         lonvect,latvect,presvect,mvect,lvect,
     ^                         evect,oceanunit,monpulse)

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable declarations 00000000000000000000000000000000000
c0c0000000000000000000000000000000000000000000000000000000000000000000000

      implicit none
      include 'netcdf.inc'

      integer                                    :: m,im,jm,pm,r,
     ^                                              mtot,lreg,elapm

      real, parameter                            :: misval=1.0e+36
      real, dimension(:),             intent(in) :: lonvect,latvect,
     ^                                              presvect,mvect,
     ^                                              lvect,evect
      real, dimension(:,:,:,:,:),     intent(in) :: oceanunit

      character (len=80)                         :: regn
      character (len=2),  dimension(lreg)        :: num
      character (len=16), dimension(lreg)        :: oceanname
      character (len=8),  dimension(mtot)        :: monpulse

ccccccccc netCDF integer declarations ccccccccccccccccccccccccccccccc

      integer                                  :: status,ncid,imid,jmid,
     ^                                            pmid,elapmid,
     ^                                            lonid,latid,presid,
     ^                                            mid,tid,
     ^                                            oceanunit_regid(lreg),
     ^                                            vdims4(4),
     ^                                            start4(4),count4(4)

c000000000 end variable declarations 0000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable description 000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
c
```

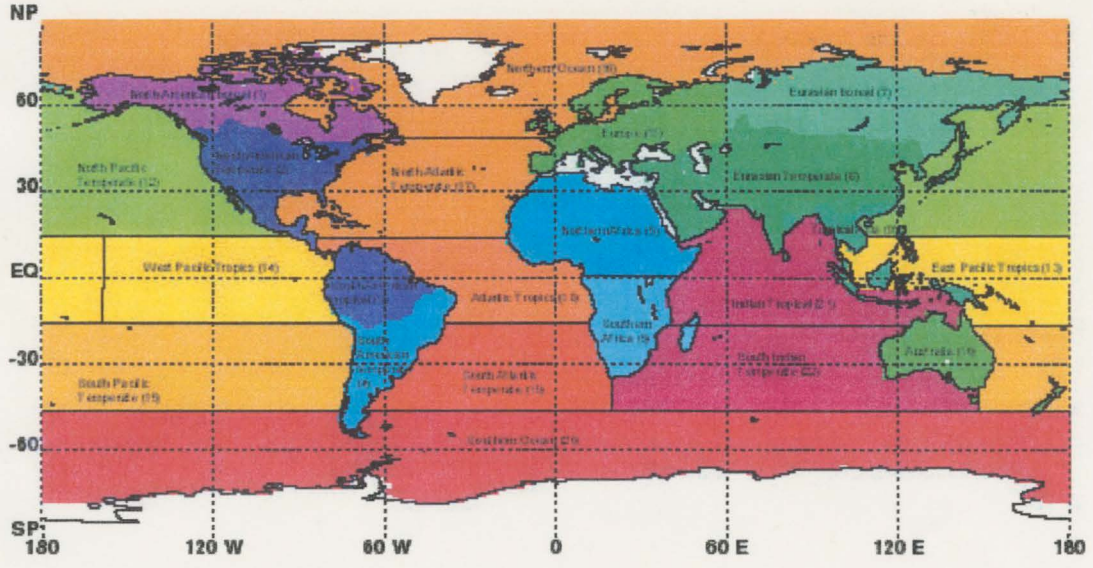
```
ccccccccc regular variables ccccccccccccccccccccccccccccccccccccccccc
c
c misval        : The designated TransCom missing value: 1.0e+36.
c r             : Loop index. Loops over lreg.
c regn          : A character variable used for the 'long_name' netCDF
c                 attribute.
c num           : A character array containing the basis function region
c                 number: 01 through 11.
c oceanname     : A character array containing the variable names for
c                 the series of netCDF arrays associated with the CO2
c                 oceanic basis function region 3D concentrations.
c monpulse      : A character array containing the month-pulse names.
c
ccccccccc netcdf variables ccccccccccccccccccccccccccccccccccccccccc
c
c ncid          : The netCDF ID representing the open file.
c XXXXXid       : A series of ID's that correspond to those variables
c                 names passed into the write_cdf subroutine. There are
c                 three exceptions to this scheme. They are:
c      oceanunit_reg# : These ID's reduce the incoming oceanunit 5D array
c                       into 4D arrays. There are 11 of these ID's, one
c                       one for each oceanic basis function region.
c vdims#        : These are netCDF integer vectors whose elements
c                 denote the dimensions of the netCDF arrays.
c start#        : These are netCDF integer vectors whose elements denote
c                 where in the netCDF array the first element of a given
c                 write statement should start.
c count#        : These are netCDF integer vectors whose elements denote
c                 the edge lengths along each dimension of a given write
c                 statement.
c
c000000000 end variable description 0000000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin variable initialization 0000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      oceanname(1)  = 'oceanunit_reg_01'; num(1)  = '01'
      oceanname(2)  = 'oceanunit_reg_02'; num(2)  = '02'
      oceanname(3)  = 'oceanunit_reg_03'; num(3)  = '03'
      oceanname(4)  = 'oceanunit_reg_04'; num(4)  = '04'
      oceanname(5)  = 'oceanunit_reg_05'; num(5)  = '05'
      oceanname(6)  = 'oceanunit_reg_06'; num(6)  = '06'
      oceanname(7)  = 'oceanunit_reg_07'; num(7)  = '07'
      oceanname(8)  = 'oceanunit_reg_08'; num(8)  = '08'
      oceanname(9)  = 'oceanunit_reg_09'; num(9)  = '09'
      oceanname(10) = 'oceanunit_reg_10'; num(10) = '10'
      oceanname(11) = 'oceanunit_reg_11'; num(11) = '11'

c000000000 end variable initialization 0000000000000000000000000000000000

c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 open netCDF file and define the dimensions 00000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

      status = NF_CREATE('output.L2.6_'//monpulse(m)//'.nc',0,ncid)
      Write(*,*) 'output.L2.6_'//monpulse(m)//'.nc created'

      status = NF_DEF_DIM(ncid,'longitude',im,imid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'latitude',jm,jmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'height',pm,pmid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_DIM(ncid,'time',elapm,elapmid)
      if (status .ne. nf_noerr) call handle_err(status)

c000000000 end open of netCDF file and dimension definition 000000000000
```

```
c000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin define variables 000000000000000000000000000000000000000000
c000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,pressure,time) ccccccccccccccccc

      status = NF_DEF_VAR(ncid,'longitude',NF_FLOAT,1,imid,lonid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'latitude',NF_FLOAT,1,jmid,latid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'height',NF_FLOAT,1,pmid,presid)
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_DEF_VAR(ncid,'time',NF_FLOAT,1,elapmid,tid)
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccc
c
c oceanunit_reg1,oceanunit_reg2.......

      vdims4(1)=imid; vdims4(2)=jmid; vdims4(3)=pmid; vdims4(4)=elapmid

      Do r = 1, lreg
         status = NF_DEF_VAR(ncid,oceanname(r),NF_FLOAT,4,vdims4,
     ^                       oceanunit_regid(r))
         if (status .ne. nf_noerr) call handle_err(status)
      End do

c000000000 end define variables 000000000000000000000000000000000000000000


c000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin apply attributes 000000000000000000000000000000000000000000
c000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,pressure,time) ccccccccccccccccc

      status = NF_PUT_ATT_TEXT(ncid,lonid,'long_name',29,
     ^                         'Longitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,lonid,'units',12,'degrees_east')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,latid,'long_name',28,
     ^                         'Latitudinal gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,latid,'units',13,'degrees_north')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,presid,'long_name',25,
     ^                         'Vertical gridcell centers')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,presid,'units',9,'millibars')
      if (status .ne. nf_noerr) call handle_err(status)

      status = NF_PUT_ATT_TEXT(ncid,tid,'long_name',13,'elapsed month')
      if (status .ne. nf_noerr) call handle_err(status)
      status = NF_PUT_ATT_TEXT(ncid,tid,'units',13,'elapsed month')
      if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccc

c oceanunit_reg1,oceanunit_reg2....

