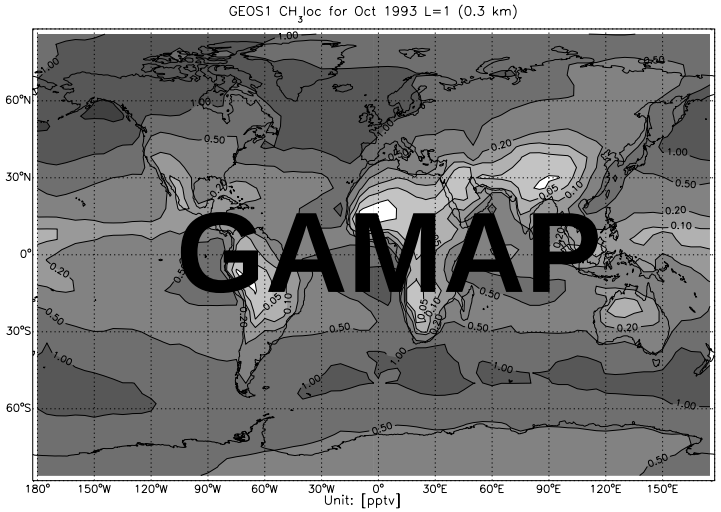

Global Atmospheric Model (output) Analysis Package

Version 1.30

by Martin Schultz and Bob Yantosca
March 1999

Contents

1. Introduction

GAMAP is a collection of programs for easy analysis of CTM (Chemical Transport Model) data. It is written in IDL ("Interactive Data Language" by Research Systems Inc.) and requires version 5.1 of IDL. The package provides an easy-to-use text-based interactive interface to fulfill most basic plotting needs for CTM output, including map plots, vertical cross sections, zonal means, etc. GAMAP 1.30 also has animation capabilities and supports multi-panel plots. But, as the name suggests, GAMAP is more than a simple analysis program: the package provides a collection of high level routines that can be used to efficiently generate own programs for more specialized needs. A list of these routines can be found in Appendix A.

GAMAP handles certain specific file formats that arise from the "diagnostic" format of the GISS II/Harvard CTM. A binary file format has been added to the GEOS model, and it is strongly recommended to migrate to this new format for reasons of file size, speed and simplicity. There are tools available to read in almost any other data file and convert it to a GAMAP record so that it can be displayed with the GAMAP plotting routines. However, this is currently mostly unexplored territory and depends on the willingness of the user to get into the details of GAMAP.

GAMAP 1.30 should be considered a beta version since it has not been fully tested, and there are a couple of known bugs or inconsistencies. However, we thought, the added animation and multi-panel capabilities should be made available as soon as possible - this is why we released this version. Stay tuned for bug fixes and updates!

This document is still under development and was written in a rush. Please report all errors or inconsistencies (as well as GAMAP bugs) to Bob Yantosca (bmy@io.harvard.edu). The future of GAMAP depends on user feedback!

A note to potential users outside our group: GAMAP was developed at Harvard University. It's authors are partly funded by gouvernment money. Therefore, it is probably OK to offer GAMAP as freeware. However, we urge you to keep the copyright notice with each file and clearly mark changes that you make to any subroutine. We would be extremely happy to hear of any improvements you make (and of errors you find and correct). Please understand that we can give only minimal support beyond this document to external users.

2. Installation

GAMAP comes as a gzip'd tar file. If you read this document, you probably already managed to unzip and "untar" it. If not:

gzip -d gamap.*.tar.gz will uncompress the file
tar -xvf gamap*tar will unpack the archive

tar will create two directories in the current working directory: tools and gamap (if these exist already, the corresponding files get probably overwritten). The tools directory contains the complete suite of mgs tools, not all of which are needed for GAMAP. The gamap directory contains all higher level GAMAP files. (At least) two programs were included from other authors: tvimage.pro and undefine.pro, both from David Fanning.

To ensure proper functioning of GAMAP, add the tools and the gamap directory to your IDL !PATH variable in the startup file (usually ~/IDL/idl_startup.pro) as follows:

```
; _after_ the last !PATH = statement add:
!PATH = '~/mydir/tools:~/gamap/tools:' + !PATH
```

(make sure to get all '/' and ':' correct!)

If you are running an IDL session, exit and start again. You should now be able to use GAMAP.

3. Running GAMAP

3.1. Quick start

Before you start GAMAP the first time, you should take a look at the file gamap.defaults in the gamap directory and modify it according to your needs. A full description of this file is given in section XX). The two entries that most likely require modification are MODELNAME and FILEMASK. Possible MODELNAMEs that are currently supported are:

```
'GEOS1         4x5'
'GEOS1         2x2.5'
'GEOS_STRAT    4x5'
'GEOS_STRAT    2x2.5'
'GISS_II       4x5'
'GISS_II_PRIME 4x5'
'FSU           2.8x2.8'
```

You can leave out the resolution information, in this case the first match will be used as default whenever you have to manually select a model type (note that the binary file format contains the model information so that a manual choice is not necessary).

To start GAMAP, first start idl, then type GAMAP at the command line prompt. GAMAP will display a greeting message and after a few seconds ask you to select a data file. You can select all ASCII "punch" files, CTM binary files or GEOS "restart" files; GAMAP will know what to do with them. For the following we will assume that you select an ASCII punch file from one of the models listed above. GAMAP will prompt you for the model that generated the data. The default is marked by a (*) in front and is used when you hit ENTER without typing a number. The default should correspond to the entry in your gamap.defaults file. After you selected a model, GAMAP will read the complete block header information from this file (may take a few seconds) and display the available records. Example:

```
       CATEGORY ILUN TRACERNAME TRC   UNIT      TAU0  DIMENSIONS
  1 :  CH3ISRCE   20     CH3Ioc 271 ng/m2/  74472.00  72 46  1
  2 :  CH3ISRCE   20     CH3Ibb 272 ng/m2/  74472.00  72 46  1
  3 :  CH3ISRCE   20     CH3Iwb 273 ng/m2/  74472.00  72 46  1
...
 81 :  IJ-AVG-$   20     CH3Ibb  72   pptv  78888.00  72 46 NA
 82 :  IJ-AVG-$   20     CH3Iwb  73   pptv  78888.00  72 46 NA
 83 :  IJ-AVG-$   20     CH3Irc  74   pptv  78888.00  72 46 NA
 84 :  IJ-AVG-$   20     CH3Iwl  75   pptv  78888.00  72 46 NA

Enter data block or data block range (default : 1, Q=Quit) >>
```


(you may have to scroll back to see the first entries)
You can now select one or more records to be displayed. Some examples:

```
1            (chooses only first record)
2,5          (select entries 2 and 5)
1-4          (select entries 1 through 4)
1-4,7,10-12
```

If you select more than one record, GAMAP will automatically loop through your selection and start an animation routine (XInteranimate). You can save all "frames" as GIF files or write an MPEG file of your animation. If you have set up a multi-panel environment (see below), GAMAP will fill the page with the selected records (and start a new page when you reach the maximum number of plots you selected). In the multi-panel environment, no animation is possible.

