MPTRAC

Generated by Doxygen 1.8.11

Contents

1 Main Page

Massive-Parallel Trajectory Calculations (MPTRAC) is a Lagrangian particle dispersion model for the troposphere and stratosphere.This reference manual provides information on the algorithms and data structures used in the code. Further information can be found at: http://www.fz-juelich.de/ias/jsc/mptrac

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 atm_t Struct Reference

Atmospheric data.

```
#include <libtrac.h>
```

Data Fields

- int np
  Number of air pacels.
- double time [NP]
  Time [s].
- double p [NP]
  Pressure [hPa].
- double lon [NP]
  Longitude [deg].
- double lat [NP]
  Latitude [deg].
- double q [NQ][NP]
  Quantitiy data (for various, user-defined attributes).
- double up [NP]
  Zonal wind perturbation [m/s].
- double vp [NP]
  Meridional wind perturbation [m/s].
- double wp [NP]
  Vertical velocity perturbation [hPa/s].

4.1.1 Detailed Description

Atmospheric data.

Definition at line 444 of file libtrac.h.

4.1.2 Field Documentation

4.1.2.1 int atm_t::np

Number of air pacels.

Definition at line 447 of file libtrac.h.

4.1.2.2 double atm_t::time[NP]

Time [s].

Definition at line 450 of file libtrac.h.

4.1.2.3 double atm_t::p[NP]

Pressure [hPa].

Definition at line 453 of file libtrac.h.

4.1.2.4 double atm_t::lon[NP]

Longitude [deg].

Definition at line 456 of file libtrac.h.

4.1.2.5 double atm_t::lat[NP]

Latitude [deg].

Definition at line 459 of file libtrac.h.

4.1.2.6 double atm_t::q[NQ][NP]

Quantitiy data (for various, user-defined attributes).

Definition at line 462 of file libtrac.h.

4.1.2.7 double atm_t::up[NP]

Zonal wind perturbation [m/s].

Definition at line 465 of file libtrac.h.

4.1.2.8 double atm_t::vp[NP]

Meridional wind perturbation [m/s].

Definition at line 468 of file libtrac.h.

4.1.2.9 double atm_t::wp[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 471 of file libtrac.h.

The documentation for this struct was generated from the following file:

- libtrac.h

4.2 ctl_t Struct Reference

Control parameters.

```
#include <libtrac.h>
```

Data Fields

- int nq
  Number of quantities.
- char qnt_name [NQ][LEN]
  Quantity names.
- char qnt_unit [NQ][LEN]
  Quantity units.
- char qnt_format [NQ][LEN]
  Quantity output format.
- int qnt_ens
  Quantity array index for ensemble IDs.
- int qnt_m
  Quantity array index for mass.
- int qnt_rho
  Quantity array index for particle density.
- int qnt_r
  Quantity array index for particle radius.
- int qnt_ps
  Quantity array index for surface pressure.
- int qnt_p
  Quantity array index for pressure.
- int qnt_t
  Quantity array index for temperature.
- int qnt_u
  Quantity array index for zonal wind.
- int qnt_v
  Quantity array index for meridional wind.

- int qnt_w
  Quantity array index for vertical velocity.
- int qnt_h2o
  Quantity array index for water vapor vmr.
- int qnt_o3
  Quantity array index for ozone vmr.
- int qnt_theta
  Quantity array index for potential temperature.
- int qnt_pv
  Quantity array index for potential vorticity.
- int qnt_tice
  Quantity array index for T_ice.
- int qnt_tsts
  Quantity array index for T_STS.
- int qnt_tnat
  Quantity array index for T_NAT.
- int qnt_stat
  Quantity array index for station flag.
- int direction
  Direction flag (1=forward calculation, -1=backward calculation).
- double t_start
  Start time of simulation [s].
- double t_stop
  Stop time of simulation [s].
- double dt_mod
  Time step of simulation [s].
- double dt_met
  Time step of meteorological data [s].
- int met_np
  Number of target pressure levels.
- double met_p [EP]
  Target pressure levels [hPa].
- int isosurf
  Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).
- char balloon [LEN]
  Balloon position filename.
- double turb_dx_trop
  Horizontal turbulent diffusion coefficient (troposphere) [m^2/s].
- double turb_dx_strat
  Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s].
- double turb_dz_trop
  Vertical turbulent diffusion coefficient (troposphere) [m^2/s].
- double turb_dz_strat
  Vertical turbulent diffusion coefficient (stratosphere) [m^2/s].
- double turb_meso
  Scaling factor for mesoscale wind fluctuations.
- double tdec_trop
  Life time of particles (troposphere) [s].
- double tdec_strat
  Life time of particles (stratosphere) [s].
- double psc_h2o

H2O volume mixing ratio for PSC analysis.

- double psc_hno3

  HNO3 volume mixing ratio for PSC analysis.
- char atm_basename [LEN]

  Basename of atmospheric data files.
- char atm_gpfile [LEN]

  Gnuplot file for atmospheric data.
- double atm_dt_out

  Time step for atmospheric data output [s].
- int atm_filter

  Time filter for atmospheric data output (0=no, 1=yes).
- char csi_basename [LEN]

  Basename of CSI data files.
- double csi_dt_out

  Time step for CSI data output [s].
- char csi_obsfile [LEN]

  Observation data file for CSI analysis.
- double csi_obsmin

  Minimum observation index to trigger detection.
- double csi_modmin

  Minimum column density to trigger detection [kg/m^2].
- int csi_nz

  Number of altitudes of gridded CSI data.
- double csi_z0

  Lower altitude of gridded CSI data [km].
- double csi_z1

  Upper altitude of gridded CSI data [km].
- int csi_nx

  Number of longitudes of gridded CSI data.
- double csi_lon0

  Lower longitude of gridded CSI data [deg].
- double csi_lon1

  Upper longitude of gridded CSI data [deg].
- int csi_ny

  Number of latitudes of gridded CSI data.
- double csi_lat0

  Lower latitude of gridded CSI data [deg].
- double csi_lat1

  Upper latitude of gridded CSI data [deg].
- char grid_basename [LEN]

  Basename of grid data files.
- char grid_gpfile [LEN]

  Gnuplot file for gridded data.
- double grid_dt_out

  Time step for gridded data output [s].
- int grid_sparse

  Sparse output in grid data files (0=no, 1=yes).
- int grid_nz

  Number of altitudes of gridded data.
- double grid_z0

  Lower altitude of gridded data [km].

- double grid_z1
  Upper altitude of gridded data [km].
- int grid_nx
  Number of longitudes of gridded data.
- double grid_lon0
  Lower longitude of gridded data [deg].
- double grid_lon1
  Upper longitude of gridded data [deg].
- int grid_ny
  Number of latitudes of gridded data.
- double grid_lat0
  Lower latitude of gridded data [deg].
- double grid_lat1
  Upper latitude of gridded data [deg].
- char prof_basename [LEN]
  Basename for profile output file.
- char prof_obsfile [LEN]
  Observation data file for profile output.
- int prof_nz
  Number of altitudes of gridded profile data.
- double prof_z0
  Lower altitude of gridded profile data [km].
- double prof_z1
  Upper altitude of gridded profile data [km].
- int prof_nx
  Number of longitudes of gridded profile data.
- double prof_lon0
  Lower longitude of gridded profile data [deg].
- double prof_lon1
  Upper longitude of gridded profile data [deg].
- int prof_ny
  Number of latitudes of gridded profile data.
- double prof_lat0
  Lower latitude of gridded profile data [deg].
- double prof_lat1
  Upper latitude of gridded profile data [deg].
- char ens_basename [LEN]
  Basename of ensemble data file.
- char stat_basename [LEN]
  Basename of station data file.
- double stat_lon
  Longitude of station [deg].
- double stat_lat
  Latitude of station [deg].
- double stat_r
  Search radius around station [km].

4.2.1 Detailed Description

Control parameters.

Definition at line 177 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl_t::nq

Number of quantities.

Definition at line 180 of file libtrac.h.

4.2.2.2 char ctl_t::qnt_name[NQ][LEN]

Quantity names.

Definition at line 183 of file libtrac.h.

4.2.2.3 char ctl_t::qnt_unit[NQ][LEN]

Quantity units.

Definition at line 186 of file libtrac.h.

4.2.2.4 char ctl_t::qnt_format[NQ][LEN]

Quantity output format.

Definition at line 189 of file libtrac.h.

4.2.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 192 of file libtrac.h.

4.2.2.6 int ctl_t::qnt_m

Quantity array index for mass.

Definition at line 195 of file libtrac.h.

4.2.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 198 of file libtrac.h.

4.2.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 201 of file libtrac.h.

4.2.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 204 of file libtrac.h.

4.2.2.10 int ctl_t::qnt_p

Quantity array index for pressure.

Definition at line 207 of file libtrac.h.

4.2.2.11 int ctl_t::qnt_t

Quantity array index for temperature.

Definition at line 210 of file libtrac.h.

4.2.2.12 int ctl_t::qnt_u

Quantity array index for zonal wind.

Definition at line 213 of file libtrac.h.

4.2.2.13 int ctl_t::qnt_v

Quantity array index for meridional wind.

Definition at line 216 of file libtrac.h.

4.2.2.14 int ctl_t::qnt_w

Quantity array index for vertical velocity.

Definition at line 219 of file libtrac.h.

4.2.2.15 int ctl_t::qnt_h2o

Quantity array index for water vapor vmr.

Definition at line 222 of file libtrac.h.

4.2.2.16 int ctl_t::qnt_o3

Quantity array index for ozone vmr.

Definition at line 225 of file libtrac.h.

4.2.2.17 int ctl_t::qnt_theta

Quantity array index for potential temperature.

Definition at line 228 of file libtrac.h.

4.2.2.18 int ctl_t::qnt_pv

Quantity array index for potential vorticity.

Definition at line 231 of file libtrac.h.

4.2.2.19 int ctl_t::qnt_tice

Quantity array index for T_ice.

Definition at line 234 of file libtrac.h.

4.2.2.20 int ctl_t::qnt_tsts

Quantity array index for T_STS.

Definition at line 237 of file libtrac.h.

4.2.2.21 int ctl_t::qnt_tnat

Quantity array index for T_NAT.

Definition at line 240 of file libtrac.h.

4.2.2.22 int ctl_t::qnt_stat

Quantity array index for station flag.

Definition at line 243 of file libtrac.h.

4.2.2.23 int ctl_t::direction

Direction flag (1=forward calculation, -1=backward calculation).

Definition at line 246 of file libtrac.h.

4.2.2.24 double ctl_t::t_start

Start time of simulation [s].

Definition at line 249 of file libtrac.h.

4.2.2.25 double ctl_t::t_stop

Stop time of simulation [s].

Definition at line 252 of file libtrac.h.

4.2.2.26 double ctl_t::dt_mod

Time step of simulation [s].

Definition at line 255 of file libtrac.h.

4.2.2.27 double ctl_t::dt_met

Time step of meteorological data [s].

Definition at line 258 of file libtrac.h.

4.2.2.28 int ctl_t::met_np

Number of target pressure levels.

Definition at line 261 of file libtrac.h.

4.2.2.29 double ctl_t::met_p[EP]

Target pressure levels [hPa].

Definition at line 264 of file libtrac.h.

4.2.2.30 int ctl_t::isosurf

Isosurface parameter (0=none, 1=pressure, 2=density, 3=theta, 4=balloon).

Definition at line 268 of file libtrac.h.

4.2.2.31 char ctl_t::balloon[LEN]

Balloon position filename.

Definition at line 271 of file libtrac.h.

4.2.2.32 double ctl_t::turb_dx_trop

Horizontal turbulent diffusion coefficient (troposphere) [m^2/s].

Definition at line 274 of file libtrac.h.

4.2.2.33 double ctl_t::turb_dx_strat

Horizontal turbulent diffusion coefficient (stratosphere) [m^2/s].

Definition at line 277 of file libtrac.h.

4.2.2.34 double ctl_t::turb_dz_trop

Vertical turbulent diffusion coefficient (troposphere) [m^2/s].

Definition at line 280 of file libtrac.h.

4.2.2.35 double ctl_t::turb_dz_strat

Vertical turbulent diffusion coefficient (stratosphere) [m^2/s].

Definition at line 283 of file libtrac.h.

4.2.2.36 double ctl_t::turb_meso

Scaling factor for mesoscale wind fluctuations.

Definition at line 286 of file libtrac.h.

4.2.2.37 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 289 of file libtrac.h.

4.2.2.38 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

Definition at line 292 of file libtrac.h.

4.2.2.39 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 295 of file libtrac.h.

4.2.2.40 double ctl_t::psc_hno3

HNO3 volume mixing ratio for PSC analysis.

Definition at line 298 of file libtrac.h.

4.2.2.41 char ctl_t::atm_basename[LEN]

Basename of atmospheric data files.

Definition at line 301 of file libtrac.h.

4.2.2.42 char ctl_t::atm_gpfile[LEN]

Gnuplot file for atmospheric data.

Definition at line 304 of file libtrac.h.

4.2.2.43 double ctl_t::atm_dt_out

Time step for atmospheric data output [s].

Definition at line 307 of file libtrac.h.

4.2.2.44 int ctl_t::atm_filter

Time filter for atmospheric data output (0=no, 1=yes).

Definition at line 310 of file libtrac.h.

4.2.2.45 char ctl_t::csi_basename[LEN]

Basename of CSI data files.

Definition at line 313 of file libtrac.h.

4.2.2.46 double ctl_t::csi_dt_out

Time step for CSI data output [s].

Definition at line 316 of file libtrac.h.

4.2.2.47 char ctl_t::csi_obsfile[LEN]

Observation data file for CSI analysis.

Definition at line 319 of file libtrac.h.

4.2.2.48 double ctl_t::csi_obsmin

Minimum observation index to trigger detection.

Definition at line 322 of file libtrac.h.

4.2.2.49 double ctl_t::csi_modmin

Minimum column density to trigger detection [kg/m^2].

Definition at line 325 of file libtrac.h.

4.2.2.50 int ctl_t::csi_nz

Number of altitudes of gridded CSI data.

Definition at line 328 of file libtrac.h.

4.2.2.51 double ctl_t::csi_z0

Lower altitude of gridded CSI data [km].

Definition at line 331 of file libtrac.h.

4.2.2.52 double ctl_t::csi_z1

Upper altitude of gridded CSI data [km].

Definition at line 334 of file libtrac.h.

4.2.2.53 int ctl_t::csi_nx

Number of longitudes of gridded CSI data.

Definition at line 337 of file libtrac.h.

4.2.2.54 double ctl_t::csi_lon0

Lower longitude of gridded CSI data [deg].

Definition at line 340 of file libtrac.h.

4.2.2.55 double ctl_t::csi_lon1

Upper longitude of gridded CSI data [deg].

Definition at line 343 of file libtrac.h.

4.2.2.56 int ctl_t::csi_ny

Number of latitudes of gridded CSI data.

Definition at line 346 of file libtrac.h.

4.2.2.57 double ctl_t::csi_lat0

Lower latitude of gridded CSI data [deg].

Definition at line 349 of file libtrac.h.

4.2.2.58 double ctl_t::csi_lat1

Upper latitude of gridded CSI data [deg].

Definition at line 352 of file libtrac.h.

4.2.2.59 char ctl_t::grid_basename[LEN]

Basename of grid data files.

Definition at line 355 of file libtrac.h.

4.2.2.60 char ctl_t::grid_gpfile[LEN]

Gnuplot file for gridded data.

Definition at line 358 of file libtrac.h.

4.2.2.61 double ctl_t::grid_dt_out

Time step for gridded data output [s].

Definition at line 361 of file libtrac.h.

4.2.2.62 int ctl_t::grid_sparse

Sparse output in grid data files (0=no, 1=yes).

Definition at line 364 of file libtrac.h.

4.2.2.63 int ctl_t::grid_nz

Number of altitudes of gridded data.

Definition at line 367 of file libtrac.h.

4.2.2.64 double ctl_t::grid_z0

Lower altitude of gridded data [km].

Definition at line 370 of file libtrac.h.

4.2.2.65 double ctl_t::grid_z1

Upper altitude of gridded data [km].

Definition at line 373 of file libtrac.h.

4.2.2.66 int ctl_t::grid_nx

Number of longitudes of gridded data.

Definition at line 376 of file libtrac.h.

4.2.2.67 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

Definition at line 379 of file libtrac.h.

4.2.2.68 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

Definition at line 382 of file libtrac.h.

4.2.2.69 int ctl_t::grid_ny

Number of latitudes of gridded data.

Definition at line 385 of file libtrac.h.

4.2.2.70 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

Definition at line 388 of file libtrac.h.

4.2.2.71 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

Definition at line 391 of file libtrac.h.

4.2.2.72 char ctl_t::prof_basename[LEN]

Basename for profile output file.

Definition at line 394 of file libtrac.h.

4.2.2.73 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

Definition at line 397 of file libtrac.h.

4.2.2.74 int ctl_t::prof_nz

Number of altitudes of gridded profile data.

Definition at line 400 of file libtrac.h.

4.2.2.75 double ctl_t::prof_z0

Lower altitude of gridded profile data [km].

Definition at line 403 of file libtrac.h.

4.2.2.76 double ctl_t::prof_z1

Upper altitude of gridded profile data [km].

Definition at line 406 of file libtrac.h.

4.2.2.77 int ctl_t::prof_nx

Number of longitudes of gridded profile data.

Definition at line 409 of file libtrac.h.

4.2.2.78 double ctl_t::prof_lon0

Lower longitude of gridded profile data [deg].

Definition at line 412 of file libtrac.h.

4.2.2.79 double ctl_t::prof_lon1

Upper longitude of gridded profile data [deg].

Definition at line 415 of file libtrac.h.

4.2.2.80 int ctl_t::prof_ny

Number of latitudes of gridded profile data.

Definition at line 418 of file libtrac.h.

4.2.2.81 double ctl_t::prof_lat0

Lower latitude of gridded profile data [deg].

Definition at line 421 of file libtrac.h.

4.2.2.82 double ctl_t::prof_lat1

Upper latitude of gridded profile data [deg].

Definition at line 424 of file libtrac.h.

4.2.2.83 char ctl_t::ens_basename[LEN]

Basename of ensemble data file.

Definition at line 427 of file libtrac.h.

4.2.2.84 char ctl_t::stat_basename[LEN]

Basename of station data file.

Definition at line 430 of file libtrac.h.

4.2.2.85 double ctl_t::stat_lon

Longitude of station [deg].

Definition at line 433 of file libtrac.h.

4.2.2.86 double ctl_t::stat_lat

Latitude of station [deg].

Definition at line 436 of file libtrac.h.

4.2.2.87 double ctl_t::stat_r

Search radius around station [km].

Definition at line 439 of file libtrac.h.

The documentation for this struct was generated from the following file:

- libtrac.h

4.3 met_t Struct Reference

Meteorological data.

`#include <libtrac.h>`

Data Fields

- double time
  Time [s].
- int nx
  Number of longitudes.
- int ny
  Number of latitudes.
- int np
  Number of pressure levels.
- double lon [EX]
  Longitude [deg].
- double lat [EY]
  Latitude [deg].
- double p [EP]
  Pressure [hPa].
- double ps [EX][EY]
  Surface pressure [hPa].
- float pl [EX][EY][EP]
  Pressure on model levels [hPa].
- float t [EX][EY][EP]
  Temperature [K].
- float u [EX][EY][EP]
  Zonal wind [m/s].
- float v [EX][EY][EP]
  Meridional wind [m/s].
- float w [EX][EY][EP]
  Vertical wind [hPa/s].
- float h2o [EX][EY][EP]
  Water vapor volume mixing ratio [1].
- float o3 [EX][EY][EP]
  Ozone volume mixing ratio [1].

4.3.1 Detailed Description

Meteorological data.

Definition at line 476 of file libtrac.h.

4.3.2 Field Documentation

4.3.2.1 double met_t::time

Time [s].

Definition at line 479 of file libtrac.h.

4.3.2.2 int met_t::nx

Number of longitudes.

Definition at line 482 of file libtrac.h.

4.3.2.3 int met_t::ny

Number of latitudes.

Definition at line 485 of file libtrac.h.

4.3.2.4 int met_t::np

Number of pressure levels.

Definition at line 488 of file libtrac.h.

4.3.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 491 of file libtrac.h.

4.3.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 494 of file libtrac.h.

4.3.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 497 of file libtrac.h.

4.3.2.8 double met_t::ps[EX][EY]

Surface pressure [hPa].

Definition at line 500 of file libtrac.h.

4.3.2.9 float met_t::pl[EX][EY][EP]

Pressure on model levels [hPa].

Definition at line 503 of file libtrac.h.

4.3.2.10 float met_t::t[EX][EY][EP]

Temperature [K].

Definition at line 506 of file libtrac.h.

4.3.2.11 float met_t::u[EX][EY][EP]

Zonal wind [m/s].

Definition at line 509 of file libtrac.h.

4.3.2.12 float met_t::v[EX][EY][EP]

Meridional wind [m/s].

Definition at line 512 of file libtrac.h.

4.3.2.13 float met_t::w[EX][EY][EP]

Vertical wind [hPa/s].

Definition at line 515 of file libtrac.h.

4.3.2.14 float met_t::h2o[EX][EY][EP]

Water vapor volume mixing ratio [1].

Definition at line 518 of file libtrac.h.

4.3.2.15 float met_t::o3[EX][EY][EP]

Ozone volume mixing ratio [1].

Definition at line 521 of file libtrac.h.

The documentation for this struct was generated from the following file:

- libtrac.h

5 File Documentation

5.1 center.c File Reference

Calculate center of mass of air parcels.

Functions

- int main (int argc, char ∗argv[ ])

5.1.1 Detailed Description

Calculate center of mass of air parcels.

Definition in file center.c.