      Do r = 1, lreg
         regn = '3D oceanic basis function field: region '//num(r)//
     ^', '//monpulse(m)
         status = NF_PUT_ATT_TEXT(ncid,oceanunit_regid(r),'long_name',
     ^                            53,regn)
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_TEXT(ncid,oceanunit_regid(r),'units',4,
     ^                            'ppmv')
```

```
         if (status .ne. nf_noerr) call handle_err(status)
         status = NF_PUT_ATT_REAL(ncid,oceanunit_regid(r),
     ^                            'missing_value',NF_FLOAT,1,misval)
         if (status .ne. nf_noerr) call handle_err(status)
       End do

ccccccccccc close define mode ccccccccccccccccccccccccccccccccccccccccccc

       status = NF_ENDDEF(ncid)
       if (status .ne. nf_noerr) call handle_err(status)

c000000000 end apply attributes 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c000000000 begin filling variables 0000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

ccccccccccc vectors (longitude,latitude,pressure,time) ccccccccccccccccc

       status = NF_PUT_VAR_REAL(ncid,lonid,lonvect)
       if (status .ne. nf_noerr) call handle_err(status)
       status = NF_PUT_VAR_REAL(ncid,latid,latvect)
       if (status .ne. nf_noerr) call handle_err(status)
       status = NF_PUT_VAR_REAL(ncid,presid,presvect)
       if (status .ne. nf_noerr) call handle_err(status)
       status = NF_PUT_VAR_REAL(ncid,tid,evect)
       if (status .ne. nf_noerr) call handle_err(status)

ccccccccccc 3D basis function fields ccccccccccccccccccccccccccccccccccc
c
c oceanunit_reg1,oceanunit_reg2....

       start4(1) = 1;  start4(2) = 1;  start4(3) = 1;  start4(4) = 1
       count4(1) = im; count4(2) = jm; count4(3) = pm; count4(4) = elapm

       Do r = 1, lreg
          status = NF_PUT_VARA_REAL(ncid,oceanunit_regid(r),start4,
     ^                              count4,oceanunit(:,:,:,r,:))
          if (status .ne. nf_noerr) call handle_err(status)
       End do

c000000000 end filling variables 0000000000000000000000000000000000000000


c0000000000000000000000000000000000000000000000000000000000000000000000000
c00000000 close file and end subroutine 000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

       status = NF_CLOSE(ncid)
       if (status .ne. nf_noerr) call handle_err(status)

       Return
       End Subroutine write_cdf_2_6


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000
cO                      Error-flagging subroutine                        OO
c0000000000000000000000000000000000000000000000000000000000000000000000000
c0000000000000000000000000000000000000000000000000000000000000000000000000

       Subroutine Handle_err(status)


       Implicit none
       include 'netcdf.inc'
       integer                                  :: status
```

```
If (status .ne. nf_noerr) then
  Write(*,*) NF_STRERROR(status)
  Stop
End if