Next you will be queried the range of vertical levels, longitudes, and latitudes you want to plot. To display surface data over the Pacific, you might enter:

```
LEVEL = 1
```

(you can enter one number or two numbers seperated by blank or comma)

```
LONGITUDE = 120 -120
LATITUDE = -88 88
```

GAMAP will then tell you how many dimensions your selected data block has, and it will ask whether you want to perform an average or total operation on either of these dimensions:

```
% Selected data is 2-D [ Longitude,Latitude ].

Do you want to average or total the data?
(0=No averaging, 1=lon, 2=lat, 4=alt, 8=Total, Q=Quit, Default=0)
```

The values displayed as options are binary flags. You can combine them as you wish by adding them together. Example: 3 will average over longitude and latitude, 12 will total over altitude. Be aware that you loose one data dimension for each average or total operation you select. GAMAP can handle any number of dimensions from 0 to 3. If you end up with 0-D (one point), you will get an average or total value over the region that you selected which will be printed on the screen (no plot is produced). A 1-D data set results in a line plot, and a 3-D data set causes GAMAP to start the IDL program slicer3 where you can interactively explore your data cube. The focus of GAMAP, however, are 2-D plots, and you can select from the following options:

```
0 = B/W Contour lines
1 = Colored contour lines
2 = Filled contours
```

```
  3 = Smooth Pixel Plot <---  Default
  4 = Coarse Pixel Plot
```

The default is to make smooth pixel plots which usually look nice on the screen. For printed output, we strongly encourage you to try one of the contour options.

Before you then see the plot, GAMAP asks you about the unit in which the data shall be displayed. If you want to plot in pptv instead of ppbv, simply enter ppbv, and GAMAP will know what to do (at least sometimes ;-). Depending on the options in your gamap.defaults file, you might also be asked if you want to store the resulting image as a GIF file, and if you select Y (or 1), you will be asked for a GIF filename. This filename can contain one of the following variables:
`%N%, %NN%, %NNN%, %NNNN%, or %NNNNN%`
which will be replaced by the current frame number with the respective number of digits.

You should now get one plot or a series of plots displayed. For the first plot, a window will be opened automatically if you haven't done so before. Otherwise GAMAP will plot into the current window (which allows you to display various model results together by changing the current window with WINDOW,N [,options] between subsequent GAMAP calls). When the last plot was made, GAMAP will either ask you whether you want to save this plot as a postscript file (it will be redrawn) or it will start the IDL animation utility XInterAnimate. XInterAnimate will only be started when you selected more than one data record and you did not set-up a multi-panel environment.

This should almost get you going. Feel free to explore the various options, but then read on to learn about the most important GAMAP command line options!

3.2 Important GAMAP command line options

NoFile, FileName

Perhaps the most important keyword to GAMAP is `/NoFile`. If you set this keyword, GAMAP will try to work with the records that you have read previously and not prompt you for another data file. If nothing was read in before, the `/NoFile` option is ignored. With `/NoFile` you have all records available from all files that you opened during this GAMAP session. Without this keyword you will only get

the records displayed that belong to the file you select. Use the `FileName` keyword to specify a data file or override the default filemask in gamap.defaults. `/NoFile` overrides `FileName`, but `FileName` will be used if you have not loaded any data previously.

DiagN, Tracer, Tau0, Date

Use these keywords and parameters to restrict the data records to choose from. `DiagN` is a parameter (the only one) and can be a diagnostic name or number (see diaginfo in section XX). `Tracer` is a tracer number. Some diagnostics use tracer numbers offset by multiples of 100 (see tracerinfo in section XX). GAMAP will only interpret the last two digits and display all diagnostics with this tracer unless you also specify `DiagN`. With `Tau0` and `Date` you can identify individual time steps. `Date` must be a (long) integer value of form YYMMDD (YY > 50 is treated as 1900+YY, YY<50 is treated as 2000+YY). If you use `Date`, the hour is always assumed to be 00 GMT. If you want to specify other hours you must use the `Tau0` keyword. You can pass the result from function nymd2tau() as follows:

```
gamap,tau0=nymd2tau(940301,140000)
```

(This example will select a data record for 1400 GMT on March 3, 1994)
CAUTION: Currently, the conversion from `Date` to `Tau0` assumes the GEOS1 model as default (for the first Tau value). For correct results with the GISS models, you must use the nymd2tau() function as shown above with the `/GISS` keyword.

All of these keywords can be single values or vectors.

YRange, AutoRange

Use these keywords to specify the data range tobe used for colorbars in pixel plots (and for 1-D plots). If you select more than one record, GAMAP will automatically turn AutoRange on unless you are working in a multi-panel environment. YRange must be a 2-element vector with min and max value to plot.

Log, C_Level

The `/Log` flag is no intrinsic GAMAP keyword but simply passed on to ctm_plot (like all other keywords that are not handled explicitely by gamap.pro). It has only effect in 1-D and pixel plots. In contour plots you can use `C_Level` to specify your own contour levels. Default is to use logarithmically spaced levels at 1, 2, 5, 10, etc.

or a linear set of 9 levels if your data falls entirely between two log spacings. A typical example would be (for CO):

```
gamap,/nofile,tracer=4,c_level=findgen(20)*5+40
```

(This creates contour levels at 40, 45, 50, 55, ..., 135)

PS, OutFileName, TimeStamp

The `/PS` switch forces GAMAP to produce postscript output, and `OutFileName` can be used to specify the postscript filename (it overrides the default in gamap.defaults). If `/PS` is set, no plot will be displayed on the screen, but GAMAP will plot directly into the postscript file. `/PS` is automatically turned off when you select multiple records in the single-panel mode, i.e. when GAMAP creates an animation sequence. `/PS` and `OutFileName` are also ignored when the current device is already set to 'PS'. This allows to create "heterogeneous" multi-panel plots with several calls to GAMAP.

`TimeStamp` forces to set or omit a timestamp label on postscript plots and overrides the default setting in gamap.defaults. Use `TimeStamp=1` (or `/TimeStamp`) to force a timestamp label, and `TimeStamp=0` to omit it. GAMAP uses the standard label from close_device.pro which consists of user ID and date.