5.1.2 Function Documentation

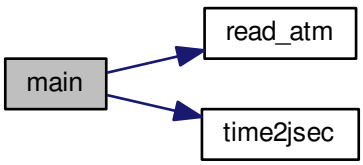
5.1.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 28 of file center.c.

```
00030                {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm;
00035
00036   FILE *out;
00037
00038   char *name, *year, *mon, *day, *hour, *min;
00039
00040   double latm, lats, lonm, lons, t, zm, zs;
00041
00042   int i, f, ip;
00043
00044   /* Allocate... */
00045   ALLOC(atm, atm_t, 1);
00046
00047   /* Check arguments... */
00048   if (argc < 3)
00049     ERRMSG("Give parameters: <outfile> <atm1> [<atm2> ...]");
00050
00051   /* Write info... */
00052   printf("Write center of mass data: %s\n", argv[1]);
00053
00054   /* Create output file... */
00055   if (!(out = fopen(argv[1], "w")))
00056     ERRMSG("Cannot create file!");
00057
00058   /* Write header... */
00059   fprintf(out,
00060           "# $1  = time [s]\n"
00061           "# $2  = altitude (mean) [km]\n"
00062           "# $3  = altitude (sigma) [km]\n"
00063           "# $4  = altitude (minimum) [km]\n"
00064           "# $5  = altitude (10%% percentile) [km]\n"
00065           "# $6  = altitude (1st quarter) [km]\n"
00066           "# $7  = altitude (median) [km]\n"
00067           "# $8  = altitude (3rd quarter) [km]\n"
00068           "# $9  = altitude (90%% percentile) [km]\n"
00069           "# $10 = altitude (maximum) [km]\n");
00070   fprintf(out,
00071           "# $11 = longitude (mean) [deg]\n"
00072           "# $12 = longitude (sigma) [deg]\n"
00073           "# $13 = longitude (minimum) [deg]\n"
00074           "# $14 = longitude (10%% percentile) [deg]\n"
00075           "# $15 = longitude (1st quarter) [deg]\n"
00076           "# $16 = longitude (median) [deg]\n"
00077           "# $17 = longitude (3rd quarter) [deg]\n"
00078           "# $18 = longitude (90%% percentile) [deg]\n"
00079           "# $19 = longitude (maximum) [deg]\n");
00080   fprintf(out,
00081           "# $20 = latitude (mean) [deg]\n"
00082           "# $21 = latitude (sigma) [deg]\n"
00083           "# $22 = latitude (minimum) [deg]\n"
00084           "# $23 = latitude (10%% percentile) [deg]\n"
00085           "# $24 = latitude (1st quarter) [deg]\n"
00086           "# $25 = latitude (median) [deg]\n"
00087           "# $26 = latitude (3rd quarter) [deg]\n"
00088           "# $27 = latitude (90%% percentile) [deg]\n"
00089           "# $28 = latitude (maximum) [deg]\n\n");
00090
00091   /* Loop over files... */
00092   for (f = 2; f < argc; f++) {
00093
00094     /* Read atmopheric data... */
00095     read_atm(argv[f], &ctl, atm);
00096
00097     /* Initialize... */
00098     zm = zs = 0;
00099     lonm = lons = 0;
00100     latm = lats = 0;
00101
00102     /* Calculate mean and standard deviation... */
00103     for (ip = 0; ip < atm->np; ip++) {
00104       zm += Z(atm->p[ip]) / atm->np;
00105       lonm += atm->lon[ip] / atm->np;
```

```
00106       latm += atm->lat[ip] / atm->np;
00107       zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00108       lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00109       lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00110     }
00111
00112     /* Normalize... */
00113     zs = sqrt(zs - gsl_pow_2(zm));
00114     lons = sqrt(lons - gsl_pow_2(lonm));
00115     lats = sqrt(lats - gsl_pow_2(latm));
00116
00117     /* Sort arrays... */
00118     gsl_sort(atm->p, 1, (size_t) atm->np);
00119     gsl_sort(atm->lon, 1, (size_t) atm->np);
00120     gsl_sort(atm->lat, 1, (size_t) atm->np);
00121
00122     /* Get date from filename... */
00123     for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
00124     name = strtok(&(argv[f][i]), "_");
00125     year = strtok(NULL, "_");
00126     mon = strtok(NULL, "_");
00127     day = strtok(NULL, "_");
00128     hour = strtok(NULL, "_");
00129     name = strtok(NULL, "_");   /* TODO: Why another "name" here? */
00130     min = strtok(name, ".");
00131     time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00132               &t);
00133
00134     /* Write data... */
00135     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g "
00136             "%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00137             t, zm, zs, Z(atm->p[atm->np - 1]),
00138             Z(atm->p[atm->np - atm->np / 10]),
00139             Z(atm->p[atm->np - atm->np / 4]),
00140             Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
00141             Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00142             lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00143             atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00144             atm->lon[atm->np - atm->np / 4],
00145             atm->lon[atm->np - atm->np / 10],
00146             atm->lon[atm->np - 1],
00147             latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00148             atm->lat[atm->np / 4], atm->lat[atm->np / 2],
00149             atm->lat[atm->np - atm->np / 4],
00150             atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00151   }
00152
00153   /* Close file... */
00154   fclose(out);
00155
00156   /* Free... */
00157   free(atm);
00158
00159   return EXIT_SUCCESS;
00160 }
```

Here is the call graph for this function:



5.2 center.c

```
00001 /*
```

```
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029   int argc,
00030   char *argv[]) {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm;
00035
00036   FILE *out;
00037
00038   char *name, *year, *mon, *day, *hour, *min;
00039
00040   double latm, lats, lonm, lons, t, zm, zs;
00041
00042   int i, f, ip;
00043
00044   /* Allocate... */
00045   ALLOC(atm, atm_t, 1);
00046
00047   /* Check arguments... */
00048   if (argc < 3)
00049     ERRMSG("Give parameters: <outfile> <atm1> [<atm2> ...]");
00050
00051   /* Write info... */
00052   printf("Write center of mass data: %s\n", argv[1]);
00053
00054   /* Create output file... */
00055   if (!(out = fopen(argv[1], "w")))
00056     ERRMSG("Cannot create file!");
00057
00058   /* Write header... */
00059   fprintf(out,
00060           "# $1  = time [s]\n"
00061           "# $2  = altitude (mean) [km]\n"
00062           "# $3  = altitude (sigma) [km]\n"
00063           "# $4  = altitude (minimum) [km]\n"
00064           "# $5  = altitude (10%% percentile) [km]\n"
00065           "# $6  = altitude (1st quarter) [km]\n"
00066           "# $7  = altitude (median) [km]\n"
00067           "# $8  = altitude (3rd quarter) [km]\n"
00068           "# $9  = altitude (90%% percentile) [km]\n"
00069           "# $10 = altitude (maximum) [km]\n");
00070   fprintf(out,
00071           "# $11 = longitude (mean) [deg]\n"
00072           "# $12 = longitude (sigma) [deg]\n"
00073           "# $13 = longitude (minimum) [deg]\n"
00074           "# $14 = longitude (10%% percentile) [deg]\n"
00075           "# $15 = longitude (1st quarter) [deg]\n"
00076           "# $16 = longitude (median) [deg]\n"
00077           "# $17 = longitude (3rd quarter) [deg]\n"
00078           "# $18 = longitude (90%% percentile) [deg]\n"
00079           "# $19 = longitude (maximum) [deg]\n");
00080   fprintf(out,
00081           "# $20 = latitude (mean) [deg]\n"
00082           "# $21 = latitude (sigma) [deg]\n"
00083           "# $22 = latitude (minimum) [deg]\n"
00084           "# $23 = latitude (10%% percentile) [deg]\n"
00085           "# $24 = latitude (1st quarter) [deg]\n"
00086           "# $25 = latitude (median) [deg]\n"
00087           "# $26 = latitude (3rd quarter) [deg]\n"
00088           "# $27 = latitude (90%% percentile) [deg]\n"
00089           "# $28 = latitude (maximum) [deg]\n\n");
00090
00091   /* Loop over files... */
00092   for (f = 2; f < argc; f++) {
00093
```

```
00094     /* Read atmopheric data... */
00095     read_atm(argv[f], &ctl, atm);
00096
00097     /* Initialize... */
00098     zm = zs = 0;
00099     lonm = lons = 0;
00100     latm = lats = 0;
00101
00102     /* Calculate mean and standard deviation... */
00103     for (ip = 0; ip < atm->np; ip++) {
00104       zm += Z(atm->p[ip]) / atm->np;
00105       lonm += atm->lon[ip] / atm->np;
00106       latm += atm->lat[ip] / atm->np;
00107       zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00108       lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00109       lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00110     }
00111
00112     /* Normalize... */
00113     zs = sqrt(zs - gsl_pow_2(zm));
00114     lons = sqrt(lons - gsl_pow_2(lonm));
00115     lats = sqrt(lats - gsl_pow_2(latm));
00116
00117     /* Sort arrays... */
00118     gsl_sort(atm->p, 1, (size_t) atm->np);
00119     gsl_sort(atm->lon, 1, (size_t) atm->np);
00120     gsl_sort(atm->lat, 1, (size_t) atm->np);
00121
00122     /* Get date from filename... */
00123     for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
00124     name = strtok(&(argv[f][i]), "_");
00125     year = strtok(NULL, "_");
00126     mon = strtok(NULL, "_");
00127     day = strtok(NULL, "_");
00128     hour = strtok(NULL, "_");
00129     name = strtok(NULL, "_");   /* TODO: Why another "name" here? */
00130     min = strtok(name, ".");
00131     time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00132               &t);
00133
00134     /* Write data... */
00135     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g "
00136             "%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00137             t, zm, zs, Z(atm->p[atm->np - 1]),
00138             Z(atm->p[atm->np - atm->np / 10]),
00139             Z(atm->p[atm->np - atm->np / 4]),
00140             Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
00141             Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00142             lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00143             atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00144             atm->lon[atm->np - atm->np / 4],
00145             atm->lon[atm->np - atm->np / 10],
00146             atm->lon[atm->np - 1],
00147             latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00148             atm->lat[atm->np / 4], atm->lat[atm->np / 2],
00149             atm->lat[atm->np - atm->np / 4],
00150             atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00151   }
00152
00153   /* Close file... */
00154   fclose(out);
00155
00156   /* Free... */
00157   free(atm);
00158
00159   return EXIT_SUCCESS;
00160 }
```

5.3 dist.c File Reference

Calculate transport deviations of trajectories.

Functions

- int main (int argc, char *argv[])

5.3.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file dist.c.

5.3.2 Function Documentation

5.3.2.1 int main (int *argc*, char * *argv[]*)

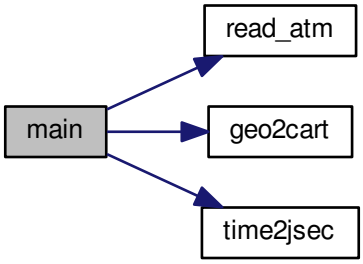
Definition at line 28 of file dist.c.

```
00030                {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm1, *atm2;
00035
00036   FILE *out;
00037
00038   char *name, *year, *mon, *day, *hour, *min;
00039
00040   double aux, x0[3], x1[3], x2[3], *lon1, *lat1, *p1, *lh1, *lv1,
00041     *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, ahtd2, avtd2,
00042     rhtd, rvtd, rhtd2, rvtd2, t, *dh, *dv;
00043
00044   int f, i, ip, iph, ipv;
00045
00046   /* Allocate... */
00047   ALLOC(atm1, atm_t, 1);
00048   ALLOC(atm2, atm_t, 1);
00049   ALLOC(lon1, double,
00050         NP);
00051   ALLOC(lat1, double,
00052         NP);
00053   ALLOC(p1, double,
00054         NP);
00055   ALLOC(lh1, double,
00056         NP);
00057   ALLOC(lv1, double,
00058         NP);
00059   ALLOC(lon2, double,
00060         NP);
00061   ALLOC(lat2, double,
00062         NP);
00063   ALLOC(p2, double,
00064         NP);
00065   ALLOC(lh2, double,
00066         NP);
00067   ALLOC(lv2, double,
00068         NP);
00069   ALLOC(dh, double,
00070         NP);
00071   ALLOC(dv, double,
00072         NP);
00073
00074   /* Check arguments... */
00075   if (argc < 4)
00076     ERRMSG
00077       ("Give parameters: <outfile> <atm1a> <atm1b> [<atm2a> <atm2b> ...]");
00078
00079   /* Write info... */
00080   printf("Write transport deviations: %s\n", argv[1]);
00081
00082   /* Create output file... */
00083   if (!(out = fopen(argv[1], "w")))
00084     ERRMSG("Cannot create file!");
00085
00086   /* Write header... */
00087   fprintf(out,
00088           "# $1  = time [s]\n"
00089           "# $2  = AHTD (mean) [km]\n"
00090           "# $3  = AHTD (sigma) [km]\n"
00091           "# $4  = AHTD (minimum) [km]\n"
00092           "# $5  = AHTD (10%% percentile) [km]\n"
00093           "# $6  = AHTD (1st quartile) [km]\n"
00094           "# $7  = AHTD (median) [km]\n"
```

```
00095           "# $8  = AHTD (3rd quartile) [km]\n"
00096           "# $9  = AHTD (90%% percentile) [km]\n"
00097           "# $10 = AHTD (maximum) [km]\n"
00098           "# $11 = AHTD (maximum trajectory index)\n"
00099           "# $12 = RHTD (mean) [%%]\n" "# $13 = RHTD (sigma) [%%]\n");
00100   fprintf(out,
00101           "# $14 = AVTD (mean) [km]\n"
00102           "# $15 = AVTD (sigma) [km]\n"
00103           "# $16 = AVTD (minimum) [km]\n"
00104           "# $17 = AVTD (10%% percentile) [km]\n"
00105           "# $18 = AVTD (1st quartile) [km]\n"
00106           "# $19 = AVTD (median) [km]\n"
00107           "# $20 = AVTD (3rd quartile) [km]\n"
00108           "# $21 = AVTD (90%% percentile) [km]\n"
00109           "# $22 = AVTD (maximum) [km]\n"
00110           "# $23 = AVTD (maximum trajectory index)\n"
00111           "# $24 = RVTD (mean) [%%]\n" "# $25 = RVTD (sigma) [%%]\n\n");
00112
00113   /* Loop over file pairs... */
00114   for (f = 2; f < argc; f += 2) {
00115
00116     /* Read atmopheric data... */
00117     read_atm(argv[f], &ctl, atm1);
00118     read_atm(argv[f + 1], &ctl, atm2);
00119
00120     /* Check if structs match... */
00121     if (atm1->np != atm2->np)
00122       ERRMSG("Different numbers of parcels!");
00123     for (ip = 0; ip < atm1->np; ip++)
00124       if (atm1->time[ip] != atm2->time[ip])
00125         ERRMSG("Times do not match!");
00126
00127     /* Init... */
00128     ahtd = ahtd2 = 0;
00129     avtd = avtd2 = 0;
00130     rhtd = rhtd2 = 0;
00131     rvtd = rvtd2 = 0;
00132
00133     /* Loop over air parcels... */
00134     for (ip = 0; ip < atm1->np; ip++) {
00135
00136       /* Get Cartesian coordinates... */
00137       geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00138       geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00139
00140       /* Calculate absolute transport deviations... */
00141       dh[ip] = DIST(x1, x2);
00142       ahtd += dh[ip];
00143       ahtd2 += gsl_pow_2(dh[ip]);
00144
00145       dv[ip] = fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00146       avtd += dv[ip];
00147       avtd2 += gsl_pow_2(dv[ip]);
00148
00149       /* Calculate relative transport deviations... */
00150       if (f > 2) {
00151
00152         /* Get trajectory lengths... */
00153         geo2cart(0, lon1[ip], lat1[ip], x0);
00154         lh1[ip] += DIST(x0, x1);
00155         lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00156
00157         geo2cart(0, lon2[ip], lat2[ip], x0);
00158         lh2[ip] += DIST(x0, x2);
00159         lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00160
00161         /* Get relative transport devations... */
00162         if (lh1[ip] + lh2[ip] > 0) {
00163           aux = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00164           rhtd += aux;
00165           rhtd2 += gsl_pow_2(aux);
00166         }
00167         if (lv1[ip] + lv2[ip] > 0) {
00168           aux =
00169             200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip])) / (lv1[ip] +
00170                                                             lv2[ip]);
00171           rvtd += aux;
00172           rvtd2 += gsl_pow_2(aux);
00173         }
00174       }
00175
00176       /* Save positions of air parcels... */
00177       lon1[ip] = atm1->lon[ip];
00178       lat1[ip] = atm1->lat[ip];
00179       p1[ip] = atm1->p[ip];
00180
00181       lon2[ip] = atm2->lon[ip];
```

```
00182       lat2[ip] = atm2->lat[ip];
00183       p2[ip] = atm2->p[ip];
00184     }
00185
00186     /* Get indices of trajectories with maximum errors... */
00187     iph = (int) gsl_stats_max_index(dh, 1, (size_t) atm1->np);
00188     ipv = (int) gsl_stats_max_index(dv, 1, (size_t) atm1->np);
00189
00190     /* Sort distances to calculate percentiles... */
00191     gsl_sort(dh, 1, (size_t) atm1->np);
00192     gsl_sort(dv, 1, (size_t) atm1->np);
00193
00194     /* Get date from filename... */
00195     for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
00196     name = strtok(&(argv[f][i]), "_");
00197     year = strtok(NULL, "_");
00198     mon = strtok(NULL, "_");
00199     day = strtok(NULL, "_");
00200     hour = strtok(NULL, "_");
00201     name = strtok(NULL, "_");   /* TODO: Why another "name" here? */
00202     min = strtok(name, ".");
00203     time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00204               &t);
00205
00206     /* Write output... */
00207     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d %g %g"
00208             " %g %g %g %g %g %g %g %g %d %g %g\n", t,
00209             ahtd / atm1->np,
00210             sqrt(ahtd2 / atm1->np - gsl_pow_2(ahtd / atm1->np)),
00211             dh[0], dh[atm1->np / 10], dh[atm1->np / 4], dh[atm1->np / 2],
00212             dh[atm1->np - atm1->np / 4], dh[atm1->np - atm1->np / 10],
00213             dh[atm1->np - 1], iph, rhtd / atm1->np,
00214             sqrt(rhtd2 / atm1->np - gsl_pow_2(rhtd / atm1->np)),
00215             avtd / atm1->np,
00216             sqrt(avtd2 / atm1->np - gsl_pow_2(avtd / atm1->np)),
00217             dv[0], dv[atm1->np / 10], dv[atm1->np / 4], dv[atm1->np / 2],
00218             dv[atm1->np - atm1->np / 4], dv[atm1->np - atm1->np / 10],
00219             dv[atm1->np - 1], ipv, rvtd / atm1->np,
00220             sqrt(rvtd2 / atm1->np - gsl_pow_2(rvtd / atm1->np)));
00221   }
00222
00223   /* Close file... */
00224   fclose(out);
00225
00226   /* Free... */
00227   free(atm1);
00228   free(atm2);
00229   free(lon1);
00230   free(lat1);
00231   free(p1);
00232   free(lh1);
00233   free(lv1);
00234   free(lon2);
00235   free(lat2);
00236   free(p2);
00237   free(lh2);
00238   free(lv2);
00239   free(dh);
00240   free(dv);
00241
00242   return EXIT_SUCCESS;
00243 }
```

Here is the call graph for this function:



5.4 dist.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029   int argc,
00030   char *argv[]) {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm1, *atm2;
00035
00036   FILE *out;
00037
00038   char *name, *year, *mon, *day, *hour, *min;
00039
00040   double aux, x0[3], x1[3], x2[3], *lon1, *lat1, *p1, *lh1, *lv1,
00041     *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, ahtd2, avtd2,
00042     rhtd, rvtd, rhtd2, rvtd2, t, *dh, *dv;
00043
00044   int f, i, ip, iph, ipv;
00045
00046   /* Allocate... */
00047   ALLOC(atm1, atm_t, 1);
00048   ALLOC(atm2, atm_t, 1);
00049   ALLOC(lon1, double,
00050         NP);
00051   ALLOC(lat1, double,
00052         NP);
00053   ALLOC(p1, double,
00054         NP);
00055   ALLOC(lh1, double,
00056         NP);
00057   ALLOC(lv1, double,
00058         NP);
00059   ALLOC(lon2, double,
```

```
00060         NP);
00061   ALLOC(lat2, double,
00062         NP);
00063   ALLOC(p2, double,
00064         NP);
00065   ALLOC(lh2, double,
00066         NP);
00067   ALLOC(lv2, double,
00068         NP);
00069   ALLOC(dh, double,
00070         NP);
00071   ALLOC(dv, double,
00072         NP);
00073
00074   /* Check arguments... */
00075   if (argc < 4)
00076     ERRMSG
00077       ("Give parameters: <outfile> <atm1a> <atm1b> [<atm2a> <atm2b> ...]");
00078
00079   /* Write info... */
00080   printf("Write transport deviations: %s\n", argv[1]);
00081
00082   /* Create output file... */
00083   if (!(out = fopen(argv[1], "w")))
00084     ERRMSG("Cannot create file!");
00085
00086   /* Write header... */
00087   fprintf(out,
00088           "# $1  = time [s]\n"
00089           "# $2  = AHTD (mean) [km]\n"
00090           "# $3  = AHTD (sigma) [km]\n"
00091           "# $4  = AHTD (minimum) [km]\n"
00092           "# $5  = AHTD (10%% percentile) [km]\n"
00093           "# $6  = AHTD (1st quartile) [km]\n"
00094           "# $7  = AHTD (median) [km]\n"
00095           "# $8  = AHTD (3rd quartile) [km]\n"
00096           "# $9  = AHTD (90%% percentile) [km]\n"
00097           "# $10 = AHTD (maximum) [km]\n"
00098           "# $11 = AHTD (maximum trajectory index)\n"
00099           "# $12 = RHTD (mean) [%%]\n" "# $13 = RHTD (sigma) [%%]\n");
00100   fprintf(out,
00101           "# $14 = AVTD (mean) [km]\n"
00102           "# $15 = AVTD (sigma) [km]\n"
00103           "# $16 = AVTD (minimum) [km]\n"
00104           "# $17 = AVTD (10%% percentile) [km]\n"
00105           "# $18 = AVTD (1st quartile) [km]\n"
00106           "# $19 = AVTD (median) [km]\n"
00107           "# $20 = AVTD (3rd quartile) [km]\n"
00108           "# $21 = AVTD (90%% percentile) [km]\n"
00109           "# $22 = AVTD (maximum) [km]\n"
00110           "# $23 = AVTD (maximum trajectory index)\n"
00111           "# $24 = RVTD (mean) [%%]\n" "# $25 = RVTD (sigma) [%%]\n\n");
00112
00113   /* Loop over file pairs... */
00114   for (f = 2; f < argc; f += 2) {
00115
00116     /* Read atmopheric data... */
00117     read_atm(argv[f], &ctl, atm1);
00118     read_atm(argv[f + 1], &ctl, atm2);
00119
00120     /* Check if structs match... */
00121     if (atm1->np != atm2->np)
00122       ERRMSG("Different numbers of parcels!");
00123     for (ip = 0; ip < atm1->np; ip++)
00124       if (atm1->time[ip] != atm2->time[ip])
00125         ERRMSG("Times do not match!");
00126
00127     /* Init... */
00128     ahtd = ahtd2 = 0;
00129     avtd = avtd2 = 0;
00130     rhtd = rhtd2 = 0;
00131     rvtd = rvtd2 = 0;
00132
00133     /* Loop over air parcels... */
00134     for (ip = 0; ip < atm1->np; ip++) {
00135
00136       /* Get Cartesian coordinates... */
00137       geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00138       geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00139
00140       /* Calculate absolute transport deviations... */
00141       dh[ip] = DIST(x1, x2);
00142       ahtd += dh[ip];
00143       ahtd2 += gsl_pow_2(dh[ip]);
00144
00145       dv[ip] = fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00146       avtd += dv[ip];
```