Return
End subroutine handle_err
```

Appendix B. Basis function map



Appendix C. Station list

There are a total of 228 gridcells for which high frequency reporting will be made.

All locations in blue require *only* reporting of the ***neighboring ocean gridcell in the direction specified*** (the "direction" column in the table below).[11] Otherwise, the resident gridcell is reported. Complete instructions on choosing the appropriate model gridcell are as follows:

- For sites with *no direction given*, use the nearest gridpoint to the latitude and longitude given. Please check whether the gridpoint is land or ocean and that this agrees with the listed surface type. For models with continuous surface masks use your discretion. A reasonable guide might be >75% agreement between your model grid and the listed surface type would constitute a match. If a match to the listed surface type is not achieved, check the other three gridpoints surrounding the site location and pick the one that has the correct surface type. If none have the correct surface type, use the nearest gridpoint anyway. Also use a nearest gridpoint, if the original point chosen is being sampled already for another site (this is quite likely in Europe depending on your grid size).
- For sites where a *direction is given*, use the nearest gridpoint to the given latitude and longitude in the given direction. If it is not the type required, check for the presence of a secondary direction in the station list information and check if it is the type required. If this does not retrieve the desired surface type, move outward one gridcell in the original direction. If this is still not the required surface type, do the same for a secondary direction, if there is one. If this does not work, use your own discretion to choose another nearby gridpoint.
- Use the nearest grid point for Antarctic sites regardless of whether it is land or ocean Ensure that you are using the same model gridpoint for Cape Grim and the Bass Strait aircraft points.
- Altitude: Report surface layer data for all sites except those listed with an "NS" (an abbreviation of "non-surface"). Interpolate or use the nearest model level if within approx. 100m (number picked at random).
- Output: Please submit with your output a text file containing the latitude, longitude, altitude, model level (or levels interpolated between) and the surface type for each site.

NOTE: In some cases, two or more stations clearly populated the same gridcell/height combination irrespective of model grid. Therefore, all but one of the overlapping stations were eliminated from the list. The one remaining station used is for locational purposes *only* and does not imply that data from that station will be used in lieu of the overlapping stations that were eliminated.

Missing values are denoted as "-999"

Station	latitude	longitude	elev (m)	type1[12]	type2[13]	Direction	NS
GLOBALVIEW[14]							
Bass Strait/Cape Grim	-40.38	144.39	500	a	O	SW	NS
Bass Strait/Cape Grim	-40.38	144.39	1500	a	O	SW	NS
Bass Strait/Cape Grim	-40.38	144.39	2500	a	O	SW	NS
Bass Strait/Cape Grim	-40.38	144.39	3500	a	O	SW	NS
Bass Strait/Cape Grim	-40.38	144.39	4500	a	O	SW	NS
Bass Strait/Cape Grim	-40.38	144.39	5500	a	O	SW	NS
Bass Strait/Cape Grim	-40.38	144.39	6500	a	O	SW	NS
Alert, Greenland	82.45	-62.52	210	u	O	SW	

[11] Some of these were from a statistical analysis of the station reporting while other were guesses based on physical location and prevailing winds.

[12] "a" - aircraft, "u" - continuous analyzer, "f" - flask, "t" - tower.

[13] "O" = ocean, "L" - land.

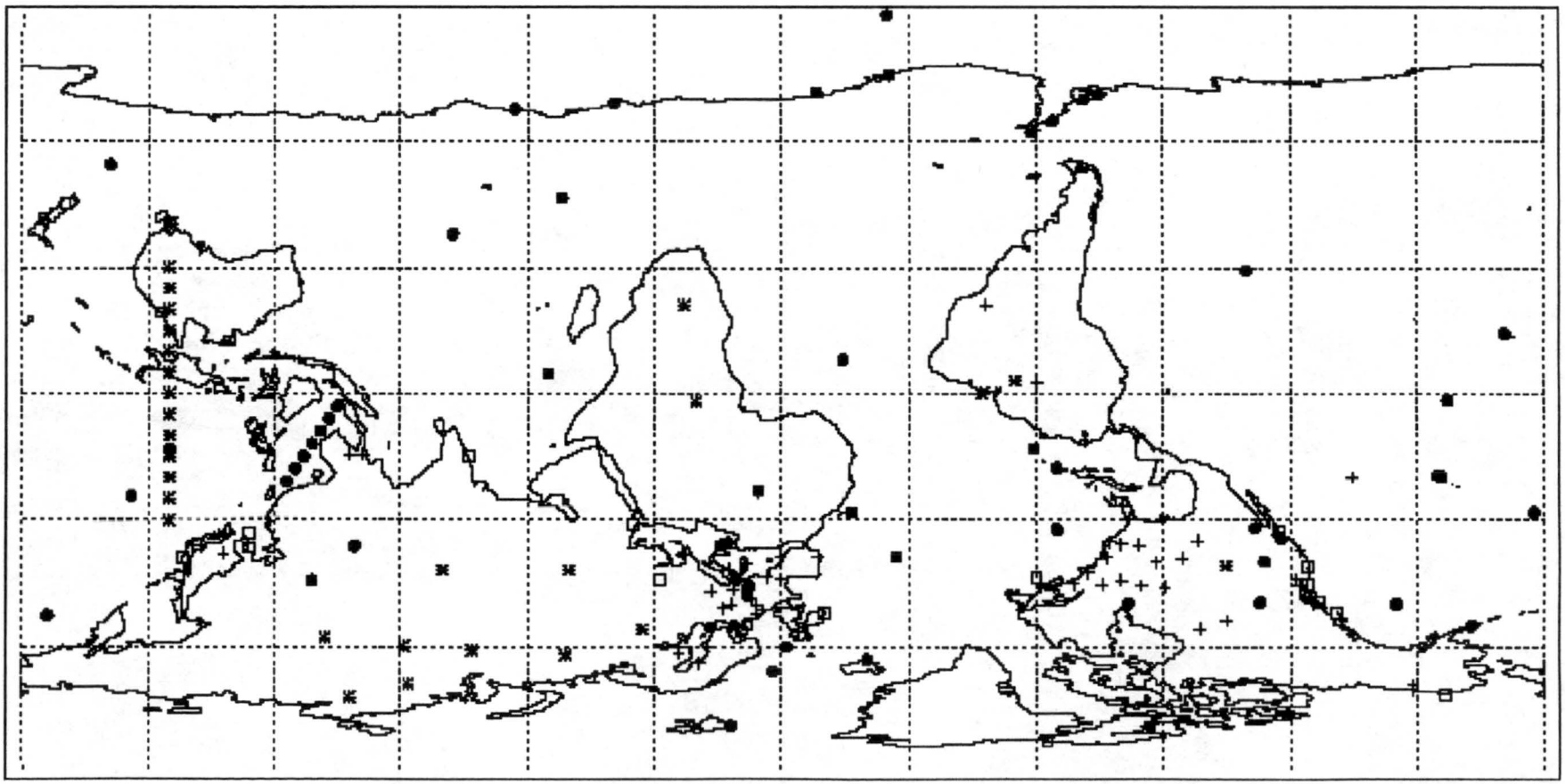
[14] The Pacific Ocean ship measurements were eliminated from the high-frequency reporting because it is unlikely they would differ much from the monthly means which are reported for every gridcell.

Amsterdam Island	-37.95	77.53	150	f	O		
Ascension Island	-7.92	-14.42	54	f	O		
Assekrem, Algeria	23.18	5.42	2728	f	L		NS
St. Croix, Virgin Is.	17.75	-64.75	3	f	O		
Azores	38.75	-27.08	30	f	O		
Baltic Sea, Poland	55.50	16.67	7	f	O		
Baring Head St., NZ	-41.41	174.87	80	u	O	S,SW	
Bermuda West	32.27	-64.88	30	f	O		
Barrow, Alaska	71.32	-156.60	11	u	O	E,NE	
Black Sea, Romania	44.17	28.68	3	u	O	NE	
Carr, CO	40.90	-104.80	3000	a	L		NS
Carr, CO	40.90	-104.80	4000	a	L		NS
Carr, CO	40.90	-104.80	5000	a	L		NS
Carr, CO	40.90	-104.80	6000	a	L		NS
Cold Bay, Alaska	55.20	-162.72	25	f	O		
Cape Ferguson, Aust.	-19.28	147.06	2	u	O	E	
Cape Grim, Tasmania	-40.68	144.69	94	u	O	SW	
Christmas Island	1.70	-157.17	3	f	O		
Mt. Cimone St., Italy	44.18	10.70	2165	f	L		NS
Cape Meares, OR	45.48	-123.97	30	u	O	W,NW	
Cape Rama, India	15.08	73.83	60	u	O	S,SE	
Crozet, Indian Ocean	-46.45	51.85	120	f	O		
Cape St. James, Canada	51.93	-131.02	89	u	O	W	