Do_GIF, Do_MPEG, GIFFileName, MPEGFileName, Frame0

These keywords override the respective settings in gamap.defaults. `Do_GIF` and `Do_MPEG` operate similar to `TimeStamp` (a value of 1 turns it on, a value of 0 turns it off). `GIFFileName` may contain one of the following variables which will be replaced by the current frame number: `%N%, %NN%, %NNN%, %NNNN%, %NNNNN%`. To start with a specific frame, use the `Frame0` keyword.

3.3 Multi-panel plots

GAMAP uses the tools routine multipanel.pro for handling of multipanel plots. It automatically detects whether you turned the multipanel environment on or off and acts accordingly (in single-panel mode, more than one selected record start an animation). To set-up a multipanel environment, type

```
multipanel,nplots=N
```

(where `N` is the number of panels you want on one page) before you run GAMAP.

The multipanel procedure will automatically compute the number of rows and columns from the number of plots so that the result approximates a square. Each time you plotted `N` panels on a page it will be erased. Note that `N` does not need to be a factor number. Alternatively you can explicitely specify the number of rows and columns with the Rows and Cols keywords. For further options (e.g. margin keywords), see the documentation of multipanel.pro (in IDL, type usage,'multipanel').

To return to single-panel mode, type

```
multipanel,/Off
```

3.4 A note on colors

In our computing environment, IDL mostly operates with a shared colortable, i.e. it has a maximum of 256 colors available which it has to share with all other applications. Under some circumstances (e.g. when you run Netscape or FrameMaker), this will cause IDL to either start with very few colors or to use a virtual color map which causes annoying screen flicker when you move the cursor from window to window. GAMAP is somewhat dependend on the tools routine myct.pro as it relies on a few specific drawing colors being loaded at the bottom of each color table. Myct can be used to replace the current colortable with a new one, and it has some features that make it more flexible than the standard IDL loadct command. You can define a Range within the color table that shall be used (`Range` must be a 2-element vector ranging from 0 to 1), or you can revert a colortable with the `/Reverse` keyword. You can also specify the number of colors to be loaded (`NColors`) and the starting point in the color table (`Bottom`). You can even combine several predefined colortables in your current colortable and use them sort of simultaneously (e.g. greyscale plots in the upper half of the page and color plots on the lower half - GAMAP also accepts the `NColors` and `Bottom` keywords).

A working standard has become the EOS-B colortable (#27) with the following options:

```
myct,27,ncolors=!d.n_colors-2,range=[0.1,0.7], sat=0.97,$
value=1.3
```

To create a greyscale colortable with dark colors for high values, use:

```
myct,0,range=[0.2,0.8],/Reverse
```

4. GAMAP as a package

Here, we give a short description of the individual routines that GAMAP is made of. For specific plotting goals, you may want to familiarize yourself with these routines so that you avoid reinventing the wheel over and over. For now we have put most emphasis in getting these routines to work together in gamap.pro, so we encourage careful testing if you exploit them further. After this list, you find a more detailed description of the two "key" routines ctm_plot.pro and ctm_get_data.pro.

4.1 Alphabetical listing of GAMAP routines

ctm_boxsize.pro -- compute grid box areas or volumes

ctm_cleanup.pro -- free memory from previous analyses. Optionally, you can save the ->datainfo structures you have already read, so you don't need to parse the whole file(s) again if you reopen them.

ctm_convert_unit.pro -- "tracer sensitive" unit conversion routine, including conversion from ppbv to ppbC and vice versa. Builds on the more general *convert_unit.pro*.

ctm_diaginfo.pro -- retrieve information about one, several, or all diagnostics that are defined in file diaginfo.dat. After the first call, the information is read from a common block. Use the */FORCE_READING* keyword if you want to reread diaginfo.dat after you made changes.

ctm_get_datablock.pro -- a wrapper routine for *ctm_get_data.pro* that allows you to retrieve data and extract a geographical region. Best to operate on individual data records.

ctm_grid.pro -- construct grid vectors from ->modelinfo structure as returned by *ctm_type.pro*

ctm_index.pro -- convert grid box indices to geographical coordinates and vice versa. Also provides a graphical user-interface to interactively select a grid box with the mouse and return it's coordinates. You can even overlay ctm_index on a previous plot and interactively compute data values or averages.

ctm_label.pro -- create a structure with formatted strings describing the data record and associated model. Then use *replace_token.pro* to substitute "variables" in title strings etc.

ctm_make_datainfo.pro -- Create a datainfo and fileinfo structure for an "external" data array so that it can be used in the GAMAP package. This happens semi-automatically depending on the keywords you specify. The data set must bear some similarity to typical model data sets.

ctm_open_file.pro -- open a CTM output file and parse all header information. Normally, this is done transparently from within *ctm_get_data.pro*, however, you may want to use *ctm_open_file.pro* with the */PRINT* keyword if you detect read errors.

ctm_print_datainfo.pro -- formatted printout of ->datainfo records (e.g. those retrieved via *ctm_get_data.pro*)

ctm_type.pro -- set model specific parameters that are needed to construct the grid vectors

expand_category.pro -- create a string array with individual diagnostic names for multilevel diagnostics (the '$' is replaced by '1', '2', '3', ... 'W').

nymd2tau.pro -- convert time values to model tau values

plotsigma.pro -- produce a plot with sigma levels for various models. You can specify the models to use and the altitude range to plot.

print_totals.pro -- compute and print totals for northern hemisphere, southern hemisphere and globe for individual diagnostics and tracer.

select_model.pro -- a simple user-menu to select a model type and resolution

tau2yymmdd.pro -- convert model tau values to time values in long integer format.

ts_diag.pro -- an older relict to read and plot time series (station) data.

4.2 CTM_PLOT

*(*** this section has not been updated since version 1.10 ***)*
ctm_plot.pro is meant to provide a general plot "wrapper" for the GAMAP package. You can produce map plots, vertical cross sections, and even line plots (although still somewhat limited). 2-D plots can be produced as "pixel" plots or contour plots with various options. For pixel plots, a colorbar is automatically added below the plot. Labelling is provided automatically but easily adjustable to your needs.

General usage:

ctm_plot [,*diagn*] [,*keywords*]

where *diagn* is either a diagnostics number or category name and keywords control the appearance of the plot as well as the data to load. Note that (almost) all of these keywords can also be passed to *gamap.pro*, thereby combining the easy interface of *gamap* with the power of *ctm_plot*.

Controlling of the data to display:

FILENAME -> Name of the punch file to read data from. FILENAME is passed to CTM_GET_DATABLOCK. You can also use a file mask, in which case FILENAME will return the full filename if it is a named variable. If an empty filename is provided, the default search mask from gamap.defaults (see gamap.cmn) will be used. If no filename is given, ctm_plot will try to find the records from data already loaded.