```
00147       avtd2 += gsl_pow_2(dv[ip]);
00148
00149       /* Calculate relative transport deviations... */
00150       if (f > 2) {
00151
00152         /* Get trajectory lengths... */
00153         geo2cart(0, lon1[ip], lat1[ip], x0);
00154         lh1[ip] += DIST(x0, x1);
00155         lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00156
00157         geo2cart(0, lon2[ip], lat2[ip], x0);
00158         lh2[ip] += DIST(x0, x2);
00159         lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00160
00161         /* Get relative transport devations... */
00162         if (lh1[ip] + lh2[ip] > 0) {
00163           aux = 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00164           rhtd += aux;
00165           rhtd2 += gsl_pow_2(aux);
00166         }
00167         if (lv1[ip] + lv2[ip] > 0) {
00168           aux =
00169             200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip])) / (lv1[ip] +
00170                                                             lv2[ip]);
00171           rvtd += aux;
00172           rvtd2 += gsl_pow_2(aux);
00173         }
00174       }
00175
00176       /* Save positions of air parcels... */
00177       lon1[ip] = atm1->lon[ip];
00178       lat1[ip] = atm1->lat[ip];
00179       p1[ip] = atm1->p[ip];
00180
00181       lon2[ip] = atm2->lon[ip];
00182       lat2[ip] = atm2->lat[ip];
00183       p2[ip] = atm2->p[ip];
00184     }
00185
00186     /* Get indices of trajectories with maximum errors... */
00187     iph = (int) gsl_stats_max_index(dh, 1, (size_t) atm1->np);
00188     ipv = (int) gsl_stats_max_index(dv, 1, (size_t) atm1->np);
00189
00190     /* Sort distances to calculate percentiles... */
00191     gsl_sort(dh, 1, (size_t) atm1->np);
00192     gsl_sort(dv, 1, (size_t) atm1->np);
00193
00194     /* Get date from filename... */
00195     for (i = (int) strlen(argv[f]) - 1; argv[f][i] != '/' || i == 0; i--);
00196     name = strtok(&(argv[f][i]), "_");
00197     year = strtok(NULL, "_");
00198     mon = strtok(NULL, "_");
00199     day = strtok(NULL, "_");
00200     hour = strtok(NULL, "_");
00201     name = strtok(NULL, "_");   /* TODO: Why another "name" here? */
00202     min = strtok(name, ".");
00203     time2jsec(atoi(year), atoi(mon), atoi(day), atoi(hour), atoi(min), 0, 0,
00204               &t);
00205
00206     /* Write output... */
00207     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %d %g %g"
00208             " %g %g %g %g %g %g %g %g %g %d %g %g\n", t,
00209             ahtd / atm1->np,
00210             sqrt(ahtd2 / atm1->np - gsl_pow_2(ahtd / atm1->np)),
00211             dh[0], dh[atm1->np / 10], dh[atm1->np / 4], dh[atm1->np / 2],
00212             dh[atm1->np - atm1->np / 4], dh[atm1->np - atm1->np / 10],
00213             dh[atm1->np - 1], iph, rhtd / atm1->np,
00214             sqrt(rhtd2 / atm1->np - gsl_pow_2(rhtd / atm1->np)),
00215             avtd / atm1->np,
00216             sqrt(avtd2 / atm1->np - gsl_pow_2(avtd / atm1->np)),
00217             dv[0], dv[atm1->np / 10], dv[atm1->np / 4], dv[atm1->np / 2],
00218             dv[atm1->np - atm1->np / 4], dv[atm1->np - atm1->np / 10],
00219             dv[atm1->np - 1], ipv, rvtd / atm1->np,
00220             sqrt(rvtd2 / atm1->np - gsl_pow_2(rvtd / atm1->np)));
00221   }
00222
00223   /* Close file... */
00224   fclose(out);
00225
00226   /* Free... */
00227   free(atm1);
00228   free(atm2);
00229   free(lon1);
00230   free(lat1);
00231   free(p1);
00232   free(lh1);
00233   free(lv1);
```

```
00234   free(lon2);
00235   free(lat2);
00236   free(p2);
00237   free(lh2);
00238   free(lv2);
00239   free(dh);
00240   free(dv);
00241
00242   return EXIT_SUCCESS;
00243 }
```

5.5 extract.c File Reference

Extract single trajectory from atmospheric data files.

Functions

- int main (int argc, char ∗argv[ ])

5.5.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

5.5.2 Function Documentation

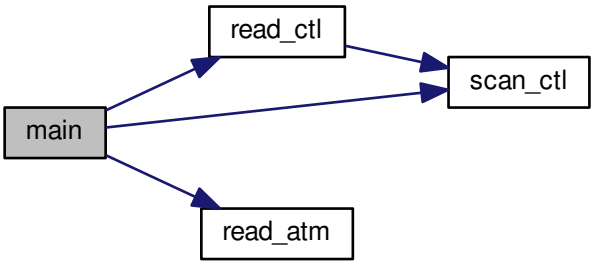
5.5.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 28 of file extract.c.

```
00030                {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm;
00035
00036   FILE *in, *out;
00037
00038   int f, ip, iq;
00039
00040   /* Allocate... */
00041   ALLOC(atm, atm_t, 1);
00042
00043   /* Check arguments... */
00044   if (argc < 4)
00045     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00046
00047   /* Read control parameters... */
00048   read_ctl(argv[1], argc, argv, &ctl);
00049   ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00050
00051   /* Write info... */
00052   printf("Write trajectory data: %s\n", argv[2]);
00053
00054   /* Create output file... */
00055   if (!(out = fopen(argv[2], "w")))
00056     ERRMSG("Cannot create file!");
00057
00058   /* Write header... */
00059   fprintf(out,
00060           "# $1 = time [s]\n"
00061           "# $2 = altitude [km]\n"
00062           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00063   for (iq = 0; iq < ctl.nq; iq++)
```

```
00064     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00065             ctl.qnt_unit[iq]);
00066   fprintf(out, "\n");
00067
00068   /* Loop over files... */
00069   for (f = 3; f < argc; f++) {
00070
00071     /* Read atmopheric data... */
00072     if (!(in = fopen(argv[f], "r")))
00073       continue;
00074     else
00075       fclose(in);
00076     read_atm(argv[f], &ctl, atm);
00077
00078     /* Write data... */
00079     fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081     for (iq = 0; iq < ctl.nq; iq++) {
00082       fprintf(out, " ");
00083       fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084     }
00085     fprintf(out, "\n");
00086   }
00087
00088   /* Close file... */
00089   fclose(out);
00090
00091   /* Free... */
00092   free(atm);
00093
00094   return EXIT_SUCCESS;
00095 }
```

Here is the call graph for this function:



5.6 extract.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
```

```
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029   int argc,
00030   char *argv[]) {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm;
00035
00036   FILE *in, *out;
00037
00038   int f, ip, iq;
00039
00040   /* Allocate... */
00041   ALLOC(atm, atm_t, 1);
00042
00043   /* Check arguments... */
00044   if (argc < 4)
00045     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00046
00047   /* Read control parameters... */
00048   read_ctl(argv[1], argc, argv, &ctl);
00049   ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00050
00051   /* Write info... */
00052   printf("Write trajectory data: %s\n", argv[2]);
00053
00054   /* Create output file... */
00055   if (!(out = fopen(argv[2], "w")))
00056     ERRMSG("Cannot create file!");
00057
00058   /* Write header... */
00059   fprintf(out,
00060           "# $1 = time [s]\n"
00061           "# $2 = altitude [km]\n"
00062           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00063   for (iq = 0; iq < ctl.nq; iq++)
00064     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00065             ctl.qnt_unit[iq]);
00066   fprintf(out, "\n");
00067
00068   /* Loop over files... */
00069   for (f = 3; f < argc; f++) {
00070
00071     /* Read atmopheric data... */
00072     if (!(in = fopen(argv[f], "r")))
00073       continue;
00074     else
00075       fclose(in);
00076     read_atm(argv[f], &ctl, atm);
00077
00078     /* Write data... */
00079     fprintf(out, "%.2f %g %g %g", atm->time[ip],
00080             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00081     for (iq = 0; iq < ctl.nq; iq++) {
00082       fprintf(out, " ");
00083       fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00084     }
00085     fprintf(out, "\n");
00086   }
00087
00088   /* Close file... */
00089   fclose(out);
00090
00091   /* Free... */
00092   free(atm);
00093
00094   return EXIT_SUCCESS;
00095 }
```

5.7 init.c File Reference

Create atmospheric data file with initial air parcel positions.

Functions

- int main (int argc, char ∗argv[ ])

5.7.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

5.7.2 Function Documentation

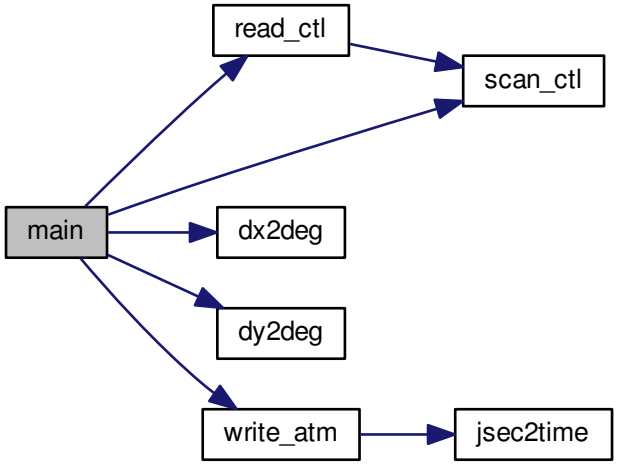
5.7.2.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 27 of file init.c.

```
00029                {
00030
00031   atm_t *atm;
00032
00033   ctl_t ctl;
00034
00035   gsl_rng *rng;
00036
00037   double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038     t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040   int ip, irep, rep;
00041
00042   /* Allocate... */
00043   ALLOC(atm, atm_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 3)
00047     ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052   t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053   dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054   z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055   z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056   dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
00057   lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058   lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059   dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060   lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061   lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062   dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
00063   st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064   sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065   slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066   slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067   sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068   ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069   uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070   ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071   ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072   rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00073   m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00074
00075   /* Initialize random number generator... */
00076   gsl_rng_env_setup();
00077   rng = gsl_rng_alloc(gsl_rng_default);
00078
00079   /* Create grid... */
00080   for (t = t0; t <= t1; t += dt)
00081     for (z = z0; z <= z1; z += dz)
00082       for (lon = lon0; lon <= lon1; lon += dlon)
00083         for (lat = lat0; lat <= lat1; lat += dlat)
00084           for (irep = 0; irep < rep; irep++) {
00085
00086             /* Set position... */
00087             atm->time[atm->np]
00088               = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00089                  + ut * (gsl_rng_uniform(rng) - 0.5));
00090             atm->p[atm->np]
00091               = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092                   + uz * (gsl_rng_uniform(rng) - 0.5));
00093             atm->lon[atm->np]
```

```
00094               = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00095                  + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
00096                  + ulon * (gsl_rng_uniform(rng) - 0.5));
00097             atm->lat[atm->np]
00098               = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00099                  + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
00100                  + ulat * (gsl_rng_uniform(rng) - 0.5));
00101
00102             /* Set particle counter... */
00103             if ((++atm->np) >= NP)
00104               ERRMSG("Too many particles!");
00105           }
00106
00107   /* Check number of air parcels... */
00108   if (atm->np <= 0)
00109     ERRMSG("Did not create any air parcels!");
00110
00111   /* Initialize mass... */
00112   if (ctl.qnt_m >= 0)
00113     for (ip = 0; ip < atm->np; ip++)
00114       atm->q[ctl.qnt_m][ip] = m / atm->np;
00115
00116   /* Save data... */
00117   write_atm(argv[2], &ctl, atm, t0);
00118
00119   /* Free... */
00120   gsl_rng_free(rng);
00121   free(atm);
00122
00123   return EXIT_SUCCESS;
00124 }
```

Here is the call graph for this function:



5.8 init.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   atm_t *atm;
00032
00033   ctl_t ctl;
00034
00035   gsl_rng *rng;
00036
00037   double dt, dz, dlon, dlat, lat0, lat1, lon0, lon1, t0, t1, z0, z1,
00038     t, z, lon, lat, st, sz, slon, slat, sx, ut, uz, ulon, ulat, m;
00039
00040   int ip, irep, rep;
00041
00042   /* Allocate... */
00043   ALLOC(atm, atm_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 3)
00047     ERRMSG("Give parameters: <ctl> <atm_out>");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   t0 = scan_ctl(argv[1], argc, argv, "INIT_T0", -1, "0", NULL);
00052   t1 = scan_ctl(argv[1], argc, argv, "INIT_T1", -1, "0", NULL);
00053   dt = scan_ctl(argv[1], argc, argv, "INIT_DT", -1, "1", NULL);
00054   z0 = scan_ctl(argv[1], argc, argv, "INIT_Z0", -1, "0", NULL);
00055   z1 = scan_ctl(argv[1], argc, argv, "INIT_Z1", -1, "0", NULL);
00056   dz = scan_ctl(argv[1], argc, argv, "INIT_DZ", -1, "1", NULL);
00057   lon0 = scan_ctl(argv[1], argc, argv, "INIT_LON0", -1, "0", NULL);
00058   lon1 = scan_ctl(argv[1], argc, argv, "INIT_LON1", -1, "0", NULL);
00059   dlon = scan_ctl(argv[1], argc, argv, "INIT_DLON", -1, "1", NULL);
00060   lat0 = scan_ctl(argv[1], argc, argv, "INIT_LAT0", -1, "0", NULL);
00061   lat1 = scan_ctl(argv[1], argc, argv, "INIT_LAT1", -1, "0", NULL);
00062   dlat = scan_ctl(argv[1], argc, argv, "INIT_DLAT", -1, "1", NULL);
00063   st = scan_ctl(argv[1], argc, argv, "INIT_ST", -1, "0", NULL);
00064   sz = scan_ctl(argv[1], argc, argv, "INIT_SZ", -1, "0", NULL);
00065   slon = scan_ctl(argv[1], argc, argv, "INIT_SLON", -1, "0", NULL);
00066   slat = scan_ctl(argv[1], argc, argv, "INIT_SLAT", -1, "0", NULL);
00067   sx = scan_ctl(argv[1], argc, argv, "INIT_SX", -1, "0", NULL);
00068   ut = scan_ctl(argv[1], argc, argv, "INIT_UT", -1, "0", NULL);
00069   uz = scan_ctl(argv[1], argc, argv, "INIT_UZ", -1, "0", NULL);
00070   ulon = scan_ctl(argv[1], argc, argv, "INIT_ULON", -1, "0", NULL);
00071   ulat = scan_ctl(argv[1], argc, argv, "INIT_ULAT", -1, "0", NULL);
00072   rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00073   m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00074
00075   /* Initialize random number generator... */
00076   gsl_rng_env_setup();
00077   rng = gsl_rng_alloc(gsl_rng_default);
00078
00079   /* Create grid... */
00080   for (t = t0; t <= t1; t += dt)
00081     for (z = z0; z <= z1; z += dz)
00082       for (lon = lon0; lon <= lon1; lon += dlon)
00083         for (lat = lat0; lat <= lat1; lat += dlat)
00084           for (irep = 0; irep < rep; irep++) {
00085
00086             /* Set position... */
00087             atm->time[atm->np]
00088               = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00089                  + ut * (gsl_rng_uniform(rng) - 0.5));
00090             atm->p[atm->np]
00091               = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00092                   + uz * (gsl_rng_uniform(rng) - 0.5));
00093             atm->lon[atm->np]
00094               = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00095                  + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
00096                  + ulon * (gsl_rng_uniform(rng) - 0.5));
00097             atm->lat[atm->np]
00098               = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00099                  + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
00100                 + ulat * (gsl_rng_uniform(rng) - 0.5));
00101
00102             /* Set particle counter... */
```

```
00103             if ((++atm->np) >= NP)
00104               ERRMSG("Too many particles!");
00105           }
00106
00107   /* Check number of air parcels... */
00108   if (atm->np <= 0)
00109     ERRMSG("Did not create any air parcels!");
00110
00111   /* Initialize mass... */
00112   if (ctl.qnt_m >= 0)
00113     for (ip = 0; ip < atm->np; ip++)
00114       atm->q[ctl.qnt_m][ip] = m / atm->np;
00115
00116   /* Save data... */
00117   write_atm(argv[2], &ctl, atm, t0);
00118
00119   /* Free... */
00120   gsl_rng_free(rng);
00121   free(atm);
00122
00123   return EXIT_SUCCESS;
00124 }
```

5.9 jsec2time.c File Reference

Convert Julian seconds to date.

Functions

- int main (int argc, char ∗argv[ ])

5.9.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

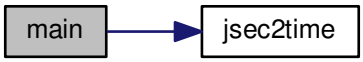
5.9.2 Function Documentation

5.9.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 27 of file jsec2time.c.

```
00029                 {
00030
00031   double jsec, remain;
00032
00033   int day, hour, min, mon, sec, year;
00034
00035   /* Check arguments... */
00036   if (argc < 2)
00037     ERRMSG("Give parameters: <jsec>");
00038
00039   /* Read arguments... */
00040   jsec = atof(argv[1]);
00041
00042   /* Convert time... */
00043   jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
00044   printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046   return EXIT_SUCCESS;
00047 }
```

Here is the call graph for this function:



5.10 jsec2time.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   double jsec, remain;
00032
00033   int day, hour, min, mon, sec, year;
00034
00035   /* Check arguments... */
00036   if (argc < 2)
00037     ERRMSG("Give parameters: <jsec>");
00038
00039   /* Read arguments... */
00040   jsec = atof(argv[1]);
00041
00042   /* Convert time... */
00043   jsec2time(jsec, &year, &mon, &day, &hour, &min, &sec, &remain);
00044   printf("%d %d %d %d %d %d %g\n", year, mon, day, hour, min, sec, remain);
00045
00046   return EXIT_SUCCESS;
00047 }
```

5.11 libtrac.c File Reference

MPTRAC library definitions.

Functions

- void cart2geo (double *x, double *z, double *lon, double *lat)

  Convert Cartesian coordinates to geolocation.
- double deg2dx (double dlon, double lat)

Measure wall-clock time.

- double tropopause (double t, double lat)
- void write_atm (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write atmospheric data.
- void write_csi (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write CSI data.
- void write_ens (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write ensemble data.
- void write_grid (const char *filename, ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double t)

  Write gridded data.
- void write_prof (const char *filename, ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double t)

  Write profile data.
- void write_station (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write station data.

5.11.1 Detailed Description

MPTRAC library definitions.

Definition in file libtrac.c.

5.11.2 Function Documentation

5.11.2.1 void cart2geo (double * *x*, double * *z*, double * *lon*, double * *lat*)

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

```
00033                {
00034
00035   double radius;
00036
00037   radius = NORM(x);
00038   *lat = asin(x[2] / radius) * 180 / M_PI;
00039   *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040   *z = radius - RE;
00041 }
```

5.11.2.2 double deg2dx (double *dlon*, double *lat*)

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

```
00047              {
00048
00049   return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00050 }
```

5.11.2.3 double deg2dy (double *dlat*)

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

```
00055                {
00056
00057   return dlat * M_PI * RE / 180.;
00058 }
```

5.11.2.4 double dp2dz (double *dp*, double *p*)

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

```
00064             {
00065
00066   return -dp * H0 / p;
00067 }
```

5.11.2.5 double dx2deg (double *dx*, double *lat*)

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

```
00073               {
00074
00075   /* Avoid singularity at poles... */
00076   if (lat < -89.999 || lat > 89.999)
00077     return 0;
00078   else
00079     return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00080 }
```

5.11.2.6 double dy2deg (double *dy*)

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

```
00085              {
00086
00087   return dy * 180. / (M_PI * RE);
00088 }
```

5.11.2.7 double dz2dp (double *dz*, double *p*)

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

```
00094             {
00095
00096   return -dz * p / H0;
00097 }
```

5.11.2.8 void geo2cart (double *z*, double *lon*, double *lat*, double * *x*)

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

```
00105             {
00106
00107   double radius;
00108
00109   radius = z + RE;
00110   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00111   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00112   x[2] = radius * sin(lat / 180 * M_PI);
00113 }
```

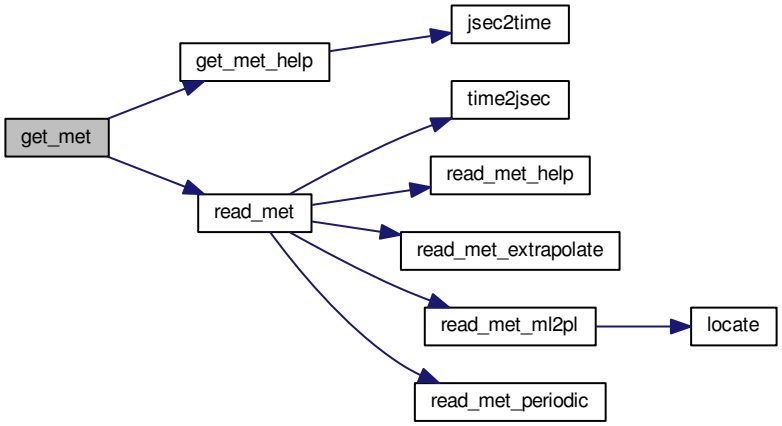
5.11.2.9 void get_met (ctl_t * *ctl*, char * *metbase*, double *t*, met_t * *met0*, met_t * *met1*)

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

```
00122                {
00123
00124   char filename[LEN];
00125
00126   static int init;
00127
00128   /* Init... */
00129   if (!init) {
00130     init = 1;
00131
00132     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00133     read_met(ctl, filename, met0);
00134
00135     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00136     read_met(ctl, filename, met1);
00137   }
00138
00139   /* Read new data for forward trajectories... */
00140   if (t > met1->time && ctl->direction == 1) {
00141     memcpy(met0, met1, sizeof(met_t));
00142     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00143     read_met(ctl, filename, met1);
00144   }
00145
00146   /* Read new data for backward trajectories... */
00147   if (t < met0->time && ctl->direction == -1) {
00148     memcpy(met1, met0, sizeof(met_t));
00149     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00150     read_met(ctl, filename, met0);
00151   }
00152 }
```

Here is the call graph for this function:



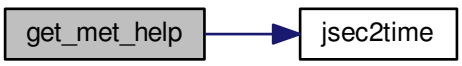
5.11.2.10 void get_met_help (double *t*, int *direct*, char * *metbase*, double *dt_met*, char * *filename*)

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

```
00161                   {
00162
00163   double t6, r;
00164
00165   int year, mon, day, hour, min, sec;
00166
00167   /* Round time to fixed intervals... */
00168   if (direct == -1)
00169     t6 = floor(t / dt_met) * dt_met;
00170   else
00171     t6 = ceil(t / dt_met) * dt_met;
00172
00173   /* Decode time... */
00174   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
00176   /* Set filename... */
00177   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00178 }
```

Here is the call graph for this function:



5.11.2.11 void intpol_met_2d (double *array[EX][EY],* int *ix,* int *iy,* double *wx,* double *wy,* double * *var*)

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

```
00188                {
00189
00190   double aux00, aux01, aux10, aux11;
00191
00192   /* Set variables... */
00193   aux00 = array[ix][iy];
00194   aux01 = array[ix][iy + 1];
00195   aux10 = array[ix + 1][iy];
00196   aux11 = array[ix + 1][iy + 1];
00197
00198   /* Interpolate horizontally... */
00199   aux00 = wy * (aux00 - aux01) + aux01;
00200   aux11 = wy * (aux10 - aux11) + aux11;
00201   *var = wx * (aux00 - aux11) + aux11;
00202 }
```

5.11.2.12 void intpol_met_3d (float *array[EX][EY][EP],* int *ip,* int *ix,* int *iy,* double *wp,* double *wx,* double *wy,* double * *var*)

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

```
00214                {
00215
00216   double aux00, aux01, aux10, aux11;
00217
00218   /* Interpolate vertically... */
00219   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00220     + array[ix][iy][ip + 1];
00221   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00222     + array[ix][iy + 1][ip + 1];
00223   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00224     + array[ix + 1][iy][ip + 1];
00225   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00226     + array[ix + 1][iy + 1][ip + 1];
00227
00228   /* Interpolate horizontally... */
00229   aux00 = wy * (aux00 - aux01) + aux01;
00230   aux11 = wy * (aux10 - aux11) + aux11;
00231   *var = wx * (aux00 - aux11) + aux11;
00232 }
```

5.11.2.13 void intpol_met_space (met_t * *met,* double *p,* double *lon,* double *lat,* double * *ps,* double * *t,* double * *u,* double * *v,* double * *w,* double * *h2o,* double * *o3*)