Darwin, Australia	-12.42	130.57	3	u	O	W,NW	
Easter Island	-29.15	-109.43	50	f	O		
Estevan Pt, BC, Canada	49.38	-126.55	39	u	O	W	
Guam	13.43	144.78	2	f	O		
Dwejra Pt., Malta	36.05	14.18	30	f	O		
Halley Bay, Antarctica	-75.67	-25.50	10	f			
Hungary	46.95	16.65	300	t	L		
Storhofdi, Iceland	63.25	-20.15	100	f	O		
North Carolina	35.35	-77.38	60	t	L		
North Carolina	35.35	-77.38	500	t	L		NS
Canary Islands	28.30	-16.48	2360	f	O		NS
Jubany St., Antarctica	-62.23	-58.82	15	f			
Key Biscayne, FL	25.67	-80.20	3	u	O	S	
Kosan, Rep. of Korea	33.28	126.15	72	u	O		
Kumukahi, Hawaii	19.52	-154.82	3	f	O		
Wisconsin tower	45.93	-90.27	500	t	L		NS
Wisconsin tower	45.93	-90.27	850	t	L		NS
Lampedusa, Italy	35.52	12.62	85	f	O		
Mawson St., Antarctica	-67.62	62.87	32	f			
Mould Bay, Canada	76.25	-119.35	58	f	O		
Sand Island, Midway	28.22	-177.37	4	f	O		
Mauna Loa, Hawaii	19.53	-155.58	3397	f	O		NS
Minamitorishima, Japan	24.30	153.97	8	f	O		
Macquarie Island	-54.48	158.97	12	f	O		
Olympic Peninsula, WA	48.25	-124.42	488	u	G	W	
Plateau Rosa St., Italy	45.93	7.70	3480	f	I		NS
Palmer St., Antarctica	-64.92	-64.00	10	f			
Qinghai Province, PRC	36.27	100.92	3810	f	L		NS
Ragged Pt., Barbados	13.17	-59.43	3	f	O		
Ryori St., Japan	39.03	141.83	230	u	O	E	
Schauinsland, Germany	48.00	8.00	1205	f	L		NS
South China Sea	3.00	105.00	15	f	O		
South China Sea	6.00	107.00	15	f	O		
South China Sea	9.00	109.00	15	f	O		
South China Sea	12.00	111.00	15	f	O		
South China Sea	15.00	113.00	15	f	O		
South China Sea	18.00	115.00	15	f	O		
South China Sea	21.00	117.00	15	f	O		
Sable Island, NS, Can.	43.93	-60.02	5	u	O	E	
Mahe Island, Seychelles	-4.67	55.17	3	f	O		
Shemya Island, Alaska	52.72	174.10	40	f	O		
Shetland Is., Scotland	60.17	-1.17	30	f	O		
Samoa	-14.25	-170.57	42	f	O		
South Pole	-89.98	-24.80	2830	f			
Atlantic Ocean, Norway	66.00	2.00	7	f	O		
Pacific Ocean, Canada	50.00	-145.00	7	f	O		
Syowa, St., Antarctica	-69.00	39.58	11	f			
Tae-ahn Pen., Korea	36.73	126.13	20	u	O		

Wendover, Utah	39.90	-113.72	1320	f	L		
Ulaan Uul, Mongolia	44.45	111.10	914	f	L		
Westerland, North Sea	55.00	8.00	8	u	O	W	
Sede Boker, Israel	31.13	34.88	400	u	O	NW	
Zeppelin St., Norway	78.90	11.88	474	f	O		
FLUX TOWERS							
amer.ca01flan.01	49.60	-112.60	951	t	L		
amer.ca01roul.01	45.41	-75.52	-999	t	L		
amer.ca02vanc.01	49.85	-125.32	300	t	L		
amer.ca03born.01	55.90	-98.50	259	t	L		
amer.ca08cmpb.01	44.32	-79.93	120	t	L		
amer.ca11sbor.05	53.90	-104.70	579	t	L		
amer.cs00lase.01	10.43	-83.98	200	t	L		
amer.usakhapp.01	69.13	-148.83	366	t	L		
amer.usakupad.01	70.27	-148.88	3	t	L		
amer.uscajasp.01	37.40	-122.22	100	t	L		
amer.uscaskyo.01	33.37	-116.62	1420	t	L		
amer.usflcypr.01	29.73	-82.08	50	t	L		
amer.usilbond.01	40.00	-88.29	213	t	L		
amer.uskskonz.01	39.12	-94.35	324	t	L		
amer.usmaharv.01	42.54	-72.18	300	t	L		
amer.usmdsmit.01	38.88	-76.55	11	t	L		
amer.usmehowl.01	45.25	-68.75	60	t	L		
amer.usmiumbs.01	45.58	-84.70	234	t	L		
amer.usncduke.01	35.87	-79.98	163	t	L		
amer.usokliww.01	34.96	-97.98	-999	t	L		
amer.usorjunp.01	44.27	-121.38	945	t	L		
amer.ustnwalk.01	35.95	-84.28	365	t	L		
amer.uswawind.01	45.82	-121.97	355	t	L		
amer.uswipark.01	45.92	-90.27	470	t	L		
aust.as00warr.01	-19.93	134.60	336	t	L		
euro.be00bras.01	51.30	4.52	10	t	L		
euro.be00viel.01	50.30	6.00	450	t	L		
euro.da00lill.01	55.49	11.65	40	t	L		
euro.fi00hyyt.01	61.85	24.28	181	t	L		
euro.fr00brdx.01	44.08	0.08	60	t	L		
euro.fr00sarr.01	48.67	7.08	300	t	L		
euro.gm00bayr.01	50.15	11.87	780	t	L		
euro.gm00thar.01	50.97	13.63	380	t	L		
euro.it00cast.01	41.75	12.37	3	t	L		
euro.it00coll.01	41.87	13.63	1550	t	L		
euro.nl03wist.01	52.17	5.74	25	t	L		
euro.sw00flak.01	64.12	19.45	225	t	L		
euro.sw00noru.01	60.08	17.47	45	t	L		
euro.uk00aber.01	56.62	-3.80	340	t	L		
flux.br00sugr.01	-21.10	-48.07	520	t	L		
flux.gm00soll.01	57.77	9.58	505	t	L		
flux.it00reno.01	46.59	11.43	1750	t	L		
flux.ja00toma.01	42.59	141.59	75	t	L		
flux.th27sakr.01	14.49	101.92	530	t	L		
flux.th50mark.01	14.59	98.86	170	t	L		
jpan.ja00jawa.01	35.90	139.50	30	t	L		
jpan.ja00tsuk.01	36.02	140.12	20	t	L		
jpan.ja09taka.01	36.13	137.42	1420	t	L		
lbaf.br00biol.01	-2.59	-60.12	90	t	L		
mede.fr00berb.01	43.73	3.58	250	t	L		
mede.gr00gree.01	38.00	22.62	840	t	L		
mede.it14sard.01	40.60	8.15	74	t	L		
mede.po00jarv.01	38.63	-8.60	230	t	L		
mede.sp00unkn.01	38.77	0.25	-999	t	L		
ALEGAGE							
Adrigole, Ireland	52.00	-10.00	50	u	O	SW	
Fraserdale,Ontario, Can.	49.88	-81.57	210	f	L		
Kitt Peak, AZ	31.90	-111.60	2090	f	L		NS
Lauder, NZ	-45.00	169.70	370	u	O	S	
Neumayer, Antarctica	-71.60	-8.30	16	f			
Scripps Pier, CA	32.83	-117.27	14	u	O	W	
Table Mtn., CA	34.40	-117.70	2258	f	L		NS
Trinidad Head, CA	41.05	-124.15	109	u	O	W	
AIRCRAFT							
Tokyo-Syd aircraft	30.00	145.00	10500	a	O		NS