TRACER -> Number of tracer to read from the punch file.

TAU0, /FIRST, /LAST -> time step to be plotted

LON -> If /INDEX is set, LON denotes the CTM longitude index of the box to plot. Otherwise, LON denotes the actual longitude value of that box.

LAT -> If /INDEX is set, LAT denotes the CTM latitude index of the box to plot. Otherwise, LAT denotes the actual latitude value of that box.

LEV -> An index array of sigma levels *OR* a two-element vector specifying the min and max sigma levels to be

included in the plot. Default is [1, GRIDINFO.LMX].

ALTRANGE -> A vector specifying the min and max altitude values to be included in the extracted data block.

PRANGE -> A vector specifying the min and max pressure levels to be included in the extracted data block.

/INDEX -> If set, will interpret LAT, LEV, and LON as CTM indices. If not set, will interpret LAT, LEV, and LON as the actual values of latitude, level, and longitude.

AVERAGE -> If = 0, will not average the data
= 1, will average data longitudinally
= 2, will average data latitudinally
= 4, will average data vertically
These are cumulative (e.g. AVERAGE=3 will average over both lat and lon, and AVERAGE=7 will average over lat, lon, and vertical levels to produce 1 data point).

TOTAL -> If = 0, will not total data
= 1, will total data longitudinally
= 2, will total data latitudinally
= 4, will total data vertically
These are cumulative (e.g. TOTAL=3 will total over both lat and lon, and TOTAL=7 will total over lat, lon, and vertical levels to produce 1 data point).

Control over the appearance of the plot:

YRANGE -> range of y values for color scaling (default: scale each plot seperately with data min and max)

NOERASE -> Do not erase previous plot.

POSITION -> A four-element array of normalized coordinates that specifies the location of map on the plot. POSITION has the same form as the POSITION keyword on a plot. Default is [0.1, 0.05, 0.9, 0.08]. (Passed to TVMAP).

COLOR -> Color index of the map or plot outline and title characters.
Defaults to BLACK (from colors_default.pro).

NCOLORS -> This is the maximum color index that will be used.

BOTTOM -> The lowest color index of the colors to be loaded
used in the color map and color bar.

TITLE -> The title string that is to be placed atop the
map window.

UNIT -> Name of the unit that the DATA array will be converted
to. If not specified, then no unit conversion will be done.

USTR -> Unit string to be plotted to the right of the colorbar.
If not specified, then CTM_PLOT will construct a unit
string based on the value of TRACERINFO.UNIT.

Contour plots:

/CONTOUR -> Will produce a line-contour map instead of the
default color-pixel image map.

/FCONTOUR -> Will produce a filled contour map instead
of the default color-pixel image map. On top of the filled contours
contour lines will be displayed unless keyword NOLINES is set.

C_LEVELS -> Vector containing the contour levels. If not
specified, TVMAP will use quasi-logarithmic levels.

C_COLORS -> Index array of color levels for each line (or
each fill section) of the contour map. If not
specified, TVMAP will select default colors from the
colortable. If a scalar is specified it will be used for all contour levels.
E.g. to produce all black contours, set c_colors=1

C_ANNOTATION -> Vector containing the contour labels.
Default is to use string representations of C_LEVELS.

C_FORMAT -> Format string used in converting C_LEVELS to
the default C_ANNOTATION values. Default is '(f8.1)'.

C_LABELS -> Specifies which contour levels should be labeled.
By default, every other contour level is labeled. C_LABELS
allows you to override this default and explicitly
specify the levels to label. This parameter is a vector,
converted to integer type if necessary. If the LEVELS
keyword is specified, the elements of C_LABELS
correspond directly to the levels specified, otherwise,
they correspond to the default levels chosen by the
CONTOUR procedure. Setting an element of the vector to
zero causes that contour label to not be labeled. A
nonzero value forces labeling.
NOTE: If C_LABELS is given as a scalar, then it will be
expanded to a vector with all elements having the same value.

/NOLINES -> Will suppress overplotting a filled-contour map
with lines. Default is to overlay filled-contour maps
with lines.

/NOLABELS -> Will suppress printing contour labels on both
line-contour and filled-contour maps.

OVERLAYCOLOR -> Color of the solid lines that will be
overlaid atop a filled contour map. Default is BLACK.

Pixel plots

/SAMPLE -> Will cause REBIN (in TVMAP) to use nearest-
neighbor sampling rather than bilinear interpolation.

/LOG -> Will create a color-pixel plot with logarithmic
scaling. /LOG has no effect on line-contour or
filled-contour plots, since the default contour levels
are quasi-logarithmic.

Maps:

CCOLOR -> The color index of the continent outline or fill region. Default is BLACK (from colors_default.pro).

CFILL -> Value passed to FILL_CONTINENTS keyword of MAP_CONTINENTS. If CFILL=1 then will fill continents with a solid color (as specified in CCOLOR above). If CFILL=2 then will fill continents with hatching.

/NOCONTINENTS -> If set, will suppress adding continent lines to the map. Default is to call MAP_CONTINENTS to plot continent outlines or filled boundaries.

/NOGRID -> If set, will suppress printing of grid lines. Default is to call MAP_GRID to overlay grid lines.

GCOLOR -> The color index of the grid lines. Default is BLACK (from colors_default.pro)

/NOISOTROPIC -> Will suppress plotting of an isotropic map (i.e. one with the same X and Y scale). Default is to print an isotropic map.

MPARAM -> A 3 element vector containing values for [P0Lat, P0Lon, Rot]. Default is [0, 0, 0] unless you look at Pacific data.

You can also specify a different projection, but there's no warranty at all!! Do not specify LIMIT, because this is set automatically from the LON and LAT keywords!

Controlling the colorbar:

/NOCBAR -> If set, will not plot the colorbar below the map in the position specified by CBPOSITION. Default is to plot a colorbar.

CBCOLOR -> Color index of the colorbar outline and characters. Defaults to BLACK (from colors_default.pro).

CBPOSITION -> A four-element array of normalized coordinates that specifies the location of the colorbar. BARPOSITION has the same form as the POSITION keyword on a plot. Default is [0.1, 0.05, 0.9, 0.08].

CBUNIT -> Passes the Unit string to COLORBAR, which will be plotted to the right of the color bar.