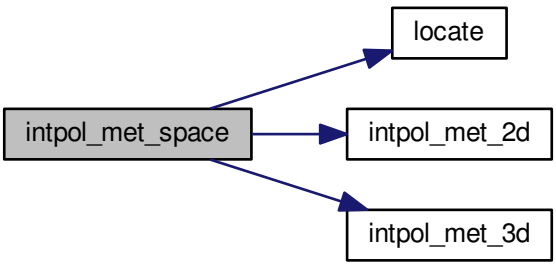
Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

```
00247                {
00248
00249   double wp, wx, wy;
00250
00251   int ip, ix, iy;
00252
00253   /* Check longitude... */
00254   if (met->lon[met->nx - 1] > 180 && lon < 0)
00255     lon += 360;
00256
00257   /* Get indices... */
00258   ip = locate(met->p, met->np, p);
00259   ix = locate(met->lon, met->nx, lon);
00260   iy = locate(met->lat, met->ny, lat);
```

```
00261
00262   /* Get weights... */
00263   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00264   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00265   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00266
00267   /* Interpolate... */
00268   if (ps != NULL)
00269     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00270   if (t != NULL)
00271     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00272   if (u != NULL)
00273     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00274   if (v != NULL)
00275     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00276   if (w != NULL)
00277     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00278   if (h2o != NULL)
00279     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00280   if (o3 != NULL)
00281     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
```

Here is the call graph for this function:



5.11.2.14 void intpol_met_time (met_t * *met0*, met_t * *met1*, double *ts*, double *p*, double *lon*, double *lat*, double * *ps*, double * *t*, double * *u*, double * *v*, double * *w*, double * *h2o*, double * *o3*)

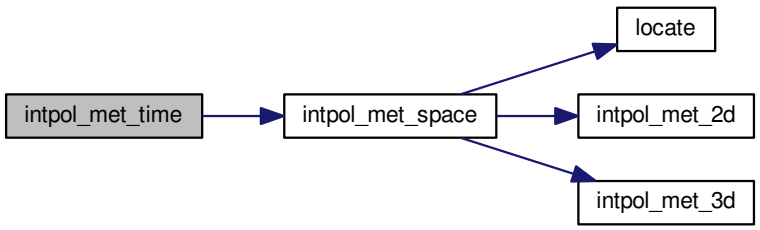
Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

```
00299             {
00300
00301   double h2o0, h2o1, o30, o31, ps0, ps1, t0, t1, u0, u1, v0, v1, w0, w1, wt;
00302
00303   /* Spatial interpolation... */
00304   intpol_met_space(met0, p, lon, lat,
00305                    ps == NULL ? NULL : &ps0,
00306                    t == NULL ? NULL : &t0,
00307                    u == NULL ? NULL : &u0,
00308                    v == NULL ? NULL : &v0,
00309                    w == NULL ? NULL : &w0,
00310                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00311   intpol_met_space(met1, p, lon, lat,
00312                    ps == NULL ? NULL : &ps1,
00313                    t == NULL ? NULL : &t1,
00314                    u == NULL ? NULL : &u1,
00315                    v == NULL ? NULL : &v1,
00316                    w == NULL ? NULL : &w1,
00317                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
```

```
00319   /* Get weighting factor... */
00320   wt = (met1->time - ts) / (met1->time - met0->time);
00321
00322   /* Interpolate... */
00323   if (ps != NULL)
00324     *ps = wt * (ps0 - ps1) + ps1;
00325   if (t != NULL)
00326     *t = wt * (t0 - t1) + t1;
00327   if (u != NULL)
00328     *u = wt * (u0 - u1) + u1;
00329   if (v != NULL)
00330     *v = wt * (v0 - v1) + v1;
00331   if (w != NULL)
00332     *w = wt * (w0 - w1) + w1;
00333   if (h2o != NULL)
00334     *h2o = wt * (h2o0 - h2o1) + h2o1;
00335   if (o3 != NULL)
00336     *o3 = wt * (o30 - o31) + o31;
00337 }
```

Here is the call graph for this function:



5.11.2.15 void jsec2time (double *jsec*, int * *year*, int * *mon*, int * *day*, int * *hour*, int * *min*, int * *sec*, double * *remain*)

Convert seconds to date.

Definition at line 341 of file libtrac.c.

```
00349                   {
00350
00351   struct tm t0, *t1;
00352
00353   time_t jsec0;
00354
00355   t0.tm_year = 100;
00356   t0.tm_mon = 0;
00357   t0.tm_mday = 1;
00358   t0.tm_hour = 0;
00359   t0.tm_min = 0;
00360   t0.tm_sec = 0;
00361
00362   jsec0 = (time_t) jsec + timegm(&t0);
00363   t1 = gmtime(&jsec0);
00364
00365   *year = t1->tm_year + 1900;
00366   *mon = t1->tm_mon + 1;
00367   *day = t1->tm_mday;
00368   *hour = t1->tm_hour;
00369   *min = t1->tm_min;
00370   *sec = t1->tm_sec;
00371   *remain = jsec - floor(jsec);
00372 }
```

5.11.2.16 int locate (double * *xx*, int *n*, double *x*)

Find array index.

Definition at line 376 of file libtrac.c.

```
00379            {
00380
00381   int i, ilo, ihi;
00382
00383   ilo = 0;
00384   ihi = n - 1;
00385   i = (ihi + ilo) >> 1;
00386
00387   if (xx[i] < xx[i + 1])
00388     while (ihi > ilo + 1) {
00389       i = (ihi + ilo) >> 1;
00390       if (xx[i] > x)
00391         ihi = i;
00392       else
00393         ilo = i;
00394   } else
00395     while (ihi > ilo + 1) {
00396       i = (ihi + ilo) >> 1;
00397       if (xx[i] <= x)
00398         ihi = i;
00399       else
00400         ilo = i;
00401     }
00402
00403   return ilo;
00404 }
```

5.11.2.17 void read_atm (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*)

Read atmospheric data.

Definition at line 408 of file libtrac.c.

```
00411               {
00412
00413   FILE *in;
00414
00415   char line[LEN], *tok;
00416
00417   int iq;
00418
00419   /* Init... */
00420   atm->np = 0;
00421
00422   /* Write info... */
00423   printf("Read atmospheric data: %s\n", filename);
00424
00425   /* Open file... */
00426   if (!(in = fopen(filename, "r")))
00427     ERRMSG("Cannot open file!");
00428
00429   /* Read line... */
00430   while (fgets(line, LEN, in)) {
00431
00432     /* Read data... */
00433     TOK(line, tok, "%lg", atm->time[atm->np]);
00434     TOK(NULL, tok, "%lg", atm->p[atm->np]);
00435     TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00436     TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00437     for (iq = 0; iq < ctl->nq; iq++)
00438       TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00439
00440     /* Convert altitude to pressure... */
00441     atm->p[atm->np] = P(atm->p[atm->np]);
00442
00443     /* Increment data point counter... */
00444     if ((++atm->np) > NP)
00445       ERRMSG("Too many data points!");
00446   }
00447
00448   /* Close file... */
00449   fclose(in);
00450
00451   /* Check number of points... */
00452   if (atm->np < 1)
00453     ERRMSG("Can not read any data!");
00454 }
```

5.11.2.18 void read_ctl (const char * *filename*, int *argc*, char * *argv[]*, ctl_t * *ctl*)

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462               {
00463
00464   int ip, iq;
00465
00466   /* Write info... */
00467   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00468          "(executable: %s | compiled: %s, %s)\n\n",
00469          argv[0], __DATE__, __TIME__);
00470
00471   /* Initialize quantity indices... */
00472   ctl->qnt_ens = -1;
00473   ctl->qnt_m = -1;
00474   ctl->qnt_r = -1;
00475   ctl->qnt_rho = -1;
00476   ctl->qnt_ps = -1;
00477   ctl->qnt_p = -1;
00478   ctl->qnt_t = -1;
00479   ctl->qnt_u = -1;
00480   ctl->qnt_v = -1;
00481   ctl->qnt_w = -1;
00482   ctl->qnt_h2o = -1;
00483   ctl->qnt_o3 = -1;
00484   ctl->qnt_theta = -1;
00485   ctl->qnt_pv = -1;
00486   ctl->qnt_tice = -1;
00487   ctl->qnt_tsts = -1;
00488   ctl->qnt_tnat = -1;
00489   ctl->qnt_stat = -1;
00490
00491   /* Read quantities... */
00492   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00493   for (iq = 0; iq < ctl->nq; iq++) {
00494
00495     /* Read quantity name and format... */
00496     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
00497     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
00498              ctl->qnt_format[iq]);
00499
00500     /* Try to identify quantity... */
00501     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00502       ctl->qnt_ens = iq;
00503       sprintf(ctl->qnt_unit[iq], "-");
00504     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00505       ctl->qnt_m = iq;
00506       sprintf(ctl->qnt_unit[iq], "kg");
00507     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
00508       ctl->qnt_r = iq;
00509       sprintf(ctl->qnt_unit[iq], "m");
00510     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00511       ctl->qnt_rho = iq;
00512       sprintf(ctl->qnt_unit[iq], "kg/m^3");
00513     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00514       ctl->qnt_ps = iq;
00515       sprintf(ctl->qnt_unit[iq], "hPa");
00516     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00517       ctl->qnt_p = iq;
00518       sprintf(ctl->qnt_unit[iq], "hPa");
00519     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00520       ctl->qnt_t = iq;
00521       sprintf(ctl->qnt_unit[iq], "K");
00522     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00523       ctl->qnt_u = iq;
00524       sprintf(ctl->qnt_unit[iq], "m/s");
00525     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00526       ctl->qnt_v = iq;
00527       sprintf(ctl->qnt_unit[iq], "m/s");
00528     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00529       ctl->qnt_w = iq;
00530       sprintf(ctl->qnt_unit[iq], "hPa/s");
00531     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00532       ctl->qnt_h2o = iq;
00533       sprintf(ctl->qnt_unit[iq], "1");
00534     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00535       ctl->qnt_o3 = iq;
00536       sprintf(ctl->qnt_unit[iq], "1");
00537     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00538       ctl->qnt_theta = iq;
```

```
00539       sprintf(ctl->qnt_unit[iq], "K");
00540     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00541       ctl->qnt_pv = iq;
00542       sprintf(ctl->qnt_unit[iq], "PVU");
00543     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00544       ctl->qnt_tice = iq;
00545       sprintf(ctl->qnt_unit[iq], "K");
00546     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00547       ctl->qnt_tsts = iq;
00548       sprintf(ctl->qnt_unit[iq], "K");
00549     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00550       ctl->qnt_tnat = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00553       ctl->qnt_stat = iq;
00554       sprintf(ctl->qnt_unit[iq], "-");
00555     } else
00556       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00557   }
00558
00559   /* Time steps of simulation... */
00560   ctl->direction =
00561     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00562   if (ctl->direction != -1 && ctl->direction != 1)
00563     ERRMSG("Set DIRECTION to -1 or 1!");
00564   ctl->t_start =
00565     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00566   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00567   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00568
00569   /* Meteorological data... */
00570   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00571   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00572   if (ctl->met_np > EP)
00573     ERRMSG("Too many levels!");
00574   for (ip = 0; ip < ctl->met_np; ip++)
00575     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00576
00577   /* Isosurface parameters... */
00578   ctl->isosurf
00579     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00580   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00581
00582   /* Diffusion parameters... */
00583   ctl->turb_dx_trop
00584     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00585   ctl->turb_dx_strat
00586     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00587   ctl->turb_dz_trop
00588     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00589   ctl->turb_dz_strat
00590     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00591   ctl->turb_meso =
00592     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00593
00594   /* Life time of particles... */
00595   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00596   ctl->tdec_strat =
00597     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00598
00599   /* PSC analysis... */
00600   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00601   ctl->psc_hno3 =
00602     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00603
00604   /* Output of atmospheric data... */
00605   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00606   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00607   ctl->atm_dt_out =
00608     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00609   ctl->atm_filter =
00610     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00611
00612   /* Output of CSI data... */
00613   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00614   ctl->csi_dt_out =
00615     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
00616   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00617            ctl->csi_obsfile);
00618   ctl->csi_obsmin =
00619     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00620   ctl->csi_modmin =
00621     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00622   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
00623   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
```

```
00624   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00625   ctl->csi_lon0 =
00626     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
00627   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00628   ctl->csi_nx =
00629     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
00630   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
00631   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
00632   ctl->csi_ny =
00633     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00634
00635   /* Output of ensemble data... */
00636   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
00637
00638   /* Output of grid data... */
00639   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00640            ctl->grid_basename);
00641   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00642   ctl->grid_dt_out =
00643     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00644   ctl->grid_sparse =
00645     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00646   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
00647   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00648   ctl->grid_nz =
00649     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00650   ctl->grid_lon0 =
00651     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
00652   ctl->grid_lon1 =
00653     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00654   ctl->grid_nx =
00655     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00656   ctl->grid_lat0 =
00657     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
00658   ctl->grid_lat1 =
00659     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00660   ctl->grid_ny =
00661     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00662
00663   /* Output of profile data... */
00664   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00665            ctl->prof_basename);
00666   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
00667   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00668   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00669   ctl->prof_nz =
00670     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00671   ctl->prof_lon0 =
00672     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
00673   ctl->prof_lon1 =
00674     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00675   ctl->prof_nx =
00676     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00677   ctl->prof_lat0 =
00678     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
00679   ctl->prof_lat1 =
00680     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00681   ctl->prof_ny =
00682     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00683
00684   /* Output of station data... */
00685   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00686            ctl->stat_basename);
00687   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
00688   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
00689   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00690 }
```

Here is the call graph for this function:

5.11.2.19 void read_met (ctl_t * *ctl*, char * *filename*, met_t * *met*)

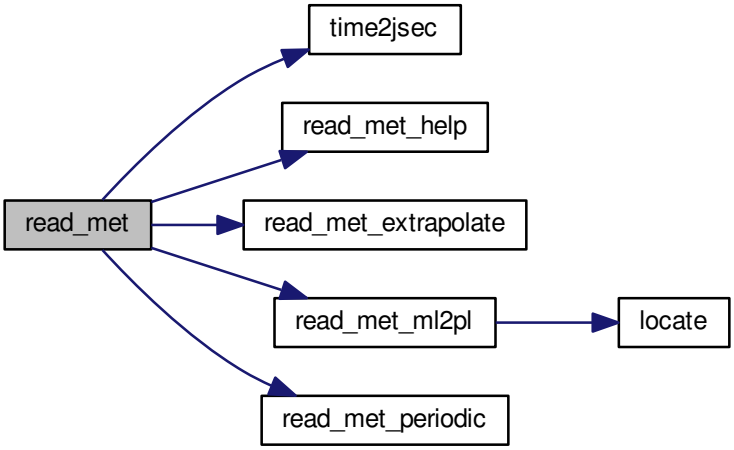
Read meteorological data file.

Definition at line 694 of file libtrac.c.

```
00697                {
00698
00699   char tstr[10];
00700
00701   static float help[EX * EY];
00702
00703   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00704
00705   size_t np, nx, ny;
00706
00707   /* Write info... */
00708   printf("Read meteorological data: %s\n", filename);
00709
00710   /* Get time from filename... */
00711   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00712   year = atoi(tstr);
00713   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00714   mon = atoi(tstr);
00715   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00716   day = atoi(tstr);
00717   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00718   hour = atoi(tstr);
00719   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00720
00721   /* Open netCDF file... */
00722   NC(nc_open(filename, NC_NOWRITE, &ncid));
00723
00724   /* Get dimensions... */
00725   NC(nc_inq_dimid(ncid, "lon", &dimid));
00726   NC(nc_inq_dimlen(ncid, dimid, &nx));
00727   if (nx > EX)
00728     ERRMSG("Too many longitudes!");
00729
00730   NC(nc_inq_dimid(ncid, "lat", &dimid));
00731   NC(nc_inq_dimlen(ncid, dimid, &ny));
00732   if (ny > EY)
00733     ERRMSG("Too many latitudes!");
00734
00735   NC(nc_inq_dimid(ncid, "lev", &dimid));
00736   NC(nc_inq_dimlen(ncid, dimid, &np));
00737   if (np > EP)
00738     ERRMSG("Too many levels!");
00739
00740   /* Store dimensions... */
00741   met->np = (int) np;
00742   met->nx = (int) nx;
00743   met->ny = (int) ny;
00744
00745   /* Get horizontal grid... */
00746   NC(nc_inq_varid(ncid, "lon", &varid));
00747   NC(nc_get_var_double(ncid, varid, met->lon));
00748   NC(nc_inq_varid(ncid, "lat", &varid));
00749   NC(nc_get_var_double(ncid, varid, met->lat));
00750
00751   /* Read meteorological data... */
00752   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00753   read_met_help(ncid, "u", "U", met, met->u, 1.0);
```

```
00754   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00755   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00756   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00757   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00758
00759   /* Meteo data on pressure levels... */
00760   if (ctl->met_np <= 0) {
00761
00762     /* Read pressure levels from file... */
00763     NC(nc_inq_varid(ncid, "lev", &varid));
00764     NC(nc_get_var_double(ncid, varid, met->p));
00765     for (ip = 0; ip < met->np; ip++)
00766       met->p[ip] /= 100.;
00767
00768     /* Extrapolate data for lower boundary... */
00769     read_met_extrapolate(met);
00770   }
00771
00772   /* Meteo data on model levels... */
00773   else {
00774
00775     /* Read pressure data from file... */
00776     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00777
00778     /* Interpolate from model levels to pressure levels... */
00779     read_met_ml2pl(ctl, met, met->t);
00780     read_met_ml2pl(ctl, met, met->u);
00781     read_met_ml2pl(ctl, met, met->v);
00782     read_met_ml2pl(ctl, met, met->w);
00783     read_met_ml2pl(ctl, met, met->h2o);
00784     read_met_ml2pl(ctl, met, met->o3);
00785
00786     /* Set pressure levels... */
00787     met->np = ctl->met_np;
00788     for (ip = 0; ip < met->np; ip++)
00789       met->p[ip] = ctl->met_p[ip];
00790   }
00791
00792   /* Check ordering of pressure levels... */
00793   for (ip = 1; ip < met->np; ip++)
00794     if (met->p[ip - 1] < met->p[ip])
00795       ERRMSG("Pressure levels must be descending!");
00796
00797   /* Read surface pressure... */
00798   if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00799     NC(nc_get_var_float(ncid, varid, help));
00800     for (iy = 0; iy < met->ny; iy++)
00801       for (ix = 0; ix < met->nx; ix++)
00802         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00803   } else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00804     NC(nc_get_var_float(ncid, varid, help));
00805     for (iy = 0; iy < met->ny; iy++)
00806       for (ix = 0; ix < met->nx; ix++)
00807         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00808   } else
00809     for (ix = 0; ix < met->nx; ix++)
00810       for (iy = 0; iy < met->ny; iy++)
00811         met->ps[ix][iy] = met->p[0];
00812
00813   /* Create periodic boundary conditions... */
00814   read_met_periodic(met);
00815
00816   /* Close file... */
00817   NC(nc_close(ncid));
00818 }
```

Here is the call graph for this function:



5.11.2.20 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 822 of file libtrac.c.

```
00823               {
00824
00825   int ip, ip0, ix, iy;
00826
00827   /* Loop over columns... */
00828   for (ix = 0; ix < met->nx; ix++)
00829     for (iy = 0; iy < met->ny; iy++) {
00830
00831       /* Find lowest valid data point... */
00832       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00833         if (!gsl_finite(met->t[ix][iy][ip0])
00834             || !gsl_finite(met->u[ix][iy][ip0])
00835             || !gsl_finite(met->v[ix][iy][ip0])
00836             || !gsl_finite(met->w[ix][iy][ip0]))
00837           break;
00838
00839       /* Extrapolate... */
00840       for (ip = ip0; ip >= 0; ip--) {
00841         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00842         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00843         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00844         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00845         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00846         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00847       }
00848     }
00849 }
```

5.11.2.21 void read_met_help (int *ncid*, char * *varname*, char * *varname2*, met_t * *met*, float *dest[EX][EY][EP]*, float *scl*)

Read and convert variable from meteorological data file.

Definition at line 853 of file libtrac.c.

```
00859             {
00860
00861   static float help[EX * EY * EP];
00862
00863   int ip, ix, iy, n = 0, varid;
00864
00865   /* Check if variable exists... */
00866   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00867     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00868       return;
00869
00870   /* Read data... */
00871   NC(nc_get_var_float(ncid, varid, help));
00872
00873   /* Copy and check data... */
00874   for (ip = 0; ip < met->np; ip++)
00875     for (iy = 0; iy < met->ny; iy++)
00876       for (ix = 0; ix < met->nx; ix++) {
00877         dest[ix][iy][ip] = scl * help[n++];
00878         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00879           dest[ix][iy][ip] = GSL_NAN;
00880       }
00881 }
```

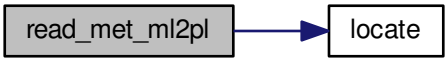
5.11.2.22 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 885 of file libtrac.c.

```
00888                         {
00889
00890   double aux[EP], p[EP], pt;
00891
00892   int ip, ip2, ix, iy;
00893
00894   /* Loop over columns... */
00895   for (ix = 0; ix < met->nx; ix++)
00896     for (iy = 0; iy < met->ny; iy++) {
00897
00898       /* Copy pressure profile... */
00899       for (ip = 0; ip < met->np; ip++)
00900         p[ip] = met->pl[ix][iy][ip];
00901
00902       /* Interpolate... */
00903       for (ip = 0; ip < ctl->met_np; ip++) {
00904         pt = ctl->met_p[ip];
00905         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00906           pt = p[0];
00907         else if ((pt > p[met->np - 1] && p[1] > p[0])
00908                  || (pt < p[met->np - 1] && p[1] < p[0]))
00909           pt = p[met->np - 1];
00910         ip2 = locate(p, met->np, pt);
00911         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00912                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00913       }
00914
00915       /* Copy data... */
00916       for (ip = 0; ip < ctl->met_np; ip++)
00917         var[ix][iy][ip] = (float) aux[ip];
00918     }
00919 }
```

Here is the call graph for this function:



5.11.2.23 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 923 of file libtrac.c.

```
00924                {
00925
00926   int ip, iy;
00927
00928   /* Check longitudes... */
00929   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00930              + met->lon[1] - met->lon[0] - 360) < 0.01))
00931     return;
00932
00933   /* Increase longitude counter... */
00934   if ((++met->nx) > EX)
00935     ERRMSG("Cannot create periodic boundary conditions!");
00936
00937   /* Set longitude... */
00938   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
00939
00940   /* Loop over latitudes and pressure levels... */
00941   for (iy = 0; iy < met->ny; iy++)
00942     for (ip = 0; ip < met->np; ip++) {
00943       met->ps[met->nx - 1][iy] = met->ps[0][iy];
00944       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
00945       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00946       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00947       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00948       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00949       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00950     }
00951 }
```

5.11.2.24 double scan_ctl (const char * *filename*, int *argc*, char * *argv[]*, const char * *varname*, int *arridx*, const char * *defvalue*, char * *value*)

Read a control parameter from file or command line.

Definition at line 955 of file libtrac.c.