Tokyo-Syd aircraft	25.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	20.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	15.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	10.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	5.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	0.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	-5.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	-10.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	-15.00	145.00	10500	a	O	NS
Tokyo-Syd aircraft	-20.00	145.00	10500	a	L	NS
Tokyo-Syd aircraft	-25.00	145.00	10500	a	L	NS
Tokyo-Syd aircraft	-30.00	145.00	10500	a	L	NS
Fyodorovskoye	56.00	33.00	500	a	L	NS
Fyodorovskoye	56.00	33.00	1500	a	L	NS
Fyodorovskoye	56.00	33.00	2500	a	L	NS
Fyodorovskoye	56.00	33.00	3500	a	L	NS
Fyodorovskoye	56.00	33.00	5000	a	L	NS
Syktyvkar	62.00	51.00	500	a	L	NS
Syktyvkar	62.00	51.00	1500	a	L	NS
Syktyvkar	62.00	51.00	2500	a	L	NS
Syktyvkar	62.00	51.00	3500	a	L	NS
Syktyvkar	62.00	51.00	5000	a	L	NS
Zotino	60.00	89.00	500	a	L	NS
Zotino	60.00	89.00	1500	a	L	NS
Zotino	60.00	89.00	2500	a	L	NS
Zotino	60.00	89.00	3500	a	L	NS
Zotino	60.00	89.00	5000	a	L	NS
Surgut	61.00	73.00	500	a	L	NS
Surgut	61.00	73.00	1500	a	L	NS
Surgut	61.00	73.00	2500	a	L	NS
Surgut	61.00	73.00	3500	a	L	NS
Surgut	61.00	73.00	5000	a	L	NS
Norilsk	69.00	88.00	500	a	L	NS
Norilsk	69.00	88.00	1500	a	L	NS
Norilsk	69.00	88.00	2500	a	L	NS
Norilsk	69.00	88.00	3500	a	L	NS
Norilsk	69.00	88.00	5000	a	L	NS
Kirensk	58.00	108.00	500	a	L	NS
Kirensk	58.00	108.00	1500	a	L	NS
Kirensk	58.00	108.00	2500	a	L	NS
Kirensk	58.00	108.00	3500	a	L	NS
Kirensk	58.00	108.00	5000	a	L	NS
Kalahari	-21.00	23.00	500	a	L	NS
Kalahari	-21.00	23.00	1500	a	L	NS
Kalahari	-21.00	23.00	2500	a	L	NS
Kalahari	-21.00	23.00	3500	a	L	NS
Kalahari	-21.00	23.00	5000	a	L	NS
Hatanga	72.00	102.00	500	a	L	NS
Hatanga	72.00	102.00	1500	a	L	NS
Hatanga	72.00	102.00	2500	a	L	NS
Hatanga	72.00	102.00	3500	a	L	NS
Hatanga	72.00	102.00	5000	a	L	NS
LBA-Santarem	-3.00	-55.00	500	a	L	NS
LBA-Santarem	-3.00	-55.00	1000	a	L	NS
LBA-Santarem	-3.00	-55.00	1500	a	L	NS
LBA-Santarem	-3.00	-55.00	2000	a	L	NS
LBA-Santarem	-3.00	-55.00	2500	a	L	NS
LBA-Santarem	-3.00	-55.00	3000	a	L	NS
LBA-Belem	0.00	-47.00	500	a	L	NS
LBA-Belem	0.00	-47.00	1000	a	L	NS
LBA-Belem	0.00	-47.00	1500	a	L	NS
LBA-Belem	0.00	-47.00	2000	a	L	NS
LBA-Belem	0.00	-47.00	2500	a	L	NS
LBA-Belem	0.00	-47.00	3000	a	L	NS
Hypo-Africa	2.00	20.00	500	a	L	NS
Hypo-Africa	2.00	20.00	1500	a	L	NS
Hypo-Africa	2.00	20.00	2500	a	L	NS
Hypo-Africa	2.00	20.00	3500	a	L	NS
Hypo-Africa	2.00	20.00	5000	a	L	NS
Hypo-Africa	2.00	20.00	8000	a	L	NS
Hypo-Africa	2.00	20.00	12000	a	L	NS

Hypo-Siberia1	42.00	50.00	500	a	L	NS
Hypo-Siberia1	42.00	50.00	1500	a	L	NS
Hypo-Siberia1	42.00	50.00	2500	a	L	NS
Hypo-Siberia1	42.00	50.00	3500	a	L	NS
Hypo-Siberia1	42.00	50.00	5000	a	L	NS
Hypo-Siberia1	42.00	50.00	8000	a	L	NS
Hypo-Siberia1	42.00	50.00	12000	a	L	NS
Hypo-Siberia2	42.00	80.00	5000	a	L	NS
Hypo-Siberia2	42.00	80.00	8000	a	L	NS
Hypo-Siberia2	42.00	80.00	12000	a	L	NS



Appendix D. Takahashi oceanic flux data

Proceedings of the 2nd CO_2 in Oceans Symposium,
Tsukuba, JAPAN, January 18-23, 1999 (in press)

Net sea-air CO_2 flux over the global oceans: An improved estimate based on the sea-air pCO_2 difference

Taro Takahashi[1], Rik H. Wanninkhof[2], Richard A. Feely[3], Ray F. Weiss[4],
David W. Chipman[1], Nicholas Bates[5], Jon Olafsson[6], Christopher Sabine[7],
S. C. Sutherland[1]

[1] Lamont-Doherty Earth Observatory of Columbia University, Palisades, NY, USA
e-mail: taka@ldeo.columbia.edu
[2] Atlantic Oceanographic and Meteorological Laboratory, NOAA, Miami, FL, USA
e-mail: wanninkhof@aoml.noaa.gov
[3] Pacific Marine Environmental Laboratory, NOAA, Seattle, WA, USA
e-mail: feely@pmel.noaa.gov
[4] Scripps Institution of Oceanography, Univ. of Calif. San Diego, La Jolla, CA, USA
e-mail: rfw@siorfw.ucsd.edu
[5] Bermuda Biological Station for Research, Bermuda
e-mail: nick@bbsr.edu
[6] Marine Research Institute, Reykjavik, Iceland
e-mail: jon@hafro.is
[7] Atmospheric & Oceanic Science Program, Princeton University, Princeton, NJ, USA
e-mail: sabine@geo.princeton.edu

INTRODUCTION:

The distribution of the oceanic sink and source areas for atmospheric CO_2 and the magnitude of the net CO_2 flux across the sea surface are important for understanding the global carbon cycle. Based on approximately 2.5 million measurements made for the pCO_2 in surface waters of the global ocean since 1960, the climatological distributions of monthly sea-air pCO_2 difference, ΔpCO_2, and the net sea-air flux have been estimated for a reference year of 1995. The method used and the results obtained are presented.

METHOD:

For this study, only the ocean water pCO_2 values measured using direct gas-seawater equilibration methods are used. Observations made in the equatorial Pacific between 10°N and 10°S during El Nino events have been excluded from the data set. Thus, the results shown in this paper represent the climatological distributions under non-El Nino conditions. Since the measurements were made in different years, during which the atmospheric pCO_2 was increasing, they were corrected to a single reference year (arbitrary chosen to be 1995) on the basis of the following assumptions. Surface waters in subtropical gyres mix vertically at slow rates with subsurface waters due to the presence of strong stratification at the base of the mixed layer. This will allow a long contact time with the atmosphere to exchange CO_2. Therefore, their CO_2 chemistry tends to follow the atmospheric CO_2 increase. Accordingly, the pCO_2 in these warm waters follows the increasing trend of atmospheric CO_2, and the sea-air pCO_2 difference tends to be independent of the year of measurements. On the other hand, since surface waters in high latitude regions are replaced by winter convection with upwelling deep waters yearly, their CO_2 properties tend to remain unchanged from year to year reflecting those of deep waters, in which the effect of increased atmospheric CO_2 is diluted to undetectable levels. Accordingly, the sea-air pCO_2 difference measured in a

given year was corrected to the reference year using the observed increase in the atmospheric CO_2 concentration. These adjusted values are binned into a total of 750,000 pixels (72 pixels along the longitude x 41 along the latitude x 365 days), which represent the global 4° x 5° grid for each day in a single virtual calendar year.

Mean monthly global distributions of ΔpCO_2 have been constructed using an interpolation method based on a lateral 2-dimensional advection-diffusion transport equation (Takahashi et al., 1995; Takahashi et al., 1997). The equation yields ΔpCO_2 values for 4° x 5° pixels where no observations exist, while it satisfies the observed values explicitly. The effects of vertical mixing and sea-air CO_2 flux are considered inherently imbedded in the observed data. Therefore, the short-term behavior of surface water properties may be approximated using the lateral transport model without vertical mixing and gas exchange terms. For advective transport, the mean monthly surface flow field of Bryan and Lewis (1979) is used and, for diffusive transport, a constant value of 2000 m^2/sec. The equation has been solved iteratively using a finite-difference algorithm, in which the computational domains are joined at December 31, 1995, with January 1, 1995, and along the prime meridian to ensure continuity in time and space. The ocean-land boundaries are assumed to be reflective boundaries. Singularities at the poles are avoided by the presence of the Antarctic continent in the south and the polar ice cap in the north, which was assumed to be a landmass. Typically, several thousand iterations are necessary before solutions are converged. Although the solutions give daily distributions, monthly mean distributions have been computed and used for flux calculations.

The net sea-air CO_2 flux in each pixel has been computed using the monthly mean ΔpCO_2 values thus obtained, and the Wanninkhof (1992) formulation (Eq. 1) for the effect of wind speed on the CO_2 gas transfer coefficient with the mean monthly wind speed of Esbensen and Kushnir (1981). The combination of the Wanninkhof relation and Esbensen & Kushnir wind data yields a mean global gas transfer rate of 0.063 mole CO_2/m^2/uatm/yr. This is consistent with 19 moles CO_2/m^2/yr (= 0.064 mole CO_2/m^2/uatm/yr) estimated on the basis of carbon-14 distribution in the atmosphere and oceans (Broecker et al., 1986). The distribution maps of ΔpCO_2 during February and August, 1995, over the global oceans are shown in Fig. 1-A and B, and a map for the mean annual net CO_2 flux across the sea surface during 1995 is shown in Fig. 2.

RESULTS:

The ΔpCO_2 maps (Fig. 1) show that oceanic sources for atmospheric CO_2 (indicated by red and yellow colors) are located in the areas of deep water upwelling. The effect of winter upwelling is seen in the sub-arctic western Pacific (Fig. 1-A) and that of upwelling induced by the southwest monsoon during July-August is seen in the Persian Gulf (Fig. 1-B). The strong CO_2 source zone located along the Pacific equatorial belt is supported by the coastal upwelling along South America as well as by the upward entrainment of the equatorial undercurrent water. The source intensity is reduced toward the western Pacific due mainly to CO_2 losses to photosynthesis and to the atmosphere. Strong CO_2 sinks (blue and purple areas) are seen along the