CBFORMAT -> format to use in call to colorbar. Default is I12 if abs(max(data)) < 1e4, else e12.2 (strings get trimmed)

A note on titles:

If you specify a title string with the title keyword, you can use certain tokens that will be replaced by their values automatically (see ctm_label.pro). The most fundamental are:

%MODEL% will be replaced by model name
%RES% will be replaced by model resolution
%TRACERNAME% will be replaced by the (short) name of the tracer
%DATE% will be replaced by an auto-formatted date string indicating the time span of the simulation
%YMD0%, %YMD1% will be replaced by start and end date in the format 'yymmdd'
%LAT% will be replaced by a latitude value or range
%LON% will be replaced by a longitude value or range
%ALT%, %LEV%, %PRS% will be replaced by an altitude, level or pressure value or range, resp.

4.3 CTM_GET_DATA

(This is a copy of the file ctm_examples.pro *which contains all necessary steps)*
*(*** This section has no tbeen updated since version 1.10 ***)*

```
; $Id: ctm_examples.pro,v 1.2 1998/10/23 02:40:55 mgs Exp mgs $


pro ctm_examples



    ; Quick an dirty demonstration of new CTM_* routines

    ; mgs, 20 Aug 1998
    ; mgs, 22 Oct 1998: adapted for new use of CTM_GET_DATA
    ;                   some more comments
    ; mgs, 26 Oct 1998: attached a few more comments about extended
    ;                   use of ctm_get_data at end
    ; mgs, 18 Nov 1998: added call to ctm_make_datainfo

    ; NOTE: Images will not be completely correctly displayed
    ; because in these examples no use is made of
    ; grid information

    ; Also, the current version of the routines does not
    ; perform any unit conversion !


    ; set color table
    myct,27,sat=0.97,value=1.3,bottom=20

    ; ============================================================
    ; Load data from a user selected file and plot surface
    ; concentrations (LEVEL 1 in DIAG 45) as image map
    ; Requires the “correct” category name for DIAG45 in
    ; diaginfo.dat
    ; ============================================================

print
print,’======   EXAMPLE   1   ================’
print,’Display Ox surface data...’
print

    DIAGN  = 45             ; diagnostic number. Could also be
                            ; a name, e.g. ‘IJ-AVG-$’

    TRACER = 2              ; Ox

    filename = ‘’           ; filename must be initialized to
                            ; force opening of a file when running
                            ; this program repeatedly!
```



```
                            ; If filename is UNDEFINED, CTM_GET_DATA
                            ; will use whatever data is available
                            ; in the global FileInfo and DataInfo
                            ; structures and only open a file on
                            ; first calling.


CTM_GET_DATA,DataInfo,diagn,  $
             file=filename,tracer=tracer,/FIRST
                            ; The user will be asked to select a file
                            ; with PICKFILE. Note that the file will not be
                            ; parsed again, and the data not re-read
                            ; if the same file is selected twice.
                            ;
                            ; /FIRST selects the first time value in
                            ; the file.
                            ;
                            ; DATAINFO will contain all records for OX
                            ; concentrations in that file.

if (n_elements(DATAINFO) eq 0) then message,’Sorry: No data loaded.’

print,strcompress(n_elements(DATAINFO)),  $
      ‘ records loaded from file ‘,filename
                            ; filename contains name of actually selected
                            ; file

data = *(DataInfo[0].data)
                            ; Retrieve data from first record
                            ; (There should only be one anyhow because
                            ; we selected a specific tracer and the first
                            ; time step)
                            ; NOTE: This is a 3D data cube although
                            ; the diagnostic was saved in individual
                            ; levels (at least in ASCII files)

srfdata = reform(data(*,*,0))
                            ; extract surface data

print,’Min and max: ‘,min(srfdata,max=m),m

img = bytscl(srfdata,min=0.,max=m,top=!d.n_colors-20)+20
                            ; convert data to color image

window,0
tvimage,img,position=p,/keep_aspect
                            ; display data

map_set,0,0,/noborder,/noerase,pos=p
                            ; prepare map

map_continents,color=1   ; show continents
map_grid,color=1         ; and grid lines
```

```
    xyouts,0.5,0.95,'Ox surface average (diag 45)',/norm,  $
           color=1,charsize=1.4,align=0.5


    ; ============================================================
    ; Now load 3D cube for a different tracer and display
    ; zonal means  (Again: Grid information is *not* used
    ; in the display.
    ; ============================================================


print
print,'======   EXAMPLE   2   ================'
print,'Display zonal means for Ethane ...'
print

    TRACER = 21             ; Ethane

    CTM_GET_DATA,DataInfo,diagn,filename=filename,  $
                 tracer=tracer,/FIRST
                            ; Note that we pass filename as variable
                            ; which now contains a full qualifier.
                            ; Therefore, we won't be asked to pick a
                            ; file again.
                            ; (If we reset filename to '' or anything
                            ; containing wildchards will we be asked
                            ; to select a file again)
                            ; NOTE: When calling CTM_EXAMPLES the
                            ; first time, you can omit FILENAME
                            ; completely. It will then use the datainfo
                            ; records that were loaded before.

    if (n_elements(DATAINFO) eq 0) then message,'Sorry: No data loaded.'

    datb = *(DataInfo[0].data)
                            ; Retrieve 3D data from first record

    zmeans = total(datb(*,*,*),1)/DataInfo[0].dim[0]
                            ; compute zonal means

    print,'Min and max: ',min(zmeans,max=m),m
help,datainfo,datb,zmeans
help,datainfo[0],/stru

    img = bytscl(zmeans,min=0.,max=m,top=!d.n_colors-20)+20

    window,1
    tvimage,img,position=p   ; display image