```
00962                {
00963
00964   FILE *in = NULL;
00965
00966   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00967     msg[LEN], rvarname[LEN], rval[LEN];
00968
00969   int contain = 0, i;
00970
00971   /* Open file... */
00972   if (filename[strlen(filename) - 1] != '-')
00973     if (!(in = fopen(filename, "r")))
00974       ERRMSG("Cannot open file!");
00975
00976   /* Set full variable name... */
00977   if (arridx >= 0) {
00978     sprintf(fullname1, "%s[%d]", varname, arridx);
00979     sprintf(fullname2, "%s[*]", varname);
00980   } else {
00981     sprintf(fullname1, "%s", varname);
00982     sprintf(fullname2, "%s", varname);
00983   }
00984
00985   /* Read data... */
00986   if (in != NULL)
00987     while (fgets(line, LEN, in))
00988       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00989         if (strcasecmp(rvarname, fullname1) == 0 ||
00990             strcasecmp(rvarname, fullname2) == 0) {
00991           contain = 1;
00992           break;
00993         }
00994   for (i = 1; i < argc - 1; i++)
```

```
00995     if (strcasecmp(argv[i], fullname1) == 0 ||
00996         strcasecmp(argv[i], fullname2) == 0) {
00997       sprintf(rval, "%s", argv[i + 1]);
00998       contain = 1;
00999       break;
01000     }
01001
01002   /* Close file... */
01003   if (in != NULL)
01004     fclose(in);
01005
01006   /* Check for missing variables... */
01007   if (!contain) {
01008     if (strlen(defvalue) > 0)
01009       sprintf(rval, "%s", defvalue);
01010     else {
01011       sprintf(msg, "Missing variable %s!\n", fullname1);
01012       ERRMSG(msg);
01013     }
01014   }
01015
01016   /* Write info... */
01017   printf("%s = %s\n", fullname1, rval);
01018
01019   /* Return values... */
01020   if (value != NULL)
01021     sprintf(value, "%s", rval);
01022   return atof(rval);
01023 }
```

5.11.2.25 void time2jsec (int *year*, int *mon*, int *day*, int *hour*, int *min*, int *sec*, double *remain*, double * *jsec*)

Convert date to seconds.

Definition at line 1027 of file libtrac.c.

```
01035                 {
01036
01037   struct tm t0, t1;
01038
01039   t0.tm_year = 100;
01040   t0.tm_mon = 0;
01041   t0.tm_mday = 1;
01042   t0.tm_hour = 0;
01043   t0.tm_min = 0;
01044   t0.tm_sec = 0;
01045
01046   t1.tm_year = year - 1900;
01047   t1.tm_mon = mon - 1;
01048   t1.tm_mday = day;
01049   t1.tm_hour = hour;
01050   t1.tm_min = min;
01051   t1.tm_sec = sec;
01052
01053   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01054 }
```

5.11.2.26 void timer (const char * *name*, int *id*, int *mode*)

Measure wall-clock time.

Definition at line 1058 of file libtrac.c.

```
01061             {
01062
01063   static double starttime[NTIMER], runtime[NTIMER];
01064
01065   /* Check id... */
01066   if (id < 0 || id >= NTIMER)
01067     ERRMSG("Too many timers!");
01068
01069   /* Start timer... */
01070   if (mode == 1) {
01071     if (starttime[id] <= 0)
```

```
01072       starttime[id] = omp_get_wtime();
01073     else
01074       ERRMSG("Timer already started!");
01075   }
01076
01077   /* Stop timer... */
01078   else if (mode == 2) {
01079     if (starttime[id] > 0) {
01080       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01081       starttime[id] = -1;
01082     } else
01083       ERRMSG("Timer not started!");
01084   }
01085
01086   /* Print timer... */
01087   else if (mode == 3)
01088     printf("%s = %g s\n", name, runtime[id]);
01089 }
```

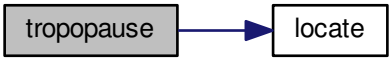
5.11.2.27 double tropopause (double *t*, double *lat*)

Definition at line 1093 of file libtrac.c.

```
01095               {
01096
01097   static double doys[12]
01098   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01099
01100   static double lats[73]
01101     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01102     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01103     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01104     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01105     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01106     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01107     75, 77.5, 80, 82.5, 85, 87.5, 90
01108   };
01109
01110   static double tps[12][73]
01111     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01112          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01113          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01114          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01115          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01116          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01117          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01118          275.3, 275.6, 275.4, 274.1, 273.5},
01119   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01120    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01121    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01122    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01123    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01124    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01125    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01126    287.5, 286.2, 285.8},
01127   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01128    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01129    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01130    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01131    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01132    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01133    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01134    304.3, 304.9, 306, 306.6, 306.2, 306},
01135   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01136    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01137    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01138    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01139    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01140    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01141    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01142    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01143   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01144    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01145    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01146    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01147    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01148    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01149    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01150    325.3, 325.8, 325.8},
01151   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01152    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
```

```
01153     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01154     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01155     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01156     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01157     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01158     308.5, 312.2, 313.1, 313.3},
01159    {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01160     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01161     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01162     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01163     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01164     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01165     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01166     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01167    {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01168     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01169     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01170     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01171     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01172     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01173     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01174     278.2, 282.6, 287.4, 290.9, 292.5, 293},
01175    {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01176     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01177     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01178     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01179     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01180     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01181     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01182     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01183    {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01184     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01185     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01186     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01187     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01188     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01189     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01190     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01191     305.1},
01192    {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01193     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01194     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01195     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01196     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01197     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01198     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01199     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01200    {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01201     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01202     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01203     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01204     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01205     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01206     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01207     281.7, 281.1, 281.2}
01208    };
01209
01210    double doy, p0, p1, pt;
01211
01212    int imon, ilat;
01213
01214    /* Get day of year... */
01215    doy = fmod(t / 86400., 365.25);
01216    while (doy < 0)
01217      doy += 365.25;
01218
01219    /* Get indices... */
01220    imon = locate(doys, 12, doy);
01221    ilat = locate(lats, 73, lat);
01222
01223    /* Get tropopause pressure... */
01224    p0 = LIN(lats[ilat], tps[imon][ilat],
01225             lats[ilat + 1], tps[imon][ilat + 1], lat);
01226    p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01227             lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01228    pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01229
01230    /* Return tropopause pressure... */
01231    return pt;
01232  }
```

Here is the call graph for this function:



5.11.2.28 void write_atm (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

Write atmospheric data.

Definition at line 1236 of file libtrac.c.

```
01240            {
01241
01242   FILE *in, *out;
01243
01244   char line[LEN];
01245
01246   double r, t0, t1;
01247
01248   int ip, iq, year, mon, day, hour, min, sec;
01249
01250   /* Set time interval for output... */
01251   t0 = t - 0.5 * ctl->dt_mod;
01252   t1 = t + 0.5 * ctl->dt_mod;
01253
01254   /* Check if gnuplot output is requested... */
01255   if (ctl->atm_gpfile[0] != '-') {
01256
01257     /* Write info... */
01258     printf("Plot atmospheric data: %s.png\n", filename);
01259
01260     /* Create gnuplot pipe... */
01261     if (!(out = popen("gnuplot", "w")))
01262       ERRMSG("Cannot create pipe to gnuplot!");
01263
01264     /* Set plot filename... */
01265     fprintf(out, "set out \"%s.png\"\n", filename);
01266
01267     /* Set time string... */
01268     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01269     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01270             year, mon, day, hour, min);
01271
01272     /* Dump gnuplot file to pipe... */
01273     if (!(in = fopen(ctl->atm_gpfile, "r")))
01274       ERRMSG("Cannot open file!");
01275     while (fgets(line, LEN, in))
01276       fprintf(out, "%s", line);
01277     fclose(in);
01278   }
01279
01280   else {
01281
01282     /* Write info... */
01283     printf("Write atmospheric data: %s\n", filename);
01284
01285     /* Create file... */
01286     if (!(out = fopen(filename, "w")))
01287       ERRMSG("Cannot create file!");
01288   }
01289
01290   /* Write header... */
01291   fprintf(out,
01292           "# $1 = time [s]\n"
01293           "# $2 = altitude [km]\n"
01294           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01295   for (iq = 0; iq < ctl->nq; iq++)
01296     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
```

```
01297             ctl->qnt_unit[iq]);
01298   fprintf(out, "\n");
01299
01300   /* Write data... */
01301   for (ip = 0; ip < atm->np; ip++) {
01302
01303     /* Check time... */
01304     if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01305       continue;
01306
01307     /* Write output... */
01308     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01309             atm->lon[ip], atm->lat[ip]);
01310     for (iq = 0; iq < ctl->nq; iq++) {
01311       fprintf(out, " ");
01312       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01313     }
01314     fprintf(out, "\n");
01315   }
01316
01317   /* Close file... */
01318   fclose(out);
01319 }
```

Here is the call graph for this function:

5.11.2.29 void write_csi (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

Write CSI data.

Definition at line 1323 of file libtrac.c.

```
01327             {
01328
01329   static FILE *in, *out;
01330
01331   static char line[LEN];
01332
01333   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01334     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01335
01336   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01337
01338   /* Init... */
01339   if (!init) {
01340     init = 1;
01341
01342     /* Check quantity index for mass... */
01343     if (ctl->qnt_m < 0)
01344       ERRMSG("Need quantity mass to analyze CSI!");
01345
01346     /* Open observation data file... */
01347     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01348     if (!(in = fopen(ctl->csi_obsfile, "r")))
01349       ERRMSG("Cannot open file!");
01350
01351     /* Create new file... */
01352     printf("Write CSI data: %s\n", filename);
01353     if (!(out = fopen(filename, "w")))
01354       ERRMSG("Cannot create file!");
01355
01356     /* Write header... */
```

```
01357     fprintf(out,
01358             "# $1 = time [s]\n"
01359             "# $2 = number of hits (cx)\n"
01360             "# $3 = number of misses (cy)\n"
01361             "# $4 = number of false alarms (cz)\n"
01362             "# $5 = number of observations (cx + cy)\n"
01363             "# $6 = number of forecasts (cx + cz)\n"
01364             "# $7 = bias (forecasts/observations) [%%]\n"
01365             "# $8 = probability of detection (POD) [%%]\n"
01366             "# $9 = false alarm rate (FAR) [%%]\n"
01367             "# $10 = critical success index (CSI) [%%]\n\n");
01368   }
01369
01370   /* Set time interval... */
01371   t0 = t - 0.5 * ctl->dt_mod;
01372   t1 = t + 0.5 * ctl->dt_mod;
01373
01374   /* Initialize grid cells... */
01375   for (ix = 0; ix < ctl->csi_nx; ix++)
01376     for (iy = 0; iy < ctl->csi_ny; iy++)
01377       for (iz = 0; iz < ctl->csi_nz; iz++)
01378         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01379
01380   /* Read data... */
01381   while (fgets(line, LEN, in)) {
01382
01383     /* Read data... */
01384     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01385         5)
01386       continue;
01387
01388     /* Check time... */
01389     if (rt < t0)
01390       continue;
01391     if (rt > t1)
01392       break;
01393
01394     /* Calculate indices... */
01395     ix = (int) ((rlon - ctl->csi_lon0)
01396                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01397     iy = (int) ((rlat - ctl->csi_lat0)
01398                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01399     iz = (int) ((rz - ctl->csi_z0)
01400                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01401
01402     /* Check indices... */
01403     if (ix < 0 || ix >= ctl->csi_nx ||
01404         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01405       continue;
01406
01407     /* Get mean observation index... */
01408     obsmean[ix][iy][iz] += robs;
01409     obscount[ix][iy][iz]++;
01410   }
01411
01412   /* Analyze model data... */
01413   for (ip = 0; ip < atm->np; ip++) {
01414
01415     /* Check time... */
01416     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01417       continue;
01418
01419     /* Get indices... */
01420     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01421                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01422     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01423                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01424     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01425                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01426
01427     /* Check indices... */
01428     if (ix < 0 || ix >= ctl->csi_nx ||
01429         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01430       continue;
01431
01432     /* Get total mass in grid cell... */
01433     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01434   }
01435
01436   /* Analyze all grid cells... */
01437   for (ix = 0; ix < ctl->csi_nx; ix++)
01438     for (iy = 0; iy < ctl->csi_ny; iy++)
01439       for (iz = 0; iz < ctl->csi_nz; iz++) {
01440
01441         /* Calculate mean observation index... */
01442         if (obscount[ix][iy][iz] > 0)
01443           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
```

```
01444
01445         /* Calculate column density... */
01446         if (modmean[ix][iy][iz] > 0) {
01447           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01448           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01449           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01450           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01451             * cos(lat * M_PI / 180.);
01452           modmean[ix][iy][iz] /= (1e6 * area);
01453         }
01454
01455         /* Calculate CSI... */
01456         if (obscount[ix][iy][iz] > 0) {
01457           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01458               modmean[ix][iy][iz] >= ctl->csi_modmin)
01459             cx++;
01460           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01461                    modmean[ix][iy][iz] < ctl->csi_modmin)
01462             cy++;
01463           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01464                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01465             cz++;
01466         }
01467       }
01468
01469   /* Write output... */
01470   if (fmod(t, ctl->csi_dt_out) == 0) {
01471
01472     /* Write... */
01473     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01474             t, cx, cy, cz, cx + cy, cx + cz,
01475             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01476             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01477             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01478             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01479
01480     /* Set counters to zero... */
01481     cx = cy = cz = 0;
01482   }
01483
01484   /* Close file... */
01485   if (t == ctl->t_stop)
01486     fclose(out);
01487 }
```

5.11.2.30 void write_ens (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

Write ensemble data.

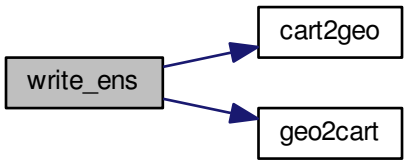
Definition at line 1491 of file libtrac.c.

```
01495             {
01496
01497   static FILE *out;
01498
01499   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01500     t0, t1, x[NENS][3], xm[3];
01501
01502   static int init, ip, iq;
01503
01504   static size_t i, n;
01505
01506   /* Init... */
01507   if (!init) {
01508     init = 1;
01509
01510     /* Check quantities... */
01511     if (ctl->qnt_ens < 0)
01512       ERRMSG("Missing ensemble IDs!");
01513
01514     /* Create new file... */
01515     printf("Write ensemble data: %s\n", filename);
01516     if (!(out = fopen(filename, "w")))
01517       ERRMSG("Cannot create file!");
01518
01519     /* Write header... */
01520     fprintf(out,
01521             "# $1 = time [s]\n"
01522             "# $2 = altitude [km]\n"
01523             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
```

```
01524     for (iq = 0; iq < ctl->nq; iq++)
01525       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01526               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01527     for (iq = 0; iq < ctl->nq; iq++)
01528       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01529               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01530     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01531   }
01532
01533   /* Set time interval... */
01534   t0 = t - 0.5 * ctl->dt_mod;
01535   t1 = t + 0.5 * ctl->dt_mod;
01536
01537   /* Init... */
01538   ens = GSL_NAN;
01539   n = 0;
01540
01541   /* Loop over air parcels... */
01542   for (ip = 0; ip < atm->np; ip++) {
01543
01544     /* Check time... */
01545     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01546       continue;
01547
01548     /* Check ensemble id... */
01549     if (atm->q[ctl->qnt_ens][ip] != ens) {
01550
01551       /* Write results... */
01552       if (n > 0) {
01553
01554         /* Get mean position... */
01555         xm[0] = xm[1] = xm[2] = 0;
01556         for (i = 0; i < n; i++) {
01557           xm[0] += x[i][0] / (double) n;
01558           xm[1] += x[i][1] / (double) n;
01559           xm[2] += x[i][2] / (double) n;
01560         }
01561         cart2geo(xm, &dummy, &lon, &lat);
01562         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01563                 lat);
01564
01565         /* Get quantity statistics... */
01566         for (iq = 0; iq < ctl->nq; iq++) {
01567           fprintf(out, " ");
01568           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01569         }
01570         for (iq = 0; iq < ctl->nq; iq++) {
01571           fprintf(out, " ");
01572           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01573         }
01574         fprintf(out, " %lu\n", n);
01575       }
01576
01577       /* Init new ensemble... */
01578       ens = atm->q[ctl->qnt_ens][ip];
01579       n = 0;
01580     }
01581
01582     /* Save data... */
01583     p[n] = atm->p[ip];
01584     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01585     for (iq = 0; iq < ctl->nq; iq++)
01586       q[iq][n] = atm->q[iq][ip];
01587     if ((++n) >= NENS)
01588       ERRMSG("Too many data points!");
01589   }
01590
01591   /* Write results... */
01592   if (n > 0) {
01593
01594     /* Get mean position... */
01595     xm[0] = xm[1] = xm[2] = 0;
01596     for (i = 0; i < n; i++) {
01597       xm[0] += x[i][0] / (double) n;
01598       xm[1] += x[i][1] / (double) n;
01599       xm[2] += x[i][2] / (double) n;
01600     }
01601     cart2geo(xm, &dummy, &lon, &lat);
01602     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01603
01604     /* Get quantity statistics... */
01605     for (iq = 0; iq < ctl->nq; iq++) {
01606       fprintf(out, " ");
01607       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01608     }
01609     for (iq = 0; iq < ctl->nq; iq++) {
01610       fprintf(out, " ");
```

```
01611       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01612     }
01613     fprintf(out, " %lu\n", n);
01614   }
01615
01616   /* Close file... */
01617   if (t == ctl->t_stop)
01618     fclose(out);
01619 }
```

Here is the call graph for this function:



5.11.2.31 void write_grid (const char * *filename*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

Write gridded data.

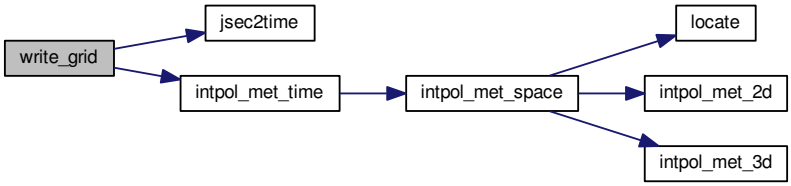
Definition at line 1623 of file libtrac.c.

```
01629             {
01630
01631   FILE *in, *out;
01632
01633   char line[LEN];
01634
01635   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01636     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01637
01638   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01639
01640   /* Check dimensions... */
01641   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01642     ERRMSG("Grid dimensions too large!");
01643
01644   /* Check quantity index for mass... */
01645   if (ctl->qnt_m < 0)
01646     ERRMSG("Need quantity mass to write grid data!");
01647
01648   /* Set time interval for output... */
01649   t0 = t - 0.5 * ctl->dt_mod;
01650   t1 = t + 0.5 * ctl->dt_mod;
01651
01652   /* Set grid box size... */
01653   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01654   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01655   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01656
01657   /* Initialize grid... */
01658   for (ix = 0; ix < ctl->grid_nx; ix++)
01659     for (iy = 0; iy < ctl->grid_ny; iy++)
01660       for (iz = 0; iz < ctl->grid_nz; iz++)
01661         grid_m[ix][iy][iz] = 0;
01662
01663   /* Average data... */
01664   for (ip = 0; ip < atm->np; ip++)
01665     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01666
01667       /* Get index... */
01668       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
```

```
01669       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01670       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01671
01672       /* Check indices... */
01673       if (ix < 0 || ix >= ctl->grid_nx ||
01674           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01675         continue;
01676
01677       /* Add mass... */
01678       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01679     }
01680
01681   /* Check if gnuplot output is requested... */
01682   if (ctl->grid_gpfile[0] != '-') {
01683
01684     /* Write info... */
01685     printf("Plot grid data: %s.png\n", filename);
01686
01687     /* Create gnuplot pipe... */
01688     if (!(out = popen("gnuplot", "w")))
01689       ERRMSG("Cannot create pipe to gnuplot!");
01690
01691     /* Set plot filename... */
01692     fprintf(out, "set out \"%s.png\"\n", filename);
01693
01694     /* Set time string... */
01695     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01696     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01697             year, mon, day, hour, min);
01698
01699     /* Dump gnuplot file to pipe... */
01700     if (!(in = fopen(ctl->grid_gpfile, "r")))
01701       ERRMSG("Cannot open file!");
01702     while (fgets(line, LEN, in))
01703       fprintf(out, "%s", line);
01704     fclose(in);
01705   }
01706
01707   else {
01708
01709     /* Write info... */
01710     printf("Write grid data: %s\n", filename);
01711
01712     /* Create file... */
01713     if (!(out = fopen(filename, "w")))
01714       ERRMSG("Cannot create file!");
01715   }
01716
01717   /* Write header... */
01718   fprintf(out,
01719           "# $1 = time [s]\n"
01720           "# $2 = altitude [km]\n"
01721           "# $3 = longitude [deg]\n"
01722           "# $4 = latitude [deg]\n"
01723           "# $5 = surface area [km^2]\n"
01724           "# $6 = layer width [km]\n"
01725           "# $7 = temperature [K]\n"
01726           "# $8 = column density [kg/m^2]\n"
01727           "# $9 = mass mixing ratio [1]\n\n");
01728
01729   /* Write data... */
01730   for (ix = 0; ix < ctl->grid_nx; ix++) {
01731     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01732       fprintf(out, "\n");
01733     for (iy = 0; iy < ctl->grid_ny; iy++) {
01734       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01735         fprintf(out, "\n");
01736       for (iz = 0; iz < ctl->grid_nz; iz++)
01737         if (!ctl->grid_sparse
01738             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01739
01740           /* Set coordinates... */
01741           z = ctl->grid_z0 + dz * (iz + 0.5);
01742           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01743           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01744
01745           /* Get pressure and temperature... */
01746           press = P(z);
01747           intpol_met_time(met0, met1, t, press, lon, lat,
01748                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01749
01750           /* Calculate surface area... */
01751           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01752             * cos(lat * M_PI / 180.);
01753
01754           /* Calculate column density... */
01755           cd = grid_m[ix][iy][iz] / (1e6 * area);
```

```
01756
01757           /* Calculate mass mixing ratio... */
01758           rho_air = 100. * press / (287.058 * temp);
01759           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01760
01761           /* Write output... */
01762           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01763                   t, z, lon, lat, area, dz, temp, cd, mmr);
01764         }
01765     }
01766   }
01767
01768   /* Close file... */
01769   fclose(out);
01770 }
```

Here is the call graph for this function:



5.11.2.32 void write_prof (const char * *filename*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

Write profile data.

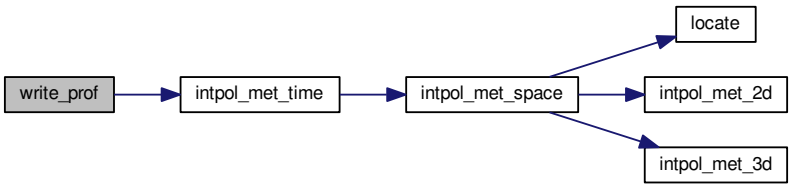
Definition at line 1774 of file libtrac.c.