pole-ward edges of subtropical gyres, where major warm currents are located. The Gulf Stream in the North Atlantic and the Kuroshio in the North Pacific are both major CO_2 sinks (see Fig. 2) due primarily to cooling as they flow from warm tropical oceans to sub-polar zones. Along the northern border of the Southern Ocean, CO_2 sink areas caused by the cooling of pole-ward flowing currents such as the Brazil current located along eastern South America, the Agulhus current located south of South Africa, and the East Australian current located along southeastern Australia. These warm water currents meet with cold currents flowing equator-ward from the Antarctic zone along the northern border of the Southern Ocean. As the sub-Antarctic waters rich in nutrients flow northward to more sun-lit regions, CO_2 is drawn down by photosynthesis thus creating strong CO_2 sink conditions. Confluence of subtropical waters with polar waters forms broad and strong CO_2 sink zones as a result of the juxtaposition of the lowering effects on pCO_2 of the cooling of warm waters and the photosynthetic drawdown of CO_2 in sub-polar waters rich in nutrients. This feature is clearly depicted between 40°S and 55°S in Figs. 1 and 2.

The climatological net sea-air CO_2 flux computed for the global ocean for the reference year of 1995 representing non-El Nino conditions is summarized in Table 1. The annual net CO_2 uptake by the global ocean is estimated to be about 2.2 Gt-C/yr. This is consistent with estimates obtained on the basis of ocean-atmosphere models (calibrated using carbon-14 distributions) with constant biology. If the effects of

wind speed on the gas transfer coefficient of Liss and Merlivat (1986) is used instead, the global uptake flux of 1.1 Gt-C/yr is obtained.

The 1995 estimate for the global ocean uptake flux is about 0.7 Gt-C/yr greater than the 1990 estimate. This difference is partly the result of improvements in observational database and partly the effect of an increase in the atmospheric pCO_2 of about 7 uatm that occurred from 1990 to 1995. An increase of about 0.5 Gt-C/yr uptake by the Southern Ocean and the South Indian Ocean may be attributed to the improved database and to a -7 uatm change in ΔpCO_2 over the southern high latitude oceans caused by the increase in atmospheric pCO_2. If a mean ΔpCO_2 over the global high latitude oceans pole-ward of 50° latitude were lowered by 7 uatm (i.e. ocean became a stronger sink), an additional uptake of 0.3 Gt-C/yr would be expected.

The uptake flux for the northern hemisphere oceans (north of 14°N) is 1.22 Gt-C/yr, whereas that for the southern hemisphere oceans (south of 14°S) is 1.79 Gt-C/yr. Thus, the southern hemisphere oceans are a stronger CO_2 sink by about 0.5 Gt-C/yr. This is due partially to the much greater oceanic areas in the southern hemisphere and partially to that the Southern Ocean south of 50°S is an efficient CO_2 sink while it has 10% of the global ocean area, it takes up about 29% of the global ocean CO_2 uptake. Cold temperature and moderate photosynthesis are both responsible for the large uptake by the Southern Ocean.

The 1995 flux values listed in Table 1 shows that the Atlantic Ocean is the largest net sink for atmospheric CO_2 (39%); the Southern Ocean (22%) and the Indian Ocean (22%) the next; and the Pacific (11%) the smallest. The large sink flux of the northern oceanic areas is attributed to the intense biological drawdown of CO_2 in the high latitude areas of the North Atlantic and arctic seas during the summer months. This is also due to low CO_2 concentrations in upwelling deep waters, which are, in turn, caused primarily by the short residence time of the North Atlantic Deep Waters. The small uptake flux of the Pacific can be attributed to the combined sink flux of the northern and southern subtropical gyres is roughly balanced by the source flux from the equatorial Pacific. The equatorial Pacific CO_2 source flux may be totally or partly eliminated during El Nino events. This effect alone could increase the global ocean uptake flux up to 0.6 Gt-C/yr during an El Nino year.

ERROR ESTIMATES:

The sea-air CO_2 flux values reported in this paper are subject to two sources of errors: 1) biases in ΔpCO_2 values interpolated from relatively sparse observations, and 2) errors due to uncertainties in the gas transfer coefficient estimated on the basis of the wind speed dependence.