    xyouts,0.5,0.95,'Zonal means for Ethane',/norm,  $
           color=1,charsize=1.4,align=0.5
```



```
    ; ============================================================
    ; Now, let's compute the difference in ozone between the last
    ; and first time step of a simulation. NOTE: we will ask the
    ; user to pick a file again, and CTM_GET_DATA will only "see"
    ; the data from the newly selected file.
    ; ============================================================

print
print,'======   EXAMPLE   3   ================'
print,'Display Ox difference between 2 time steps ...'
print

    TRACER = 2              ; Ox again

    CTM_GET_DATA,DataInfo,diagn,  $
                 filename='',tracer=tracer, $
                 tau0=tau0
                            ; The (empty) string constant in the
                            ; filename parameter ensures the
                            ; file selection.
                            ; Since no selection is made for tau
                            ; (tau0 is un-initialized)
                            ; we will return all time steps that were
                            ; recorded in that file

    if (n_elements(DATAINFO) eq 0) then message,'Sorry: No data loaded.'

    mintau = min(DataInfo.tau0,max=maxtau)
    first = where(DataInfo.tau0 eq mintau)
    last  = where(DataInfo.tau0 eq maxtau)
                            ; get index to index for first and
                            ; last time step

    if (first[0] eq last[0]) then print,'% First and last time step identical!'

    data = *(DataInfo[first].data)
    datb = *(DataInfo[last].data)
                            ; Retrieve 3D data

    diff = datb-data
                            ; compute difference. In a more general
                            ; application, we must first test the
                            ; dimensions and grid compatibility

    slice = reform(diff[*,34,*])
                            ; extract a slice along a northern mid lat.
                            ; (4x5 models)

    mi = min(slice,max=m)
    print,'Min and max: ',mi,m

    absmax = abs(mi) > abs(m)
                            ; compute scaling factor for image
```

```
    img = bytscl(slice,min=-absmax,max=absmax,  $
                top=!d.n_colors-20)+20

    window,2
    tvimage,img,position=p   ; display image

    xyouts,0.5,0.95,'Ox difference last-first time step - latitudinal cut', $
           /norm,color=1,charsize=1.4,align=0.5



    ; ============================================================
    ; add difference as record in datainfo
    ; ============================================================

    dummy = ctm_make_datainfo(diff,diagn='Ox difference',tracer=2, $
             tau0=DataInfo[last].tau1,tau1=DataInfo[first].tau0, $
             unit='ppbv')


    ; ============================================================
    ; And here is an example for a GISS_II_PRIME file
    ; (including Prashnat's blanks for MATLAB)
    ; Let's display PORL-L=6, but note it's extracted from the
    ; 3D cube!
    ; ============================================================

print
print,'======   EXAMPLE   4   ================'
print,'Display Loretta''s production/loss diagnostics ...'
print

    DIAGN  = 'PORL-L=$'      ; Use category name

    TRACER = 2               ; whatever it is

    CTM_GET_DATA,DataInfo,diagn,  $
                 filename='~ppm/terra/Runs/ljmmod/ctm.pch',tracer=tracer
                             ; All tau values will be returned
                             ; Since the file is specified fully,
                             ; there will be no file selection

    if (n_elements(DATAINFO) eq 0) then message,'Sorry: No data loaded.'


    data = *(DataInfo[0].data)
                             ; Retrieve data from first record
                             ; (see first example)

    levdata = reform(data(*,*,5))
                             ; extract data for layer 6
                             ; NOTE: IDL always starts with 0 !!

    print,'Min and max: ',min(levdata,max=m),m
```



```
    img = bytscl(levdata,min=0.,max=m,top=!d.n_colors-20)+20
                             ; convert data to color image

    window,3
    tvimage,img,position=p,/keep_aspect
                             ; display data

    map_set,0,0,/noborder,/noerase,pos=p

    map_continents,color=1   ; show continents
    map_grid,color=1         ; and grid lines

    xyouts,0.5,0.95,'PORL-L=6 (diag 99, Loretta special)',/norm,  $
           color=1,charsize=1.4,align=0.5




    ; ============================================================
    ; Now let us summarize and check what is stored in memory
    ; ============================================================


@gamap_cmn.pro                   ; include global common block


    ; File information records
    FileInfo = *pGlobalFileInfo

    for i=0,n_elements(FileInfo)-1 do $
        print,FileInfo[i].ilun,FileInfo[i].filename,  $
              FileInfo[i].modelinfo.name,format='(I5,A45,A20)'

    print

    ; data information records
    DataInfo = *pGlobalDataInfo

    ; extract unique category names
    all_cat = DataInfo.category
    all_cat = all_cat( uniq(all_cat,sort(all_cat)) )
                             ; get a list of all categories

    print,'loaded data by category:'
    print,'  CATEGORY   TRACER     TAU0     TAU1   DIMENSIONS'

    for i=0,n_elements(all_cat)-1 do begin
       all_loaded = ctm_doselect_data(all_cat[i],DataInfo)

       if (all_loaded[0] ge 0) then $
          for j=0,n_elements(all_loaded)-1 do begin
             tmp = DataInfo[all_loaded[j]]
             print,tmp.category,tmp.tracer,tmp.tau0,tmp.tau1,tmp.dim, $
```

```
                format='(A10,3I9,4I5)'
        endfor

    endfor


    return


    ; ================================================================
    ; EXTENDED USE OF CTM_GET_DATA
    ; (section added Oct 26, 1998)
    ; ================================================================

    ; Read data for tracer N for all diagnostics
    ; (use diagn=0 !)

    ctm_get_data,datainfo,diagn,file=filename,tracer=N  ; [tau0=...]


    ; Read data for SRF-AVRG and HZ$-AVRG in one step
    ; (NOTE that this will result in two datainfo records! The two
    ; diagnostics are not combined unless you rename SRF-AVRG to HZ1-AVRG!)

    ctm_get_data,datainfo,['SRF-AVRG','HZ$-AVRG'],file=filename,tracer=N


    ; Read all data for tracer 2 from 2nd file
    ; (logical units (usually) start with 20. Use help,/files to check!)

    ctm_get_data,datainfo,0,ilun=21,tracer=2


end
```