```
01780           {
01781
01782   static FILE *in, *out;
01783
01784   static char line[LEN];
01785
01786   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01787     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01788     press, temp, rho_air, mmr, h2o, o3;
01789
01790   static int init, obscount[GX][GY], ip, ix, iy, iz;
01791
01792   /* Init... */
01793   if (!init) {
01794     init = 1;
01795
01796     /* Check quantity index for mass... */
01797     if (ctl->qnt_m < 0)
01798       ERRMSG("Need quantity mass!");
01799
01800     /* Check dimensions... */
01801     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01802       ERRMSG("Grid dimensions too large!");
01803
01804     /* Open observation data file... */
01805     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01806     if (!(in = fopen(ctl->prof_obsfile, "r")))
01807       ERRMSG("Cannot open file!");
01808
01809     /* Create new file... */
01810     printf("Write profile data: %s\n", filename);
01811     if (!(out = fopen(filename, "w")))
01812       ERRMSG("Cannot create file!");
01813
01814     /* Write header... */
01815     fprintf(out,
```

```
01816             "# $1  = time [s]\n"
01817             "# $2  = altitude [km]\n"
01818             "# $3  = longitude [deg]\n"
01819             "# $4  = latitude [deg]\n"
01820             "# $5  = pressure [hPa]\n"
01821             "# $6  = temperature [K]\n"
01822             "# $7  = mass mixing ratio [1]\n"
01823             "# $8  = H2O volume mixing ratio [1]\n"
01824             "# $9  = O3 volume mixing ratio [1]\n"
01825             "# $10 = mean BT index [K]\n");
01826
01827     /* Set grid box size... */
01828     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01829     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01830     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01831   }
01832
01833   /* Set time interval... */
01834   t0 = t - 0.5 * ctl->dt_mod;
01835   t1 = t + 0.5 * ctl->dt_mod;
01836
01837   /* Initialize... */
01838   for (ix = 0; ix < ctl->prof_nx; ix++)
01839     for (iy = 0; iy < ctl->prof_ny; iy++) {
01840       obsmean[ix][iy] = 0;
01841       obscount[ix][iy] = 0;
01842       tmean[ix][iy] = 0;
01843       for (iz = 0; iz < ctl->prof_nz; iz++)
01844         mass[ix][iy][iz] = 0;
01845     }
01846
01847   /* Read data... */
01848   while (fgets(line, LEN, in)) {
01849
01850     /* Read data... */
01851     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01852       continue;
01853
01854     /* Check time... */
01855     if (rt < t0)
01856       continue;
01857     if (rt > t1)
01858       break;
01859
01860     /* Calculate indices... */
01861     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01862     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01863
01864     /* Check indices... */
01865     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01866       continue;
01867
01868     /* Get mean observation index... */
01869     obsmean[ix][iy] += robs;
01870     tmean[ix][iy] += rt;
01871     obscount[ix][iy]++;
01872   }
01873
01874   /* Analyze model data... */
01875   for (ip = 0; ip < atm->np; ip++) {
01876
01877     /* Check time... */
01878     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01879       continue;
01880
01881     /* Get indices... */
01882     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01883     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01884     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01885
01886     /* Check indices... */
01887     if (ix < 0 || ix >= ctl->prof_nx ||
01888         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01889       continue;
01890
01891     /* Get total mass in grid cell... */
01892     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01893   }
01894
01895   /* Extract profiles... */
01896   for (ix = 0; ix < ctl->prof_nx; ix++)
01897     for (iy = 0; iy < ctl->prof_ny; iy++)
01898       if (obscount[ix][iy] > 0) {
01899
01900         /* Write output... */
01901         fprintf(out, "\n");
01902
```

```
01903         /* Loop over altitudes... */
01904         for (iz = 0; iz < ctl->prof_nz; iz++) {
01905
01906           /* Set coordinates... */
01907           z = ctl->prof_z0 + dz * (iz + 0.5);
01908           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01909           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01910
01911           /* Get meteorological data... */
01912           press = P(z);
01913           intpol_met_time(met0, met1, t, press, lon, lat,
01914                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01915
01916           /* Calculate mass mixing ratio... */
01917           rho_air = 100. * press / (287.058 * temp);
01918           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
01919             * cos(lat * M_PI / 180.);
01920           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01921
01922           /* Write output... */
01923           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
01924                   tmean[ix][iy] / obscount[ix][iy],
01925                   z, lon, lat, press, temp, mmr, h2o, o3,
01926                   obsmean[ix][iy] / obscount[ix][iy]);
01927         }
01928       }
01929
01930   /* Close file... */
01931   if (t == ctl->t_stop)
01932     fclose(out);
01933 }
```

Here is the call graph for this function:



5.11.2.33 void write_station (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

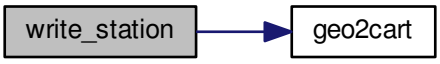
Write station data.

Definition at line 1937 of file libtrac.c.

```
01941             {
01942
01943   static FILE *out;
01944
01945   static double rmax2, t0, t1, x0[3], x1[3];
01946
01947   static int init, ip, iq;
01948
01949   /* Init... */
01950   if (!init) {
01951     init = 1;
01952
01953     /* Write info... */
01954     printf("Write station data: %s\n", filename);
01955
01956     /* Create new file... */
01957     if (!(out = fopen(filename, "w")))
01958       ERRMSG("Cannot create file!");
01959
01960     /* Write header... */
```