Possible biases in ΔpCO_2 differences have been estimated by Takahashi et al. (1997) using sea surface water temperatures (SST) as a proxy. The monthly SST in each pixel was computed using the SST values measured concurrently with pCO_2 and the same interpolation scheme used for obtaining the global distribution of ΔpCO_2. This was compared with the climatological SST value obtained by Shea et al. (1992), and the global mean of the differences was computed. The global mean SST estimated on the basis of our data and method has been found to be about 0.4 °C greater than the climatological mean estimated by Shae et al. (see Takahashi et al, 1997). Using the mean effect of SST on seawater pCO_2 observed over the global ocean (3.5% per °C), we estimate that the SST difference corresponds to about 5 uatm or 50% error in ΔpCO_2. Therefore, the estimated global sea-air CO_2 flux is subject to a systematic error of up to 50%.

The reliability of the wind speed dependence on the CO_2 gas transfer coefficient has been significantly improved as a result of the recent GASEX98 study reported by W. McGillis, J. Edson and R. Wanninkhof during this symposium. Eq. (1) of Wanninkhof used in the present study is consistent within about $\pm$ 20% with their new eddy correlation flux measurements conducted over the North Atlantic. Hence, the estimated fluxes are subject to this level of uncertainty.

REFERENCES:

Broecker, W. S., Ledwell, J. R., Takahashi, T., Weiss, R. F., Merlivat, L., Memery, L., Peng, T.-H., Jahne, B. and Munnich, K. O. (1986). Isotopic versus micrometeorologic ocean CO2 fluxes: a serious conflict. Jour. Geophys. Res., 91, 10,517-10,527.

Bryan, K. and Lewis, L. J. (1979). A water mass model of the world ocean. Jour. Geophys. Res., **84**, 2503-2517.

Esbensen, S. K. and Kushnir, Y. (1981). The heat budget of the global ocean: An atlas based on estimates from the surface marine observations. Climatic Research Institute Report #29, Oregon State University, Corvalis, OR.

Liss, P. S. and Merlivat, L. (1986). Air-sea gas exchange rates: introduction and synthesis. In "The Role of Air-Sea Exchange in Geochemical Cycling", P. Buat-Menard editor, D. Reidel Publishing Co., Holland, 113-127.

McGillis, W., Edson, J. and Wanninkhof R. (1999), This symposium.

Shea, D. J., Trenberth, K. E. and Reynolds, R. W. (1992) A global monthly sea surface temperature climatology, Jour. Climate, **5**, 987-1001.

Takahashi, T., Takahashi, T. T. and Sutherland, S. C. (1995). An assessment of the role of the North Atlantic as a CO_2 sink. Phil. Trans. Roy. Soc. London, Series B, **348**, 143-152.

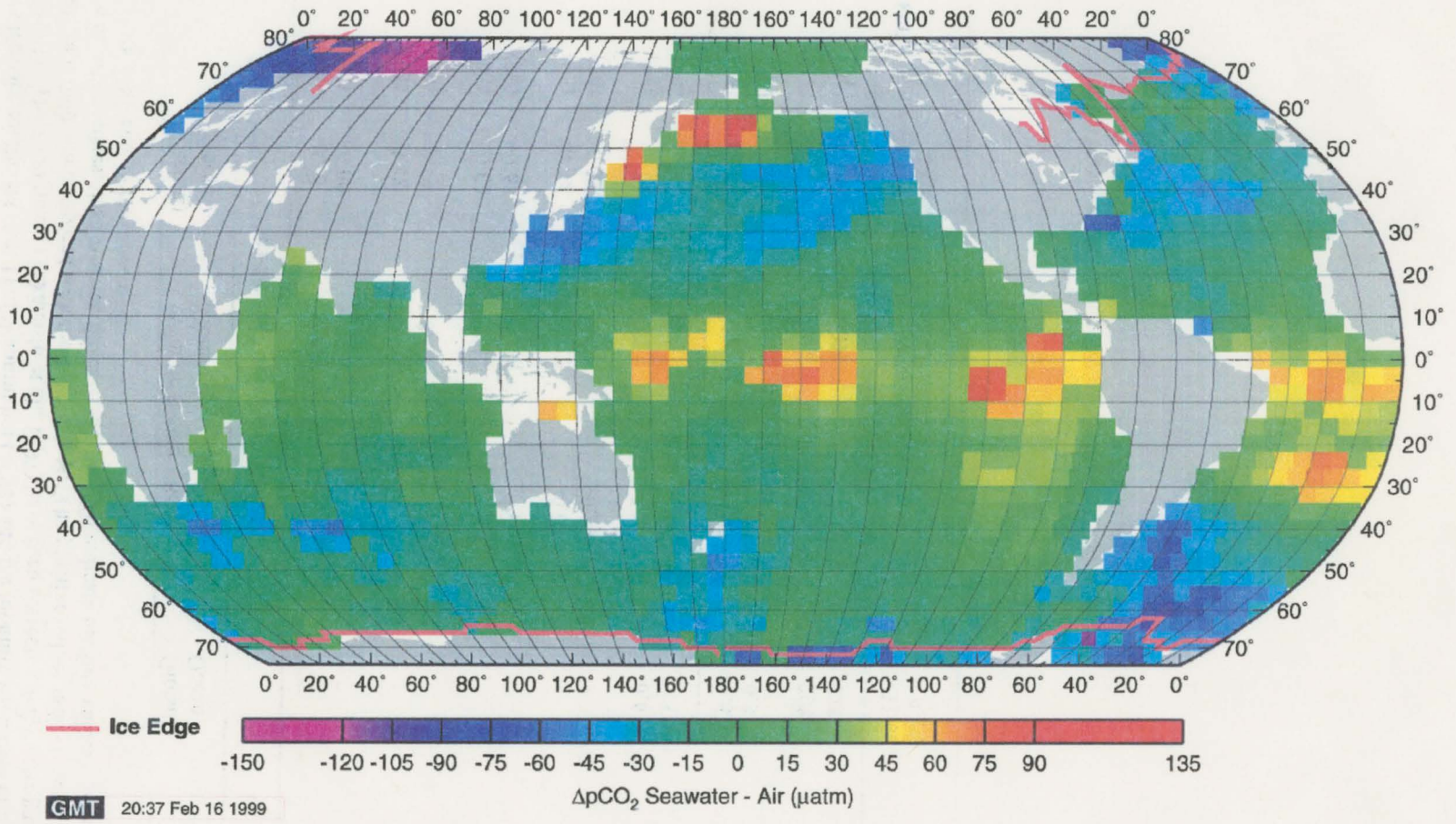
Takahashi, T., Feely, R. A., Weiss, R., Wanninkhof, R. H., Chipman, D. W., Sutherland, S. C. and Takahashi, T. T. (1997). Global air-sea flux of CO_2: an estimate based on measurements of sea-air pCO_2 difference. Proc. Nat. Acad. Sci., **94**, 8292-8299.

Tans, P. P., I. Y. Fung, and T. Takahashi (1990). Observational constraints on the global atmospheric CO_2 budget. Science, **247**, 1431-1438.

Wanninkhof, R. (1992). Relationship between wind speed and gas exchange. Jour. Geophys. Res., **97**, 7373-7382.

Table 1 The net sea-air flux of CO_2 estimated for a reference year of 1995 using the effect of wind speed on the CO_2 gas transfer coefficient (Eq. 1) of Wanninkhof (1992) and the monthly wind field of Esbensen and Kushnir (1981). The 1990 fluxes (Takahashi et al., 1997) have been corrected to the same gas transfer coefficient and wind field and compared with the 1995 values. The positive values indicate sea-to-air fluxes, and the negative values, the air-to-sea fluxes. (95) and (90) indicate the flux values estimated for the reference years 1995 and 1990 respectively.

Latitudes		Pacific Ocean	Atlantic Indian Ocean	Southern Ocean	Global Ocean	Oceans
		Sea-air Flux in 10^{15} grams Carbon / year				
N. of 50°N	(95)	-0.03	-0.46	-	-	-0.49
	(90)	+0.00	-0.45	-	-	-0.45
50°N-14°N	(95)	-0.48	-0.29	+0.03	-	-0.74
	(90)	-0.39	-0.32	+0.02	-	-0.69
14°N-14°S	(95)	+0.62	+0.12	+0.09	-	+0.83
	(90)	+0.68	+0.10	+0.12	-	+0.90
14°S-50°S	(95)	-0.34	-0.22	-0.60	-	-1.17
	(90)	-0.32	-0.20	-0.38	-	-0.90
S. of 50°S	(95)	-	-	-	-0.63	-0.63
	(90)	-	-	-	-0.31	-0.31
TOTAL	(95)	-0.24	-0.86	-0.47	-0.63	-2.20
	(90)	-0.04	-0.87	-0.23	-0.31	-1.44
% UPTAKE	(95)	11%	39%	22%	29%	100%
	(90)	2%	60%	16%	22%	100%
AREA (10^6 km^2)		151.6	72.7	53.2	31.68	309.12
AREA (%)		49.0%	23.5%	17.2%	10.2%	100%


Ice Edge
-150 -120 -105 -90 -75 -60 -45 -30 -15 0 15 30 45 60 75 90 135
ΔpCO2 Seawater - Air (µatm)
GMT 20:37 Feb 16 1999

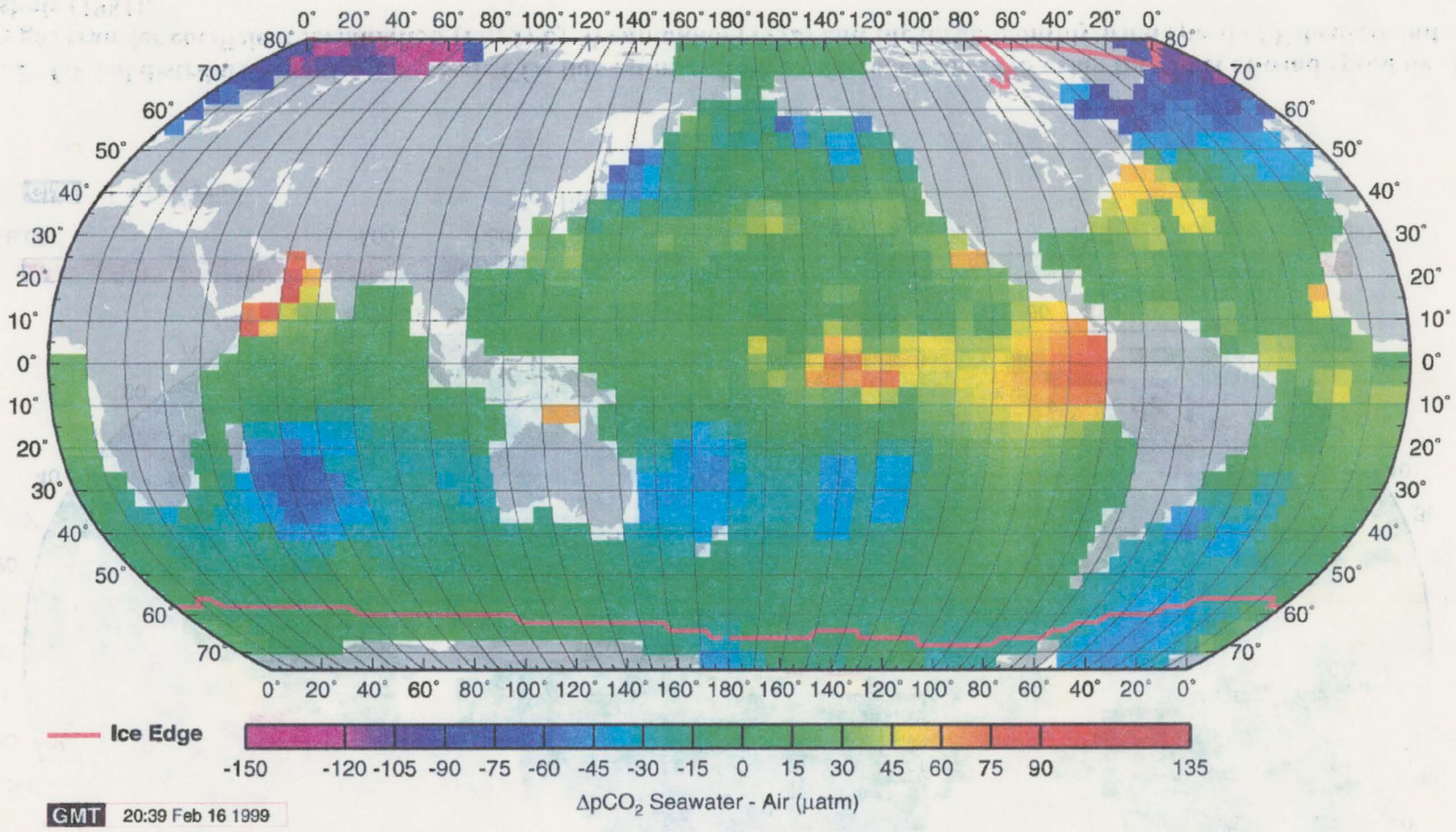


Fig. 1 Global distribution of the sea-air pCO_2 difference estimated for: (A) February, 1995 and (B) August, 1995. Pink lines indicate the edge of ice fields.

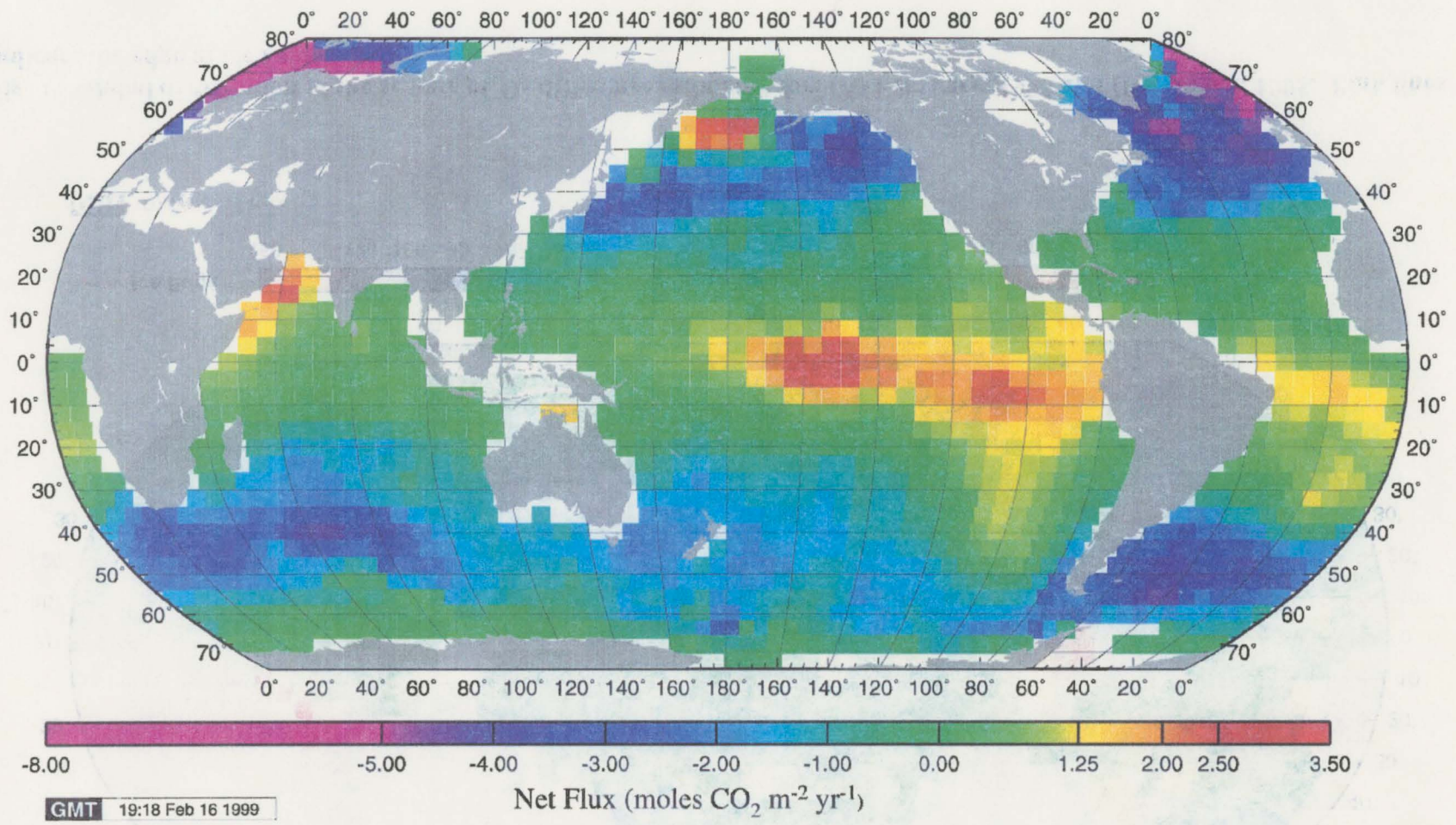


Fig. 2 Global distribution of the net sea-air CO_2 flux estimated for a reference year 1995 using the effect of wind speed on the CO_2 gas transfer coefficient formulation (Eq. 1) by Wanninkhof (1992) and the mean monthly wind speed of Esbensen and Kushnir (1981).