The use of CTM_GET_DATA (cont'd)

(This is a copy of the file header from ctm_get_data.pro, i.e. the output of usage,'ctm_get_data')

```
; CALLING SEQUENCE:
;       CTM_GET_DATA,DATAINFO [,DIAGN] [,keywords]
;
; INPUTS:
;       DIAGN -> A diagnostic number or category name (see
;            (CTM_DIAGINFO). A value of 0 (or an empty string)
;            prompts processing of all available diagnostics.
;            DIAGN can also be an array of diagnostic numbers or
;            category names.
;
; KEYWORD PARAMETERS:
;       FILENAME -> (optional) If FILENAME is a fully qualified file path
;            the specified file is opened without user interaction.
;            If filename is empty or contains wildchards (*,?), a
;            pickfile dialog will be displayed.
;            If FILENAME is a named variable it will contain the full
;            file path upon return so that a subsequent call to
;            CTM_GET_DATA with the same FILENAME argument will not prompt
;            another file selection dialog.
;               If the FILENAME keyword is present, CTM_GET_DATA
;            will restrict it's scope to records from the selected
;            is file (even if FILENAME contains an empty string, it will
;            restrict the scope of the search!).
;               If the file is found in the global FILEINFO structure or
;            the USE_FILEINFO structure (i.e. it has been opened
;            previously), then it will not be parsed again; instead the
;            data records are returned from memory.
;               The data itself is loaded transparently via
;            CTM_RETRIEVE_DATA.
;
;       ILUN -> An optional value or array of logical unit numbers. If
;            given, the search is restricted to data from the specified
;            files. Default is to use all files (unless the FILENAME
;            keyword is present). If an undefined variable
;            is passed into ILUN, information about all accessible files
;            in the global FILEINFO structure (or USE_FILEINFO) is returned.
;
;       TRACER -> A tracer ID number or a list of those. If given, the
;            search is restricted to those tracers. Default is to use all
;            tracers. If an undefined variable is passed into TRACER,
;            and one specific diagnostics is requested with DIAGN,
;            information about all accessible tracers in the global
;            DATAINFO structure or USE_DATAINFO structure or the
;            DATAINFO structure associated with a specific file is returned.
;
;       TAU0 -> A time value or list of values to restrict the search.
;            Default handling as with ILUN or TRACER. TAU0 superseeds
;            /FIRST, /LAST or TAURANGE.
```

```
;
;       TAURANGE -> A 2-element vector containing the first and last tau0
;            value to look for.
;
;       /FIRST, /LAST -> extract first or last time step that is stored for
;            the selected diagnostics, ilun, and tracer. Only one can be
;            be active at a time. /LAST superseeds /FIRST.
;
;       INDEX -> A named variable that will contain the indices of the
;            records in USE_DATAINFO that match the selection criteria.
;            You can test INDEX[0] ge 0 in order to see if CTM_GET_DATA has
;            been successful although it is now recommended to test for
;            n_elements(DATAINFO) eq 0.
;               The INDEX keyword is useful if you want to change the
;            information contained in the selected DATAINFO structures
;            globally.
;
;       USE_FILEINFO -> (optional) If provided, CTM_GET_DATA will
;            restrict its search to only the files that are
;            contained in USE_FILEINFO which must be a FILEINFO
;            structure array. Default is to use the global information
;            (see gamap_cmn.pro).
;            If an undefined named variable is provided in USE_FILEINFO,
;            it will either contain the global FILEINFO structure array
;            or the FILEINFO record of the specified file.
;            USE_FILEINFO must contain entries for all logical unit numbers
;            that are used in USE_DATAINFO.
;
;       USE_DATAINFO -> (optional) Restrict search to records contained
;            in USE_DATAINFO which must be a DATAINFO structure array.
;            If an undefined named variable is provided in USE_DATAINFO,
;            it will either contain the global DATAINFO structure array
;            or all DATAINFO records of the specified file.
;            See also USE_FILEINFO.
;
;       /INTERNAL_USE -> Set this keyword if you want to prevent a call
;            to CTM_OPEN_FILE, but instead abort in case of undefined
;            (global) FILEINFO or DATAINFO structures.
;
; OUTPUTS:
;       DATAINFO -> An array of DATAINFO records that match the selected
;            criteria. You can then simply loop over
;            0..n_elements(DATAINFO)-1 to access all data records and
;            extract the data as *(DATAINFO[i].data).
;            DATAINFO will be undefined if no records are found!!
;            Always test for  IF (n_elements(DATAINFO) eq 0) ... !
;            NOTE: Alternatively you can return the INDEX to the selected
;            data records in the global (or USE_) datainfo structure array
;            with the INDEX keyword. This may in some cases eliminate the
;            need to make a local copy of the selected DATAINFO records.
;
; EXAMPLE:
;       See CTM_EXAMPLES
```


5. Data formats

5.1 ASCII “punch” files:

title line

GLOBAL-RUN--4X5+SOM-PFILT--CTM version 1.0 (3/93 Prather) 24 72 46 20 0 0 1985

each data block consists of 1 header line + data lines

IJ-AVG-1 276 2 75960 76680 1.000E-10 72 46 0

category nlines trc tau0 tau1 scale nx ny nz

288.86288.86288.90288.90288.92288.96288.96288.98288.99289.00289.01289.01
289.01289.02289.01289.00289.01289.02289.00289.01289.02289.02289.03289.04

format: mostly 12F6.2 (last line may be incomplete!)

5.2 Binary output files

These were designed to interact smoothly with GAMAP. They also load much faster and save about 1/3 of disk space.

General header:

ftype: A file identifier string (length 40) ‘CTM binary ...’

toptitle: as title line in ASCII files (length 80)

modelname, modelres: name of the model (e.g. ‘GEOS1’) and resolution as lon, lat (character*20 and 2 real*4)

Datablock header:

category, tracer, tau0, tau1, skip: The name of the diagnostics (identical to ASCII), the tracer number, time information and length of following data block in bytes (character*40, integer*4, 2 real*8, integer*4)

dimensions (6 integer*4: NI, NJ, NL, I0, J0, L0)

data (as real*4)

NOTE: While “multilevel” diagnostics are stored as individual layers in ASCII files, they are stored as 3D block in binary files. Category name e.g. ‘*IJ-AVG-$*’

5.3 The FILEINFO structure

Contains information about a file that has been accessed in a GAMAP session. All files are stored in a global common block which is accessible via the include statement *@gamap_cmn*

filename (string)
ilun (long integer): logical unit number. This field links FILEINFO and DATAINFO structures
filetype (integer): 0=ASCII "punch" file, 1=binary file, 2=(GEOS) restart file; others still open
status (integer): should be 1 if everything is correct, 0 indicates error (not thoroughly implemented though)
toptitle (string): Holds the title line from the model output file (ASCII or binary)
modelinfo (structure): Contains information about the model, so that grid can be computed (see below)
gridinfo (pointer): points to a structure that contains the horizontal and vertical grid (see below)

Use function create3dfstru to create new fileinfo structure(s).

5.4 The DATAINFO structure

Contains information about each data record that is in a model file. If a new file is opened, the complete header information is parsed once and the location of the individual data blocks is saved. Subsequent access is therefore really fast.

ilun (long integer): logical unit number. This field links FILEINFO and DATAINFO structures
filepos (long integer): saves position of associated data record in the file
category (string): The diagnostics name (see data formats)
tracer (integer): A number identifying a tracer. This number corresponds to the CTM tracers but can be offset by multiples of 100: e.g. NOX-flux is 101. (information on all tracers is stored in tracerinfo.dat)
tracername (string): The (full) name of the tracer as stored in tracerinfo.dat
tau0, tau1 (long integers): The tau values indicating the model time. In general, GAMAP uses tau0 rather than tau1 to identify data records.

scale (real): A scale factor that is applied to the data. Upon reading, the data is scaled automatically (a) to accomodate the scale factor stored in the ASCII header, (b) to convert the data to the default unit given in tracerinfo.dat. Therefore, *scale* will usually have a value of 1.
unit (string): The physical unit of the data (e.g. 'ppbv'). Units can be converted with *ctm_convert_unit* (based on *convert_unit*)
format (string): A format string describing the ASCII format of the data (e.g. '(12F6.2)'). Takes the value 'binary' for binary data files. If teh data i sfree-format, leave this field empty
status (integer): 0 indicates data not read, 1 indicates data read. This field is used in ctm_get_data to specifically select (un)read data records. Convention: 2 indicates either loaded or unloaded data.
dim (4xinteger): Dimensions of the data array: dim[0]=NI, dim[1]=NJ, dim[2]=NL, dim[3]=ntimesteps. This is important to correctly assign a 2D field to LON-LAT or LON-ALT or LAT-ALT. The 4th dimension is not fully implemented. If a dimension is not present, the respective field can be either 0 or 1.
data (pointer): points to a 2D or 3D (or even 1D) float array. Access data as data=*(datainfo[index].data)