```
01961     fprintf(out,
01962             "# $1 = time [s]\n"
01963             "# $2 = altitude [km]\n"
01964             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01965     for (iq = 0; iq < ctl->nq; iq++)
01966       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01967               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01968     fprintf(out, "\n");
01969
01970     /* Set geolocation and search radius... */
01971     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01972     rmax2 = gsl_pow_2(ctl->stat_r);
01973   }
01974
01975   /* Set time interval for output... */
01976   t0 = t - 0.5 * ctl->dt_mod;
01977   t1 = t + 0.5 * ctl->dt_mod;
01978
01979   /* Loop over air parcels... */
01980   for (ip = 0; ip < atm->np; ip++) {
01981
01982     /* Check time... */
01983     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01984       continue;
01985
01986     /* Check station flag... */
01987     if (ctl->qnt_stat >= 0)
01988       if (atm->q[ctl->qnt_stat][ip])
01989         continue;
01990
01991     /* Get Cartesian coordinates... */
01992     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01993
01994     /* Check horizontal distance... */
01995     if (DIST2(x0, x1) > rmax2)
01996       continue;
01997
01998     /* Set station flag... */
01999     if (ctl->qnt_stat >= 0)
02000       atm->q[ctl->qnt_stat][ip] = 1;
02001
02002     /* Write data... */
02003     fprintf(out, "%.2f %g %g %g",
02004             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02005     for (iq = 0; iq < ctl->nq; iq++) {
02006       fprintf(out, " ");
02007       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02008     }
02009     fprintf(out, "\n");
02010   }
02011
02012   /* Close file... */
02013   if (t == ctl->t_stop)
02014     fclose(out);
02015 }
```

Here is the call graph for this function:



5.12 libtrac.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
```

```
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /*****************************************************************************/
00028
00029 void cart2geo(
00030   double *x,
00031   double *z,
00032   double *lon,
00033   double *lat) {
00034
00035   double radius;
00036
00037   radius = NORM(x);
00038   *lat = asin(x[2] / radius) * 180 / M_PI;
00039   *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040   *z = radius - RE;
00041 }
00042
00043 /*****************************************************************************/
00044
00045 double deg2dx(
00046   double dlon,
00047   double lat) {
00048
00049   return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00050 }
00051
00052 /*****************************************************************************/
00053
00054 double deg2dy(
00055   double dlat) {
00056
00057   return dlat * M_PI * RE / 180.;
00058 }
00059
00060 /*****************************************************************************/
00061
00062 double dp2dz(
00063   double dp,
00064   double p) {
00065
00066   return -dp * H0 / p;
00067 }
00068
00069 /*****************************************************************************/
00070
00071 double dx2deg(
00072   double dx,
00073   double lat) {
00074
00075   /* Avoid singularity at poles... */
00076   if (lat < -89.999 || lat > 89.999)
00077     return 0;
00078   else
00079     return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00080 }
00081
00082 /*****************************************************************************/
00083
00084 double dy2deg(
00085   double dy) {
00086
00087   return dy * 180. / (M_PI * RE);
00088 }
00089
00090 /*****************************************************************************/
00091
00092 double dz2dp(
00093   double dz,
00094   double p) {
00095
00096   return -dz * p / H0;
00097 }
```

```
00098
00099 /*****************************************************************************/
00100
00101 void geo2cart(
00102   double z,
00103   double lon,
00104   double lat,
00105   double *x) {
00106
00107   double radius;
00108
00109   radius = z + RE;
00110   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00111   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00112   x[2] = radius * sin(lat / 180 * M_PI);
00113 }
00114
00115 /*****************************************************************************/
00116
00117 void get_met(
00118   ctl_t * ctl,
00119   char *metbase,
00120   double t,
00121   met_t * met0,
00122   met_t * met1) {
00123
00124   char filename[LEN];
00125
00126   static int init;
00127
00128   /* Init... */
00129   if (!init) {
00130     init = 1;
00131
00132     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00133     read_met(ctl, filename, met0);
00134
00135     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00136     read_met(ctl, filename, met1);
00137   }
00138
00139   /* Read new data for forward trajectories... */
00140   if (t > met1->time && ctl->direction == 1) {
00141     memcpy(met0, met1, sizeof(met_t));
00142     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00143     read_met(ctl, filename, met1);
00144   }
00145
00146   /* Read new data for backward trajectories... */
00147   if (t < met0->time && ctl->direction == -1) {
00148     memcpy(met1, met0, sizeof(met_t));
00149     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00150     read_met(ctl, filename, met0);
00151   }
00152 }
00153
00154 /*****************************************************************************/
00155
00156 void get_met_help(
00157   double t,
00158   int direct,
00159   char *metbase,
00160   double dt_met,
00161   char *filename) {
00162
00163   double t6, r;
00164
00165   int year, mon, day, hour, min, sec;
00166
00167   /* Round time to fixed intervals... */
00168   if (direct == -1)
00169     t6 = floor(t / dt_met) * dt_met;
00170   else
00171     t6 = ceil(t / dt_met) * dt_met;
00172
00173   /* Decode time... */
00174   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
00176   /* Set filename... */
00177   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00178 }
00179
00180 /*****************************************************************************/
00181
00182 void intpol_met_2d(
00183   double array[EX][EY],
```

```
00184   int ix,
00185   int iy,
00186   double wx,
00187   double wy,
00188   double *var) {
00189
00190   double aux00, aux01, aux10, aux11;
00191
00192   /* Set variables... */
00193   aux00 = array[ix][iy];
00194   aux01 = array[ix][iy + 1];
00195   aux10 = array[ix + 1][iy];
00196   aux11 = array[ix + 1][iy + 1];
00197
00198   /* Interpolate horizontally... */
00199   aux00 = wy * (aux00 - aux01) + aux01;
00200   aux11 = wy * (aux10 - aux11) + aux11;
00201   *var = wx * (aux00 - aux11) + aux11;
00202 }
00203
00204 /*****************************************************************************/
00205
00206 void intpol_met_3d(
00207   float array[EX][EY][EP],
00208   int ip,
00209   int ix,
00210   int iy,
00211   double wp,
00212   double wx,
00213   double wy,
00214   double *var) {
00215
00216   double aux00, aux01, aux10, aux11;
00217
00218   /* Interpolate vertically... */
00219   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00220     + array[ix][iy][ip + 1];
00221   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00222     + array[ix][iy + 1][ip + 1];
00223   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00224     + array[ix + 1][iy][ip + 1];
00225   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00226     + array[ix + 1][iy + 1][ip + 1];
00227
00228   /* Interpolate horizontally... */
00229   aux00 = wy * (aux00 - aux01) + aux01;
00230   aux11 = wy * (aux10 - aux11) + aux11;
00231   *var = wx * (aux00 - aux11) + aux11;
00232 }
00233
00234 /*****************************************************************************/
00235
00236 void intpol_met_space(
00237   met_t * met,
00238   double p,
00239   double lon,
00240   double lat,
00241   double *ps,
00242   double *t,
00243   double *u,
00244   double *v,
00245   double *w,
00246   double *h2o,
00247   double *o3) {
00248
00249   double wp, wx, wy;
00250
00251   int ip, ix, iy;
00252
00253   /* Check longitude... */
00254   if (met->lon[met->nx - 1] > 180 && lon < 0)
00255     lon += 360;
00256
00257   /* Get indices... */
00258   ip = locate(met->p, met->np, p);
00259   ix = locate(met->lon, met->nx, lon);
00260   iy = locate(met->lat, met->ny, lat);
00261
00262   /* Get weights... */
00263   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00264   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00265   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00266
00267   /* Interpolate... */
00268   if (ps != NULL)
00269     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00270   if (t != NULL)
```

```
00271     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00272   if (u != NULL)
00273     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00274   if (v != NULL)
00275     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00276   if (w != NULL)
00277     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00278   if (h2o != NULL)
00279     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00280   if (o3 != NULL)
00281     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
00283
00284 /*****************************************************************************/
00285
00286 void intpol_met_time(
00287   met_t * met0,
00288   met_t * met1,
00289   double ts,
00290   double p,
00291   double lon,
00292   double lat,
00293   double *ps,
00294   double *t,
00295   double *u,
00296   double *v,
00297   double *w,
00298   double *h2o,
00299   double *o3) {
00300
00301   double h2o0, h2o1, o30, o31, ps0, ps1, t0, t1, u0, u1, v0, v1, w0, w1, wt;
00302
00303   /* Spatial interpolation... */
00304   intpol_met_space(met0, p, lon, lat,
00305                    ps == NULL ? NULL : &ps0,
00306                    t == NULL ? NULL : &t0,
00307                    u == NULL ? NULL : &u0,
00308                    v == NULL ? NULL : &v0,
00309                    w == NULL ? NULL : &w0,
00310                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00311   intpol_met_space(met1, p, lon, lat,
00312                    ps == NULL ? NULL : &ps1,
00313                    t == NULL ? NULL : &t1,
00314                    u == NULL ? NULL : &u1,
00315                    v == NULL ? NULL : &v1,
00316                    w == NULL ? NULL : &w1,
00317                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
00319   /* Get weighting factor... */
00320   wt = (met1->time - ts) / (met1->time - met0->time);
00321
00322   /* Interpolate... */
00323   if (ps != NULL)
00324     *ps = wt * (ps0 - ps1) + ps1;
00325   if (t != NULL)
00326     *t = wt * (t0 - t1) + t1;
00327   if (u != NULL)
00328     *u = wt * (u0 - u1) + u1;
00329   if (v != NULL)
00330     *v = wt * (v0 - v1) + v1;
00331   if (w != NULL)
00332     *w = wt * (w0 - w1) + w1;
00333   if (h2o != NULL)
00334     *h2o = wt * (h2o0 - h2o1) + h2o1;
00335   if (o3 != NULL)
00336     *o3 = wt * (o30 - o31) + o31;
00337 }
00338
00339 /*****************************************************************************/
00340
00341 void jsec2time(
00342   double jsec,
00343   int *year,
00344   int *mon,
00345   int *day,
00346   int *hour,
00347   int *min,
00348   int *sec,
00349   double *remain) {
00350
00351   struct tm t0, *t1;
00352
00353   time_t jsec0;
00354
00355   t0.tm_year = 100;
00356   t0.tm_mon = 0;
00357   t0.tm_mday = 1;
```

```
00358   t0.tm_hour = 0;
00359   t0.tm_min = 0;
00360   t0.tm_sec = 0;
00361
00362   jsec0 = (time_t) jsec + timegm(&t0);
00363   t1 = gmtime(&jsec0);
00364
00365   *year = t1->tm_year + 1900;
00366   *mon = t1->tm_mon + 1;
00367   *day = t1->tm_mday;
00368   *hour = t1->tm_hour;
00369   *min = t1->tm_min;
00370   *sec = t1->tm_sec;
00371   *remain = jsec - floor(jsec);
00372 }
00373
00374 /*****************************************************************************/
00375
00376 int locate(
00377   double *xx,
00378   int n,
00379   double x) {
00380
00381   int i, ilo, ihi;
00382
00383   ilo = 0;
00384   ihi = n - 1;
00385   i = (ihi + ilo) >> 1;
00386
00387   if (xx[i] < xx[i + 1])
00388     while (ihi > ilo + 1) {
00389       i = (ihi + ilo) >> 1;
00390       if (xx[i] > x)
00391         ihi = i;
00392       else
00393         ilo = i;
00394   } else
00395     while (ihi > ilo + 1) {
00396       i = (ihi + ilo) >> 1;
00397       if (xx[i] <= x)
00398         ihi = i;
00399       else
00400         ilo = i;
00401     }
00402
00403   return ilo;
00404 }
00405
00406 /*****************************************************************************/
00407
00408 void read_atm(
00409   const char *filename,
00410   ctl_t * ctl,
00411   atm_t * atm) {
00412
00413   FILE *in;
00414
00415   char line[LEN], *tok;
00416
00417   int iq;
00418
00419   /* Init... */
00420   atm->np = 0;
00421
00422   /* Write info... */
00423   printf("Read atmospheric data: %s\n", filename);
00424
00425   /* Open file... */
00426   if (!(in = fopen(filename, "r")))
00427     ERRMSG("Cannot open file!");
00428
00429   /* Read line... */
00430   while (fgets(line, LEN, in)) {
00431
00432     /* Read data... */
00433     TOK(line, tok, "%lg", atm->time[atm->np]);
00434     TOK(NULL, tok, "%lg", atm->p[atm->np]);
00435     TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00436     TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00437     for (iq = 0; iq < ctl->nq; iq++)
00438       TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00439
00440     /* Convert altitude to pressure... */
00441     atm->p[atm->np] = P(atm->p[atm->np]);
00442
00443     /* Increment data point counter... */
00444     if ((++atm->np) > NP)
```

```
00445       ERRMSG("Too many data points!");
00446   }
00447
00448   /* Close file... */
00449   fclose(in);
00450
00451   /* Check number of points... */
00452   if (atm->np < 1)
00453     ERRMSG("Can not read any data!");
00454 }
00455
00456 /*****************************************************************************/
00457
00458 void read_ctl(
00459   const char *filename,
00460   int argc,
00461   char *argv[],
00462   ctl_t * ctl) {
00463
00464   int ip, iq;
00465
00466   /* Write info... */
00467   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00468          "(executable: %s | compiled: %s, %s)\n\n",
00469          argv[0], __DATE__, __TIME__);
00470
00471   /* Initialize quantity indices... */
00472   ctl->qnt_ens = -1;
00473   ctl->qnt_m = -1;
00474   ctl->qnt_r = -1;
00475   ctl->qnt_rho = -1;
00476   ctl->qnt_ps = -1;
00477   ctl->qnt_p = -1;
00478   ctl->qnt_t = -1;
00479   ctl->qnt_u = -1;
00480   ctl->qnt_v = -1;
00481   ctl->qnt_w = -1;
00482   ctl->qnt_h2o = -1;
00483   ctl->qnt_o3 = -1;
00484   ctl->qnt_theta = -1;
00485   ctl->qnt_pv = -1;
00486   ctl->qnt_tice = -1;
00487   ctl->qnt_tsts = -1;
00488   ctl->qnt_tnat = -1;
00489   ctl->qnt_stat = -1;
00490
00491   /* Read quantities... */
00492   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00493   for (iq = 0; iq < ctl->nq; iq++) {
00494
00495     /* Read quantity name and format... */
00496     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
00497     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
00498              ctl->qnt_format[iq]);
00499
00500     /* Try to identify quantity... */
00501     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00502       ctl->qnt_ens = iq;
00503       sprintf(ctl->qnt_unit[iq], "-");
00504     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00505       ctl->qnt_m = iq;
00506       sprintf(ctl->qnt_unit[iq], "kg");
00507     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
00508       ctl->qnt_r = iq;
00509       sprintf(ctl->qnt_unit[iq], "m");
00510     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00511       ctl->qnt_rho = iq;
00512       sprintf(ctl->qnt_unit[iq], "kg/m^3");
00513     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00514       ctl->qnt_ps = iq;
00515       sprintf(ctl->qnt_unit[iq], "hPa");
00516     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00517       ctl->qnt_p = iq;
00518       sprintf(ctl->qnt_unit[iq], "hPa");
00519     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00520       ctl->qnt_t = iq;
00521       sprintf(ctl->qnt_unit[iq], "K");
00522     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00523       ctl->qnt_u = iq;
00524       sprintf(ctl->qnt_unit[iq], "m/s");
00525     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00526       ctl->qnt_v = iq;
00527       sprintf(ctl->qnt_unit[iq], "m/s");
00528     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00529       ctl->qnt_w = iq;
00530       sprintf(ctl->qnt_unit[iq], "hPa/s");
00531     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
```

```
00532       ctl->qnt_h2o = iq;
00533       sprintf(ctl->qnt_unit[iq], "1");
00534     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00535       ctl->qnt_o3 = iq;
00536       sprintf(ctl->qnt_unit[iq], "1");
00537     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00538       ctl->qnt_theta = iq;
00539       sprintf(ctl->qnt_unit[iq], "K");
00540     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00541       ctl->qnt_pv = iq;
00542       sprintf(ctl->qnt_unit[iq], "PVU");
00543     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00544       ctl->qnt_tice = iq;
00545       sprintf(ctl->qnt_unit[iq], "K");
00546     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00547       ctl->qnt_tsts = iq;
00548       sprintf(ctl->qnt_unit[iq], "K");
00549     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00550       ctl->qnt_tnat = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00553       ctl->qnt_stat = iq;
00554       sprintf(ctl->qnt_unit[iq], "-");
00555     } else
00556       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00557   }
00558
00559   /* Time steps of simulation... */
00560   ctl->direction =
00561     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00562   if (ctl->direction != -1 && ctl->direction != 1)
00563     ERRMSG("Set DIRECTION to -1 or 1!");
00564   ctl->t_start =
00565     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00566   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00567   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00568
00569   /* Meteorological data... */
00570   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00571   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00572   if (ctl->met_np > EP)
00573     ERRMSG("Too many levels!");
00574   for (ip = 0; ip < ctl->met_np; ip++)
00575     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00576
00577   /* Isosurface parameters... */
00578   ctl->isosurf
00579     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00580   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00581
00582   /* Diffusion parameters... */
00583   ctl->turb_dx_trop
00584     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00585   ctl->turb_dx_strat
00586     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00587   ctl->turb_dz_trop
00588     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00589   ctl->turb_dz_strat
00590     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00591   ctl->turb_meso =
00592     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00593
00594   /* Life time of particles... */
00595   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00596   ctl->tdec_strat =
00597     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00598
00599   /* PSC analysis... */
00600   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00601   ctl->psc_hno3 =
00602     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00603
00604   /* Output of atmospheric data... */
00605   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00606   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00607   ctl->atm_dt_out =
00608     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00609   ctl->atm_filter =
00610     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00611
00612   /* Output of CSI data... */
00613   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00614   ctl->csi_dt_out =
00615     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
00616   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
```

```
00617             ctl->csi_obsfile);
00618   ctl->csi_obsmin =
00619     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00620   ctl->csi_modmin =
00621     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00622   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
00623   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
00624   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00625   ctl->csi_lon0 =
00626     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
00627   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00628   ctl->csi_nx =
00629     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
00630   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
00631   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
00632   ctl->csi_ny =
00633     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00634
00635   /* Output of ensemble data... */
00636   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
00637
00638   /* Output of grid data... */
00639   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00640             ctl->grid_basename);
00641   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00642   ctl->grid_dt_out =
00643     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00644   ctl->grid_sparse =
00645     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00646   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
00647   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00648   ctl->grid_nz =
00649     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00650   ctl->grid_lon0 =
00651     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
00652   ctl->grid_lon1 =
00653     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00654   ctl->grid_nx =
00655     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00656   ctl->grid_lat0 =
00657     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
00658   ctl->grid_lat1 =
00659     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00660   ctl->grid_ny =
00661     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00662
00663   /* Output of profile data... */
00664   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00665             ctl->prof_basename);
00666   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
00667   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00668   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00669   ctl->prof_nz =
00670     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00671   ctl->prof_lon0 =
00672     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
00673   ctl->prof_lon1 =
00674     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00675   ctl->prof_nx =
00676     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00677   ctl->prof_lat0 =
00678     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
00679   ctl->prof_lat1 =
00680     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00681   ctl->prof_ny =
00682     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00683
00684   /* Output of station data... */
00685   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00686             ctl->stat_basename);
00687   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
00688   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
00689   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00690 }
00691
00692 /*****************************************************************************/
00693
00694 void read_met(
00695   ctl_t * ctl,
00696   char *filename,
00697   met_t * met) {
00698
00699   char tstr[10];
00700
```

```
00701   static float help[EX * EY];
00702
00703   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00704
00705   size_t np, nx, ny;
00706
00707   /* Write info... */
00708   printf("Read meteorological data: %s\n", filename);
00709
00710   /* Get time from filename... */
00711   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00712   year = atoi(tstr);
00713   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00714   mon = atoi(tstr);
00715   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00716   day = atoi(tstr);
00717   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00718   hour = atoi(tstr);
00719   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00720
00721   /* Open netCDF file... */
00722   NC(nc_open(filename, NC_NOWRITE, &ncid));
00723
00724   /* Get dimensions... */
00725   NC(nc_inq_dimid(ncid, "lon", &dimid));
00726   NC(nc_inq_dimlen(ncid, dimid, &nx));
00727   if (nx > EX)
00728     ERRMSG("Too many longitudes!");
00729
00730   NC(nc_inq_dimid(ncid, "lat", &dimid));
00731   NC(nc_inq_dimlen(ncid, dimid, &ny));
00732   if (ny > EY)
00733     ERRMSG("Too many latitudes!");
00734
00735   NC(nc_inq_dimid(ncid, "lev", &dimid));
00736   NC(nc_inq_dimlen(ncid, dimid, &np));
00737   if (np > EP)
00738     ERRMSG("Too many levels!");
00739
00740   /* Store dimensions... */
00741   met->np = (int) np;
00742   met->nx = (int) nx;
00743   met->ny = (int) ny;
00744
00745   /* Get horizontal grid... */
00746   NC(nc_inq_varid(ncid, "lon", &varid));
00747   NC(nc_get_var_double(ncid, varid, met->lon));
00748   NC(nc_inq_varid(ncid, "lat", &varid));
00749   NC(nc_get_var_double(ncid, varid, met->lat));
00750
00751   /* Read meteorological data... */
00752   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00753   read_met_help(ncid, "u", "U", met, met->u, 1.0);
00754   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00755   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00756   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00757   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00758
00759   /* Meteo data on pressure levels... */
00760   if (ctl->met_np <= 0) {
00761
00762     /* Read pressure levels from file... */
00763     NC(nc_inq_varid(ncid, "lev", &varid));
00764     NC(nc_get_var_double(ncid, varid, met->p));
00765     for (ip = 0; ip < met->np; ip++)
00766       met->p[ip] /= 100.;
00767
00768     /* Extrapolate data for lower boundary... */
00769     read_met_extrapolate(met);
00770   }
00771
00772   /* Meteo data on model levels... */
00773   else {
00774
00775     /* Read pressure data from file... */
00776     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00777
00778     /* Interpolate from model levels to pressure levels... */
00779     read_met_ml2pl(ctl, met, met->t);
00780     read_met_ml2pl(ctl, met, met->u);
00781     read_met_ml2pl(ctl, met, met->v);
00782     read_met_ml2pl(ctl, met, met->w);
00783     read_met_ml2pl(ctl, met, met->h2o);
00784     read_met_ml2pl(ctl, met, met->o3);
00785
00786     /* Set pressure levels... */
00787     met->np = ctl->met_np;
```

```
00788     for (ip = 0; ip < met->np; ip++)
00789       met->p[ip] = ctl->met_p[ip];
00790   }
00791
00792   /* Check ordering of pressure levels... */
00793   for (ip = 1; ip < met->np; ip++)
00794     if (met->p[ip - 1] < met->p[ip])
00795       ERRMSG("Pressure levels must be descending!");
00796
00797   /* Read surface pressure... */
00798   if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00799     NC(nc_get_var_float(ncid, varid, help));
00800     for (iy = 0; iy < met->ny; iy++)
00801       for (ix = 0; ix < met->nx; ix++)
00802         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00803   } else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00804     NC(nc_get_var_float(ncid, varid, help));
00805     for (iy = 0; iy < met->ny; iy++)
00806       for (ix = 0; ix < met->nx; ix++)
00807         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00808   } else
00809     for (ix = 0; ix < met->nx; ix++)
00810       for (iy = 0; iy < met->ny; iy++)
00811         met->ps[ix][iy] = met->p[0];
00812
00813   /* Create periodic boundary conditions... */
00814   read_met_periodic(met);
00815
00816   /* Close file... */
00817   NC(nc_close(ncid));
00818 }
00819
00820 /*****************************************************************************/
00821
00822 void read_met_extrapolate(
00823   met_t * met) {
00824
00825   int ip, ip0, ix, iy;
00826
00827   /* Loop over columns... */
00828   for (ix = 0; ix < met->nx; ix++)
00829     for (iy = 0; iy < met->ny; iy++) {
00830
00831       /* Find lowest valid data point... */
00832       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00833         if (!gsl_finite(met->t[ix][iy][ip0])
00834             || !gsl_finite(met->u[ix][iy][ip0])
00835             || !gsl_finite(met->v[ix][iy][ip0])
00836             || !gsl_finite(met->w[ix][iy][ip0]))
00837           break;
00838
00839       /* Extrapolate... */
00840       for (ip = ip0; ip >= 0; ip--) {
00841         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00842         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00843         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00844         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00845         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00846         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00847       }
00848     }
00849 }
00850
00851 /*****************************************************************************/
00852
00853 void read_met_help(
00854   int ncid,
00855   char *varname,
00856   char *varname2,
00857   met_t * met,
00858   float dest[EX][EY][EP],
00859   float scl) {
00860
00861   static float help[EX * EY * EP];
00862
00863   int ip, ix, iy, n = 0, varid;
00864
00865   /* Check if variable exists... */
00866   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00867     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00868       return;
00869
00870   /* Read data... */
00871   NC(nc_get_var_float(ncid, varid, help));
00872
00873   /* Copy and check data... */
00874   for (ip = 0; ip < met->np; ip++)
```

```
00875     for (iy = 0; iy < met->ny; iy++)
00876       for (ix = 0; ix < met->nx; ix++) {
00877         dest[ix][iy][ip] = scl * help[n++];
00878         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00879           dest[ix][iy][ip] = GSL_NAN;
00880       }
00881 }
00882
00883 /*****************************************************************************/
00884
00885 void read_met_ml2pl(
00886   ctl_t * ctl,
00887   met_t * met,
00888   float var[EX][EY][EP]) {
00889
00890   double aux[EP], p[EP], pt;
00891
00892   int ip, ip2, ix, iy;
00893
00894   /* Loop over columns... */
00895   for (ix = 0; ix < met->nx; ix++)
00896     for (iy = 0; iy < met->ny; iy++) {
00897
00898       /* Copy pressure profile... */
00899       for (ip = 0; ip < met->np; ip++)
00900         p[ip] = met->pl[ix][iy][ip];
00901
00902       /* Interpolate... */
00903       for (ip = 0; ip < ctl->met_np; ip++) {
00904         pt = ctl->met_p[ip];
00905         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00906           pt = p[0];
00907         else if ((pt > p[met->np - 1] && p[1] > p[0])
00908                  || (pt < p[met->np - 1] && p[1] < p[0]))
00909           pt = p[met->np - 1];
00910         ip2 = locate(p, met->np, pt);
00911         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00912                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00913       }
00914
00915       /* Copy data... */
00916       for (ip = 0; ip < ctl->met_np; ip++)
00917         var[ix][iy][ip] = (float) aux[ip];
00918     }
00919 }
00920
00921 /*****************************************************************************/
00922
00923 void read_met_periodic(
00924   met_t * met) {
00925
00926   int ip, iy;
00927
00928   /* Check longitudes... */
00929   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00930              + met->lon[1] - met->lon[0] - 360) < 0.01))
00931     return;
00932
00933   /* Increase longitude counter... */
00934   if ((++met->nx) > EX)
00935     ERRMSG("Cannot create periodic boundary conditions!");
00936
00937   /* Set longitude... */
00938   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
00939
00940   /* Loop over latitudes and pressure levels... */
00941   for (iy = 0; iy < met->ny; iy++)
00942     for (ip = 0; ip < met->np; ip++) {
00943       met->ps[met->nx - 1][iy] = met->ps[0][iy];
00944       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
00945       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00946       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00947       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00948       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00949       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00950     }
00951 }
00952
00953 /*****************************************************************************/
00954
00955 double scan_ctl(
00956   const char *filename,
00957   int argc,
00958   char *argv[],
00959   const char *varname,
00960   int arridx,
```

```
00961   const char *defvalue,
00962   char *value) {
00963
00964   FILE *in = NULL;
00965
00966   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00967     msg[LEN], rvarname[LEN], rval[LEN];
00968
00969   int contain = 0, i;
00970
00971   /* Open file... */
00972   if (filename[strlen(filename) - 1] != '-')
00973     if (!(in = fopen(filename, "r")))
00974       ERRMSG("Cannot open file!");
00975
00976   /* Set full variable name... */
00977   if (arridx >= 0) {
00978     sprintf(fullname1, "%s[%d]", varname, arridx);
00979     sprintf(fullname2, "%s[*]", varname);
00980   } else {
00981     sprintf(fullname1, "%s", varname);
00982     sprintf(fullname2, "%s", varname);
00983   }
00984
00985   /* Read data... */
00986   if (in != NULL)
00987     while (fgets(line, LEN, in))
00988       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00989         if (strcasecmp(rvarname, fullname1) == 0 ||
00990             strcasecmp(rvarname, fullname2) == 0) {
00991           contain = 1;
00992           break;
00993         }
00994   for (i = 1; i < argc - 1; i++)
00995     if (strcasecmp(argv[i], fullname1) == 0 ||
00996         strcasecmp(argv[i], fullname2) == 0) {
00997       sprintf(rval, "%s", argv[i + 1]);
00998       contain = 1;
00999       break;
01000     }
01001
01002   /* Close file... */
01003   if (in != NULL)
01004     fclose(in);
01005
01006   /* Check for missing variables... */
01007   if (!contain) {
01008     if (strlen(defvalue) > 0)
01009       sprintf(rval, "%s", defvalue);
01010     else {
01011       sprintf(msg, "Missing variable %s!\n", fullname1);
01012       ERRMSG(msg);
01013     }
01014   }
01015
01016   /* Write info... */
01017   printf("%s = %s\n", fullname1, rval);
01018
01019   /* Return values... */
01020   if (value != NULL)
01021     sprintf(value, "%s", rval);
01022   return atof(rval);
01023 }
01024
01025 /*****************************************************************************/
01026
01027 void time2jsec(
01028   int year,
01029   int mon,
01030   int day,
01031   int hour,
01032   int min,
01033   int sec,
01034   double remain,
01035   double *jsec) {
01036
01037   struct tm t0, t1;
01038
01039   t0.tm_year = 100;
01040   t0.tm_mon = 0;
01041   t0.tm_mday = 1;
01042   t0.tm_hour = 0;
01043   t0.tm_min = 0;
01044   t0.tm_sec = 0;
01045
01046   t1.tm_year = year - 1900;
01047   t1.tm_mon = mon - 1;
```

```
01048   t1.tm_mday = day;
01049   t1.tm_hour = hour;
01050   t1.tm_min = min;
01051   t1.tm_sec = sec;
01052
01053   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01054 }
01055
01056 /*****************************************************************************/
01057
01058 void timer(
01059   const char *name,
01060   int id,
01061   int mode) {
01062
01063   static double starttime[NTIMER], runtime[NTIMER];
01064
01065   /* Check id... */
01066   if (id < 0 || id >= NTIMER)
01067     ERRMSG("Too many timers!");
01068
01069   /* Start timer... */
01070   if (mode == 1) {
01071     if (starttime[id] <= 0)
01072       starttime[id] = omp_get_wtime();
01073     else
01074       ERRMSG("Timer already started!");
01075   }
01076
01077   /* Stop timer... */
01078   else if (mode == 2) {
01079     if (starttime[id] > 0) {
01080       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01081       starttime[id] = -1;
01082     } else
01083       ERRMSG("Timer not started!");
01084   }
01085
01086   /* Print timer... */
01087   else if (mode == 3)
01088     printf("%s = %g s\n", name, runtime[id]);
01089 }
01090
01091 /*****************************************************************************/
01092
01093 double tropopause(
01094   double t,
01095   double lat) {
01096
01097   static double doys[12]
01098   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01099
01100   static double lats[73]
01101     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01102     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01103     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01104     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01105     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01106     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01107     75, 77.5, 80, 82.5, 85, 87.5, 90
01108   };
01109
01110   static double tps[12][73]
01111     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01112          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01113          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01114          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01115          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01116          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01117          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01118          275.3, 275.6, 275.4, 274.1, 273.5},
01119   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01120    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01121    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01122    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01123    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01124    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01125    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01126    287.5, 286.2, 285.8},
01127   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01128    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01129    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01130    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01131    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01132    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01133    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01134    304.3, 304.9, 306, 306.6, 306.2, 306},
```

```
01135   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01136    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01137    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01138    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01139    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01140    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01141    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01142    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01143   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01144    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01145    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01146    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01147    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01148    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01149    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01150    325.3, 325.8, 325.8},
01151   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01152    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01153    228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01154    105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01155    106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01156    127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01157    251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01158    308.5, 312.2, 313.1, 313.3},
01159   {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01160    187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01161    235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01162    110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01163    111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01164    117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01165    224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01166    275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01167   {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01168    185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01169    233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01170    110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01171    112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01172    120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01173    230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01174    278.2, 282.6, 287.4, 290.9, 292.5, 293},
01175   {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01176    183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01177    243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01178    114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01179    110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01180    114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01181    203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01182    276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01183   {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01184    215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01185    237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01186    111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01187    106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01188    112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01189    206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01190    279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01191    305.1},
01192   {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01193    253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01194    223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01195    108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01196    102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01197    109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01198    241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01199    286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01200   {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01201    284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01202    175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01203    100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01204    100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01205    186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01206    280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01207    281.7, 281.1, 281.2}
01208   };
01209
01210   double doy, p0, p1, pt;
01211
01212   int imon, ilat;
01213
01214   /* Get day of year... */
01215   doy = fmod(t / 86400., 365.25);
01216   while (doy < 0)
01217     doy += 365.25;
01218
01219   /* Get indices... */
01220   imon = locate(doys, 12, doy);
01221   ilat = locate(lats, 73, lat);
```

```
01222
01223   /* Get tropopause pressure... */
01224   p0 = LIN(lats[ilat], tps[imon][ilat],
01225            lats[ilat + 1], tps[imon][ilat + 1], lat);
01226   p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01227            lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01228   pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01229
01230   /* Return tropopause pressure... */
01231   return pt;
01232 }
01233
01234 /*****************************************************************************/
01235
01236 void write_atm(
01237   const char *filename,
01238   ctl_t * ctl,
01239   atm_t * atm,
01240   double t) {
01241
01242   FILE *in, *out;
01243
01244   char line[LEN];
01245
01246   double r, t0, t1;
01247
01248   int ip, iq, year, mon, day, hour, min, sec;
01249
01250   /* Set time interval for output... */
01251   t0 = t - 0.5 * ctl->dt_mod;
01252   t1 = t + 0.5 * ctl->dt_mod;
01253
01254   /* Check if gnuplot output is requested... */
01255   if (ctl->atm_gpfile[0] != '-') {
01256
01257     /* Write info... */
01258     printf("Plot atmospheric data: %s.png\n", filename);
01259
01260     /* Create gnuplot pipe... */
01261     if (!(out = popen("gnuplot", "w")))
01262       ERRMSG("Cannot create pipe to gnuplot!");
01263
01264     /* Set plot filename... */
01265     fprintf(out, "set out \"%s.png\"\n", filename);
01266
01267     /* Set time string... */
01268     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01269     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01270             year, mon, day, hour, min);
01271
01272     /* Dump gnuplot file to pipe... */
01273     if (!(in = fopen(ctl->atm_gpfile, "r")))
01274       ERRMSG("Cannot open file!");
01275     while (fgets(line, LEN, in))
01276       fprintf(out, "%s", line);
01277     fclose(in);
01278   }
01279
01280   else {
01281
01282     /* Write info... */
01283     printf("Write atmospheric data: %s\n", filename);
01284
01285     /* Create file... */
01286     if (!(out = fopen(filename, "w")))
01287       ERRMSG("Cannot create file!");
01288   }
01289
01290   /* Write header... */
01291   fprintf(out,
01292           "# $1 = time [s]\n"
01293           "# $2 = altitude [km]\n"
01294           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01295   for (iq = 0; iq < ctl->nq; iq++)
01296     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01297             ctl->qnt_unit[iq]);
01298   fprintf(out, "\n");
01299
01300   /* Write data... */
01301   for (ip = 0; ip < atm->np; ip++) {
01302
01303     /* Check time... */
01304     if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01305       continue;
01306
01307     /* Write output... */
01308     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
```

```
01309             atm->lon[ip], atm->lat[ip]);
01310     for (iq = 0; iq < ctl->nq; iq++) {
01311       fprintf(out, " ");
01312       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01313     }
01314     fprintf(out, "\n");
01315   }
01316
01317   /* Close file... */
01318   fclose(out);
01319 }
01320
01321 /*****************************************************************************/
01322
01323 void write_csi(
01324   const char *filename,
01325   ctl_t * ctl,
01326   atm_t * atm,
01327   double t) {
01328
01329   static FILE *in, *out;
01330
01331   static char line[LEN];
01332
01333   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01334     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01335
01336   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01337
01338   /* Init... */
01339   if (!init) {
01340     init = 1;
01341
01342     /* Check quantity index for mass... */
01343     if (ctl->qnt_m < 0)
01344       ERRMSG("Need quantity mass to analyze CSI!");
01345
01346     /* Open observation data file... */
01347     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01348     if (!(in = fopen(ctl->csi_obsfile, "r")))
01349       ERRMSG("Cannot open file!");
01350
01351     /* Create new file... */
01352     printf("Write CSI data: %s\n", filename);
01353     if (!(out = fopen(filename, "w")))
01354       ERRMSG("Cannot create file!");
01355
01356     /* Write header... */
01357     fprintf(out,
01358             "# $1 = time [s]\n"
01359             "# $2 = number of hits (cx)\n"
01360             "# $3 = number of misses (cy)\n"
01361             "# $4 = number of false alarms (cz)\n"
01362             "# $5 = number of observations (cx + cy)\n"
01363             "# $6 = number of forecasts (cx + cz)\n"
01364             "# $7 = bias (forecasts/observations) [%%]\n"
01365             "# $8 = probability of detection (POD) [%%]\n"
01366             "# $9 = false alarm rate (FAR) [%%]\n"
01367             "# $10 = critical success index (CSI) [%%]\n\n");
01368   }
01369
01370   /* Set time interval... */
01371   t0 = t - 0.5 * ctl->dt_mod;
01372   t1 = t + 0.5 * ctl->dt_mod;
01373
01374   /* Initialize grid cells... */
01375   for (ix = 0; ix < ctl->csi_nx; ix++)
01376     for (iy = 0; iy < ctl->csi_ny; iy++)
01377       for (iz = 0; iz < ctl->csi_nz; iz++)
01378         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01379
01380   /* Read data... */
01381   while (fgets(line, LEN, in)) {
01382
01383     /* Read data... */
01384     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01385         5)
01386       continue;
01387
01388     /* Check time... */
01389     if (rt < t0)
01390       continue;
01391     if (rt > t1)
01392       break;
01393
01394     /* Calculate indices... */
01395     ix = (int) ((rlon - ctl->csi_lon0)
```

```
01396                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01397     iy = (int) ((rlat - ctl->csi_lat0)
01398                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01399     iz = (int) ((rz - ctl->csi_z0)
01400                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01401
01402     /* Check indices... */
01403     if (ix < 0 || ix >= ctl->csi_nx ||
01404         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01405       continue;
01406
01407     /* Get mean observation index... */
01408     obsmean[ix][iy][iz] += robs;
01409     obscount[ix][iy][iz]++;
01410   }
01411
01412   /* Analyze model data... */
01413   for (ip = 0; ip < atm->np; ip++) {
01414
01415     /* Check time... */
01416     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01417       continue;
01418
01419     /* Get indices... */
01420     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01421                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01422     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01423                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01424     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01425                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01426
01427     /* Check indices... */
01428     if (ix < 0 || ix >= ctl->csi_nx ||
01429         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01430       continue;
01431
01432     /* Get total mass in grid cell... */
01433     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01434   }
01435
01436   /* Analyze all grid cells... */
01437   for (ix = 0; ix < ctl->csi_nx; ix++)
01438     for (iy = 0; iy < ctl->csi_ny; iy++)
01439       for (iz = 0; iz < ctl->csi_nz; iz++) {
01440
01441         /* Calculate mean observation index... */
01442         if (obscount[ix][iy][iz] > 0)
01443           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01444
01445         /* Calculate column density... */
01446         if (modmean[ix][iy][iz] > 0) {
01447           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01448           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01449           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01450           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01451             * cos(lat * M_PI / 180.);
01452           modmean[ix][iy][iz] /= (1e6 * area);
01453         }
01454
01455         /* Calculate CSI... */
01456         if (obscount[ix][iy][iz] > 0) {
01457           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01458               modmean[ix][iy][iz] >= ctl->csi_modmin)
01459             cx++;
01460           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01461                    modmean[ix][iy][iz] < ctl->csi_modmin)
01462             cy++;
01463           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01464                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01465             cz++;
01466         }
01467       }
01468
01469   /* Write output... */
01470   if (fmod(t, ctl->csi_dt_out) == 0) {
01471
01472     /* Write... */
01473     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01474             t, cx, cy, cz, cx + cy, cx + cz,
01475             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01476             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01477             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01478             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01479
01480     /* Set counters to zero... */
01481     cx = cy = cz = 0;
01482   }
```

```
01483
01484   /* Close file... */
01485   if (t == ctl->t_stop)
01486     fclose(out);
01487 }
01488
01489 /*****************************************************************************/
01490
01491 void write_ens(
01492   const char *filename,
01493   ctl_t * ctl,
01494   atm_t * atm,
01495   double t) {
01496
01497   static FILE *out;
01498
01499   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01500     t0, t1, x[NENS][3], xm[3];
01501
01502   static int init, ip, iq;
01503
01504   static size_t i, n;
01505
01506   /* Init... */
01507   if (!init) {
01508     init = 1;
01509
01510     /* Check quantities... */
01511     if (ctl->qnt_ens < 0)
01512       ERRMSG("Missing ensemble IDs!");
01513
01514     /* Create new file... */
01515     printf("Write ensemble data: %s\n", filename);
01516     if (!(out = fopen(filename, "w")))
01517       ERRMSG("Cannot create file!");
01518
01519     /* Write header... */
01520     fprintf(out,
01521             "# $1 = time [s]\n"
01522             "# $2 = altitude [km]\n"
01523             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01524     for (iq = 0; iq < ctl->nq; iq++)
01525       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01526               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01527     for (iq = 0; iq < ctl->nq; iq++)
01528       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01529               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01530     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01531   }
01532
01533   /* Set time interval... */
01534   t0 = t - 0.5 * ctl->dt_mod;
01535   t1 = t + 0.5 * ctl->dt_mod;
01536
01537   /* Init... */
01538   ens = GSL_NAN;
01539   n = 0;
01540
01541   /* Loop over air parcels... */
01542   for (ip = 0; ip < atm->np; ip++) {
01543
01544     /* Check time... */
01545     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01546       continue;
01547
01548     /* Check ensemble id... */
01549     if (atm->q[ctl->qnt_ens][ip] != ens) {
01550
01551       /* Write results... */
01552       if (n > 0) {
01553
01554         /* Get mean position... */
01555         xm[0] = xm[1] = xm[2] = 0;
01556         for (i = 0; i < n; i++) {
01557           xm[0] += x[i][0] / (double) n;
01558           xm[1] += x[i][1] / (double) n;
01559           xm[2] += x[i][2] / (double) n;
01560         }
01561         cart2geo(xm, &dummy, &lon, &lat);
01562         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01563                 lat);
01564
01565         /* Get quantity statistics... */
01566         for (iq = 0; iq < ctl->nq; iq++) {
01567           fprintf(out, " ");
01568           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01569         }
```

```
01570         for (iq = 0; iq < ctl->nq; iq++) {
01571           fprintf(out, " ");
01572           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01573         }
01574         fprintf(out, " %lu\n", n);
01575       }
01576
01577       /* Init new ensemble... */
01578       ens = atm->q[ctl->qnt_ens][ip];
01579       n = 0;
01580     }
01581
01582     /* Save data... */
01583     p[n] = atm->p[ip];
01584     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01585     for (iq = 0; iq < ctl->nq; iq++)
01586       q[iq][n] = atm->q[iq][ip];
01587     if ((++n) >= NENS)
01588       ERRMSG("Too many data points!");
01589   }
01590
01591   /* Write results... */
01592   if (n > 0) {
01593
01594     /* Get mean position... */
01595     xm[0] = xm[1] = xm[2] = 0;
01596     for (i = 0; i < n; i++) {
01597       xm[0] += x[i][0] / (double) n;
01598       xm[1] += x[i][1] / (double) n;
01599       xm[2] += x[i][2] / (double) n;
01600     }
01601     cart2geo(xm, &dummy, &lon, &lat);
01602     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01603
01604     /* Get quantity statistics... */
01605     for (iq = 0; iq < ctl->nq; iq++) {
01606       fprintf(out, " ");
01607       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01608     }
01609     for (iq = 0; iq < ctl->nq; iq++) {
01610       fprintf(out, " ");
01611       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01612     }
01613     fprintf(out, " %lu\n", n);
01614   }
01615
01616   /* Close file... */
01617   if (t == ctl->t_stop)
01618     fclose(out);
01619 }
01620
01621 /*****************************************************************************/
01622
01623 void write_grid(
01624   const char *filename,
01625   ctl_t * ctl,
01626   met_t * met0,
01627   met_t * met1,
01628   atm_t * atm,
01629   double t) {
01630
01631   FILE *in, *out;
01632
01633   char line[LEN];
01634
01635   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01636     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01637
01638   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01639
01640   /* Check dimensions... */
01641   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01642     ERRMSG("Grid dimensions too large!");
01643
01644   /* Check quantity index for mass... */
01645   if (ctl->qnt_m < 0)
01646     ERRMSG("Need quantity mass to write grid data!");
01647
01648   /* Set time interval for output... */
01649   t0 = t - 0.5 * ctl->dt_mod;
01650   t1 = t + 0.5 * ctl->dt_mod;
01651
01652   /* Set grid box size... */
01653   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01654   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01655   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01656
```

```
01657   /* Initialize grid... */
01658   for (ix = 0; ix < ctl->grid_nx; ix++)
01659     for (iy = 0; iy < ctl->grid_ny; iy++)
01660       for (iz = 0; iz < ctl->grid_nz; iz++)
01661         grid_m[ix][iy][iz] = 0;
01662
01663   /* Average data... */
01664   for (ip = 0; ip < atm->np; ip++)
01665     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01666
01667       /* Get index... */
01668       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01669       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01670       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01671
01672       /* Check indices... */
01673       if (ix < 0 || ix >= ctl->grid_nx ||
01674           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01675         continue;
01676
01677       /* Add mass... */
01678       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01679     }
01680
01681   /* Check if gnuplot output is requested... */
01682   if (ctl->grid_gpfile[0] != '-') {
01683
01684     /* Write info... */
01685     printf("Plot grid data: %s.png\n", filename);
01686
01687     /* Create gnuplot pipe... */
01688     if (!(out = popen("gnuplot", "w")))
01689       ERRMSG("Cannot create pipe to gnuplot!");
01690
01691     /* Set plot filename... */
01692     fprintf(out, "set out \"%s.png\"\n", filename);
01693
01694     /* Set time string... */
01695     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01696     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01697             year, mon, day, hour, min);
01698
01699     /* Dump gnuplot file to pipe... */
01700     if (!(in = fopen(ctl->grid_gpfile, "r")))
01701       ERRMSG("Cannot open file!");
01702     while (fgets(line, LEN, in))
01703       fprintf(out, "%s", line);
01704     fclose(in);
01705   }
01706
01707   else {
01708
01709     /* Write info... */
01710     printf("Write grid data: %s\n", filename);
01711
01712     /* Create file... */
01713     if (!(out = fopen(filename, "w")))
01714       ERRMSG("Cannot create file!");
01715   }
01716
01717   /* Write header... */
01718   fprintf(out,
01719           "# $1 = time [s]\n"
01720           "# $2 = altitude [km]\n"
01721           "# $3 = longitude [deg]\n"
01722           "# $4 = latitude [deg]\n"
01723           "# $5 = surface area [km^2]\n"
01724           "# $6 = layer width [km]\n"
01725           "# $7 = temperature [K]\n"
01726           "# $8 = column density [kg/m^2]\n"
01727           "# $9 = mass mixing ratio [1]\n\n");
01728
01729   /* Write data... */
01730   for (ix = 0; ix < ctl->grid_nx; ix++) {
01731     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01732       fprintf(out, "\n");
01733     for (iy = 0; iy < ctl->grid_ny; iy++) {
01734       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01735         fprintf(out, "\n");
01736       for (iz = 0; iz < ctl->grid_nz; iz++)
01737         if (!ctl->grid_sparse
01738             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01739
01740           /* Set coordinates... */
01741           z = ctl->grid_z0 + dz * (iz + 0.5);
01742           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01743           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
```

```
01744
01745           /* Get pressure and temperature... */
01746           press = P(z);
01747           intpol_met_time(met0, met1, t, press, lon, lat,
01748                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01749
01750           /* Calculate surface area... */
01751           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01752             * cos(lat * M_PI / 180.);
01753
01754           /* Calculate column density... */
01755           cd = grid_m[ix][iy][iz] / (1e6 * area);
01756
01757           /* Calculate mass mixing ratio... */
01758           rho_air = 100. * press / (287.058 * temp);
01759           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01760
01761           /* Write output... */
01762           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01763                   t, z, lon, lat, area, dz, temp, cd, mmr);
01764         }
01765     }
01766   }
01767
01768   /* Close file... */
01769   fclose(out);
01770 }
01771
01772 /*****************************************************************************/
01773
01774 void write_prof(
01775   const char *filename,
01776   ctl_t * ctl,
01777   met_t * met0,
01778   met_t * met1,
01779   atm_t * atm,
01780   double t) {
01781
01782   static FILE *in, *out;
01783
01784   static char line[LEN];
01785
01786   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01787     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01788     press, temp, rho_air, mmr, h2o, o3;
01789
01790   static int init, obscount[GX][GY], ip, ix, iy, iz;
01791
01792   /* Init... */
01793   if (!init) {
01794     init = 1;
01795
01796     /* Check quantity index for mass... */
01797     if (ctl->qnt_m < 0)
01798       ERRMSG("Need quantity mass!");
01799
01800     /* Check dimensions... */
01801     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01802       ERRMSG("Grid dimensions too large!");
01803
01804     /* Open observation data file... */
01805     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01806     if (!(in = fopen(ctl->prof_obsfile, "r")))
01807       ERRMSG("Cannot open file!");
01808
01809     /* Create new file... */
01810     printf("Write profile data: %s\n", filename);
01811     if (!(out = fopen(filename, "w")))
01812       ERRMSG("Cannot create file!");
01813
01814     /* Write header... */
01815     fprintf(out,
01816             "# $1  = time [s]\n"
01817             "# $2  = altitude [km]\n"
01818             "# $3  = longitude [deg]\n"
01819             "# $4  = latitude [deg]\n"
01820             "# $5  = pressure [hPa]\n"
01821             "# $6  = temperature [K]\n"
01822             "# $7  = mass mixing ratio [1]\n"
01823             "# $8  = H2O volume mixing ratio [1]\n"
01824             "# $9  = O3 volume mixing ratio [1]\n"
01825             "# $10 = mean BT index [K]\n");
01826
01827     /* Set grid box size... */
01828     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01829     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01830     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
```

```
01831   }
01832
01833   /* Set time interval... */
01834   t0 = t - 0.5 * ctl->dt_mod;
01835   t1 = t + 0.5 * ctl->dt_mod;
01836
01837   /* Initialize... */
01838   for (ix = 0; ix < ctl->prof_nx; ix++)
01839     for (iy = 0; iy < ctl->prof_ny; iy++) {
01840       obsmean[ix][iy] = 0;
01841       obscount[ix][iy] = 0;
01842       tmean[ix][iy] = 0;
01843       for (iz = 0; iz < ctl->prof_nz; iz++)
01844         mass[ix][iy][iz] = 0;
01845     }
01846
01847   /* Read data... */
01848   while (fgets(line, LEN, in)) {
01849
01850     /* Read data... */
01851     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01852       continue;
01853
01854     /* Check time... */
01855     if (rt < t0)
01856       continue;
01857     if (rt > t1)
01858       break;
01859
01860     /* Calculate indices... */
01861     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01862     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01863
01864     /* Check indices... */
01865     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01866       continue;
01867
01868     /* Get mean observation index... */
01869     obsmean[ix][iy] += robs;
01870     tmean[ix][iy] += rt;
01871     obscount[ix][iy]++;
01872   }
01873
01874   /* Analyze model data... */
01875   for (ip = 0; ip < atm->np; ip++) {
01876
01877     /* Check time... */
01878     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01879       continue;
01880
01881     /* Get indices... */
01882     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01883     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01884     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01885
01886     /* Check indices... */
01887     if (ix < 0 || ix >= ctl->prof_nx ||
01888         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01889       continue;
01890
01891     /* Get total mass in grid cell... */
01892     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01893   }
01894
01895   /* Extract profiles... */
01896   for (ix = 0; ix < ctl->prof_nx; ix++)
01897     for (iy = 0; iy < ctl->prof_ny; iy++)
01898       if (obscount[ix][iy] > 0) {
01899
01900         /* Write output... */
01901         fprintf(out, "\n");
01902
01903         /* Loop over altitudes... */
01904         for (iz = 0; iz < ctl->prof_nz; iz++) {
01905
01906           /* Set coordinates... */
01907           z = ctl->prof_z0 + dz * (iz + 0.5);
01908           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01909           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01910
01911           /* Get meteorological data... */
01912           press = P(z);
01913           intpol_met_time(met0, met1, t, press, lon, lat,
01914                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01915
01916           /* Calculate mass mixing ratio... */
01917           rho_air = 100. * press / (287.058 * temp);
```

```
01918           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
01919             * cos(lat * M_PI / 180.);
01920           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01921
01922           /* Write output... */
01923           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
01924                   tmean[ix][iy] / obscount[ix][iy],
01925                   z, lon, lat, press, temp, mmr, h2o, o3,
01926                   obsmean[ix][iy] / obscount[ix][iy]);
01927         }
01928       }
01929
01930   /* Close file... */
01931   if (t == ctl->t_stop)
01932     fclose(out);
01933 }
01934
01935 /*****************************************************************************/
01936
01937 void write_station(
01938   const char *filename,
01939   ctl_t * ctl,
01940   atm_t * atm,
01941   double t) {
01942
01943   static FILE *out;
01944
01945   static double rmax2, t0, t1, x0[3], x1[3];
01946
01947   static int init, ip, iq;
01948
01949   /* Init... */
01950   if (!init) {
01951     init = 1;
01952
01953     /* Write info... */
01954     printf("Write station data: %s\n", filename);
01955
01956     /* Create new file... */
01957     if (!(out = fopen(filename, "w")))
01958       ERRMSG("Cannot create file!");
01959
01960     /* Write header... */
01961     fprintf(out,
01962             "# $1 = time [s]\n"
01963             "# $2 = altitude [km]\n"
01964             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01965     for (iq = 0; iq < ctl->nq; iq++)
01966       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01967               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01968     fprintf(out, "\n");
01969
01970     /* Set geolocation and search radius... */
01971     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01972     rmax2 = gsl_pow_2(ctl->stat_r);
01973   }
01974
01975   /* Set time interval for output... */
01976   t0 = t - 0.5 * ctl->dt_mod;
01977   t1 = t + 0.5 * ctl->dt_mod;
01978
01979   /* Loop over air parcels... */
01980   for (ip = 0; ip < atm->np; ip++) {
01981
01982     /* Check time... */
01983     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01984       continue;
01985
01986     /* Check station flag... */
01987     if (ctl->qnt_stat >= 0)
01988       if (atm->q[ctl->qnt_stat][ip])
01989         continue;
01990
01991     /* Get Cartesian coordinates... */
01992     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01993
01994     /* Check horizontal distance... */
01995     if (DIST2(x0, x1) > rmax2)
01996       continue;
01997
01998     /* Set station flag... */
01999     if (ctl->qnt_stat >= 0)
02000       atm->q[ctl->qnt_stat][ip] = 1;
02001
02002     /* Write data... */
02003     fprintf(out, "%.2f %g %g %g",
02004             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
```

```
02005     for (iq = 0; iq < ctl->nq; iq++) {
02006       fprintf(out, " ");
02007       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02008     }
02009     fprintf(out, "\n");
02010   }
02011
02012   /* Close file... */
02013   if (t == ctl->t_stop)
02014     fclose(out);
02015 }
```