5.5 The MODELINFO structure

This structure holds basic information about the model so that a grid can be computed.
General usage:
    minfo = ctm_type(<model name> [options])
Example:
    minfo = ctm_type('GEOS-1', resolution=4, psurf=10013.)

name (string): the name of the model (or data set). Current options are GEOS-1, GEOS_START, GISS_II, GISS_II_PRIME (aka II_PRIME), FSU
family (string): a model family. GEOS, GISS, FSU
online (integer): a flag indicating whether the model is driven with online or offline winds (not used)
nlayers (integer): the maximum number of vertical layers
ntrop (integer): the number of layers in the troposphere (i.e. with chemistry)

ptop, psurf (float): pressure at model top and surface (used to convert sigma layers into pressure)
resolution (2xfloat): DI x DJ
halfpolar (integer): flag that indicates whether polar boxes are half size or not
center180 (integer): flag that indicates whether grid boxes are centered (or edged) on 180 degrees
fullchem (integer): flag that indicates whether full or reduced chemistry set was used (affects tracer numbers, but is not really supported!)

NOTE that most parameters are automatically set to the correct values in function ctm_type, so that you only need to provide the model name and possibly resolution.

5.6 The GRIDINFO structure

This structure holds the actual grid for the data set (both horizontal and vertical). It is created from the information stored in MODELINFO, but may be frequently re-generated e.g to accomodate for different surface pressures.

It is generated with a call to *ctm_grid*:
ginfo = ctm_grid(minfo [,psurf=newpsurf])

IMX, JMX, DI, DJ (integers): number of grid boxes in LON and LAT, width of grid boxes
XEDGE, YEDGE, XMID, YMID (float arrays of variable size): edge and center coordinates for longitudes and latitudes

When GRIDINFO contains vertical information (i.e. almost always), the following fields are added:
LMX (integer): the number of vertical layers
SIGEDGE, SIGMID (float arrays): edge and center sigma coordinates
PEDGE, PMID (float arrays): edge and center pressures (computed from sigma values using ptop and psurf)
ZEDGE, ZMID (float arrays): edge and center altitudes (computed from pressure values using a fit to the US standard atmosphere)

NOTE: The computation of pressures and altitudes may at some point be improved to allow for different surface pressures for individual grid boxes. However, this can be pretty complicated in terms of averaging ...

5.7 The TRACERINFO structure

This structure contains information about the various quantities stored in the model output. Tracers are indexed by number, and the number is usually identical to the tracer number used in the model, but it may be offset by a multiple of 100 to distinguish between different diagnostics.

This structure will be returned from procedure *ctm_tracerinfo*. The information about all tracers is stored in the file *tracerinfo.dat*. This file is only read once, however, it can be reloaded with *ctm_tracerinfo,/force_reading*. You can pass either a tracer name or number to *ctm_tracerinfo*.

name, fullname (string): Two variants of the tracer name. The first one will usually be used for plotting etc., while the second may be more explanatory.
mwt (float): mole weight of tracer in g. Can be used to convert from mixing ratio to mass and vice versa.
fulli, smalli (integer): tracer number (index) in the full chemistry (default) and reduced chemistry mode
molc (float): number of carbon atoms (used to convert ppbv to ppbC)
scale (float): scale factor that must be applied to convert output to standard unit given in tracerinfo.unit. NOTE: This relies on consistent output within all models!
unit (string): standard unit that shall be used for display of data for this tracer. Units can be converted with ctm_convert_unit (still in progress...)

IMPORTANT: In order to accomplish the use of GAMAP for all models, tracer numbers must follow a common scheme. If new tracers are added to a model, they should be given unique new numbers so that they do not interfere with the current set. Also, it would be desirable to change tracer numbers in the reduced chemistry

mode to be consistent with those from the full chemistry mode (This only affects the output in the "punch" file or binary file, not the number that is used internally in the CTM)

5.8 The DIAGINFO structure

This structure holds information about the individual diagnostics from all models. It is be returned from procedure *ctm_diaginfo*. The information about all diagnostics is stored in the file *diaginfo.dat*. This file is only read once, however, it can be reloaded with *ctm_diaginfo,/force_reading*. You can pass either a category name or diagnostic number to *ctm_tracerinfo*. Note, however, that category names are unique, while numbers may be used repeatedly (e.g. 45 for IJ-AVG-$, HZ$-AVRG).

index (integer): The diagnostic number
category (string): The diagnostic name
type (string): A code describing the "coordinates" of the diagnostics (e.g. 'I-J', 'J-L', etc.). This field is not really used.
maxtracer (integer): This field is used for "source-type" diagnostics where the tracer number is misused as a general identifier e.g. for lightning NOx. For all "good" diagnostics, this number is 1.
offset (integer): An offset (multiple of 100) that is added to the tracer number as stored in the output file to uniquely identify the quantity

As with TRACERINFO, an effort should be made to keep diagnostic categories and numbers consistent between the various models.

6. Getting started with IDL

If you haven't used IDL before, follow the instructions in ~mgs/IDL/README

Otherwise: take a look at ~mgs/IDL/idl_startup.pro,

6.1 A few features and pecularities

- IDL is very flexible in handling different data types
  Example: a=2.0 & help,sqrt(a) & a=2.0d0 & help,sqrt(a)
  (use caution with big integers! *Example:* tau=30000 & help,tau+20000
  and with divisions! *Example:* print,5/2)

- IDL array indices always start with 0
  (i.e. grid box 68,23 (FORTRAN) is 67,22 in IDL)

6.2 Some general helper tools

usage,'<routine_name>': prints information from file header about how to use a specific routine
open_device: opens a file for postscript output or a (new) window on the screen
close_device: closes postscript file (mandatory!) or optionally a window
open_file: assigns a filename to a logical unit and opens the file. You can use wild-cards (*,?) -> PICKFILE dialog box
percentiles: compute arbitrary percentiles of data
colorbar: add a colorbar to a plot
legend: add a legend to a plot
myct: define your colortable

These and many others can be found in your tools directory after you install GAMAP. Most of the tools are also posted on Martin's IDL website (*http://www-as.harvard.edu/people/staff/mgs/idl/*).

There is also a really good book on IDL available ("IDL Programming Techniques" by D. Fanning).