5.13 libtrac.h File Reference

MPTRAC library declarations.

Data Structures

- struct ctl_t

  Control parameters.
- struct atm_t

  Atmospheric data.
- struct met_t

  Meteorological data.

Functions

- void cart2geo (double *x, double *z, double *lon, double *lat)

  Convert Cartesian coordinates to geolocation.
- double deg2dx (double dlon, double lat)

  Convert degrees to horizontal distance.
- double deg2dy (double dlat)

  Convert degrees to horizontal distance.
- double dp2dz (double dp, double p)

  Convert pressure to vertical distance.
- double dx2deg (double dx, double lat)

  Convert horizontal distance to degrees.
- double dy2deg (double dy)

  Convert horizontal distance to degrees.
- double dz2dp (double dz, double p)

  Convert vertical distance to pressure.
- void geo2cart (double z, double lon, double lat, double *x)

  Convert geolocation to Cartesian coordinates.
- void get_met (ctl_t *ctl, char *metbase, double t, met_t *met0, met_t *met1)

  Get meteorological data for given timestep.
- void get_met_help (double t, int direct, char *metbase, double dt_met, char *filename)

  Get meteorological data for timestep.
- void intpol_met_2d (double array[EX][EY], int ix, int iy, double wx, double wy, double *var)

  Linear interpolation of 2-D meteorological data.
- void intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double *var)

  Linear interpolation of 3-D meteorological data.
- void intpol_met_space (met_t *met, double p, double lon, double lat, double *ps, double *t, double *u, double *v, double *w, double *h2o, double *o3)

Spatial interpolation of meteorological data.

- void intpol_met_time (met_t *met0, met_t *met1, double ts, double p, double lon, double lat, double *ps, double *t, double *u, double *v, double *w, double *h2o, double *o3)

  Temporal interpolation of meteorological data.
- void jsec2time (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)

  Convert seconds to date.
- int locate (double *xx, int n, double x)

  Find array index.
- void read_atm (const char *filename, ctl_t *ctl, atm_t *atm)

  Read atmospheric data.
- void read_ctl (const char *filename, int argc, char *argv[ ], ctl_t *ctl)

  Read control parameters.
- void read_met (ctl_t *ctl, char *filename, met_t *met)

  Read meteorological data file.
- void read_met_extrapolate (met_t *met)

  Extrapolate meteorological data at lower boundary.
- void read_met_help (int ncid, char *varname, char *varname2, met_t *met, float dest[EX][EY][EP], float scl)

  Read and convert variable from meteorological data file.
- void read_met_ml2pl (ctl_t *ctl, met_t *met, float var[EX][EY][EP])

  Convert meteorological data from model levels to pressure levels.
- void read_met_periodic (met_t *met)

  Create meteorological data with periodic boundary conditions.
- double scan_ctl (const char *filename, int argc, char *argv[ ], const char *varname, int arridx, const char *defvalue, char *value)

  Read a control parameter from file or command line.
- void time2jsec (int year, int mon, int day, int hour, int min, int sec, double remain, double *jsec)

  Convert date to seconds.
- void timer (const char *name, int id, int mode)

  Measure wall-clock time.
- double tropopause (double t, double lat)
- void write_atm (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write atmospheric data.
- void write_csi (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write CSI data.
- void write_ens (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write ensemble data.
- void write_grid (const char *filename, ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double t)

  Write gridded data.
- void write_prof (const char *filename, ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double t)

  Write profile data.
- void write_station (const char *filename, ctl_t *ctl, atm_t *atm, double t)

  Write station data.

5.13.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

5.13.2 Function Documentation

5.13.2.1 void cart2geo (double * *x*, double * *z*, double * *lon*, double * *lat*)

Convert Cartesian coordinates to geolocation.

Definition at line 29 of file libtrac.c.

```
00033                {
00034
00035   double radius;
00036
00037   radius = NORM(x);
00038   *lat = asin(x[2] / radius) * 180 / M_PI;
00039   *lon = atan2(x[1], x[0]) * 180 / M_PI;
00040   *z = radius - RE;
00041 }
```

5.13.2.2 double deg2dx (double *dlon*, double *lat*)

Convert degrees to horizontal distance.

Definition at line 45 of file libtrac.c.

```
00047               {
00048
00049   return dlon * M_PI * RE / 180. * cos(lat / 180. * M_PI);
00050 }
```

5.13.2.3 double deg2dy (double *dlat*)

Convert degrees to horizontal distance.

Definition at line 54 of file libtrac.c.

```
00055                {
00056
00057   return dlat * M_PI * RE / 180.;
00058 }
```

5.13.2.4 double dp2dz (double *dp*, double *p*)

Convert pressure to vertical distance.

Definition at line 62 of file libtrac.c.

```
00064             {
00065
00066   return -dp * H0 / p;
00067 }
```

5.13.2.5 double dx2deg (double *dx*, double *lat*)

Convert horizontal distance to degrees.

Definition at line 71 of file libtrac.c.

```
00073               {
00074
00075   /* Avoid singularity at poles... */
00076   if (lat < -89.999 || lat > 89.999)
00077     return 0;
00078   else
00079     return dx * 180. / (M_PI * RE * cos(lat / 180. * M_PI));
00080 }
```

5.13.2.6 double dy2deg (double *dy*)

Convert horizontal distance to degrees.

Definition at line 84 of file libtrac.c.

```
00085               {
00086
00087   return dy * 180. / (M_PI * RE);
00088 }
```

5.13.2.7 double dz2dp (double *dz*, double *p*)

Convert vertical distance to pressure.

Definition at line 92 of file libtrac.c.

```
00094             {
00095
00096   return -dz * p / H0;
00097 }
```

5.13.2.8 void geo2cart (double *z*, double *lon*, double *lat*, double * *x*)

Convert geolocation to Cartesian coordinates.

Definition at line 101 of file libtrac.c.

```
00105             {
00106
00107   double radius;
00108
00109   radius = z + RE;
00110   x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
00111   x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
00112   x[2] = radius * sin(lat / 180 * M_PI);
00113 }
```

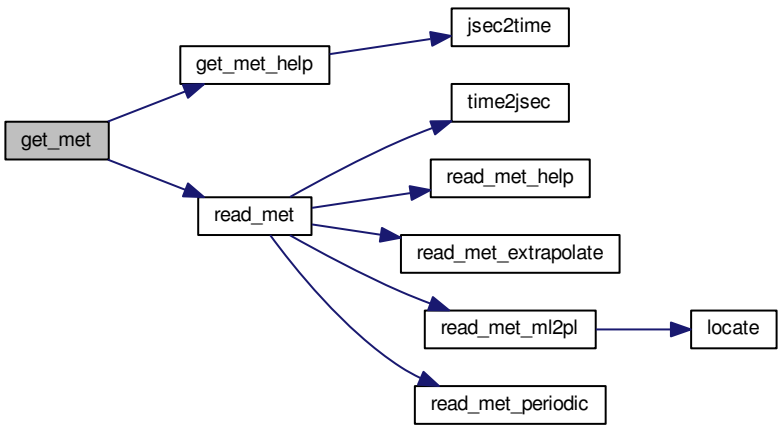
5.13.2.9 void get_met (ctl_t * *ctl,* char * *metbase,* double *t,* met_t * *met0,* met_t * *met1*)

Get meteorological data for given timestep.

Definition at line 117 of file libtrac.c.

```
00122                {
00123
00124   char filename[LEN];
00125
00126   static int init;
00127
00128   /* Init... */
00129   if (!init) {
00130     init = 1;
00131
00132     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00133     read_met(ctl, filename, met0);
00134
00135     get_met_help(t + 1.0 * ctl->direction, 1, metbase, ctl->
      dt_met, filename);
00136     read_met(ctl, filename, met1);
00137   }
00138
00139   /* Read new data for forward trajectories... */
00140   if (t > met1->time && ctl->direction == 1) {
00141     memcpy(met0, met1, sizeof(met_t));
00142     get_met_help(t, 1, metbase, ctl->dt_met, filename);
00143     read_met(ctl, filename, met1);
00144   }
00145
00146   /* Read new data for backward trajectories... */
00147   if (t < met0->time && ctl->direction == -1) {
00148     memcpy(met1, met0, sizeof(met_t));
00149     get_met_help(t, -1, metbase, ctl->dt_met, filename);
00150     read_met(ctl, filename, met0);
00151   }
00152 }
```

Here is the call graph for this function:



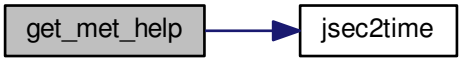
5.13.2.10 void get_met_help (double *t,* int *direct,* char * *metbase,* double *dt_met,* char * *filename*)

Get meteorological data for timestep.

Definition at line 156 of file libtrac.c.

```
00161                   {
00162
00163   double t6, r;
00164
00165   int year, mon, day, hour, min, sec;
00166
00167   /* Round time to fixed intervals... */
00168   if (direct == -1)
00169     t6 = floor(t / dt_met) * dt_met;
00170   else
00171     t6 = ceil(t / dt_met) * dt_met;
00172
00173   /* Decode time... */
00174   jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);
00175
00176   /* Set filename... */
00177   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
00178 }
```

Here is the call graph for this function:



5.13.2.11 void intpol_met_2d (double *array[EX][EY]*, int *ix*, int *iy*, double *wx*, double *wy*, double * *var*)

Linear interpolation of 2-D meteorological data.

Definition at line 182 of file libtrac.c.

```
00188               {
00189
00190   double aux00, aux01, aux10, aux11;
00191
00192   /* Set variables... */
00193   aux00 = array[ix][iy];
00194   aux01 = array[ix][iy + 1];
00195   aux10 = array[ix + 1][iy];
00196   aux11 = array[ix + 1][iy + 1];
00197
00198   /* Interpolate horizontally... */
00199   aux00 = wy * (aux00 - aux01) + aux01;
00200   aux11 = wy * (aux10 - aux11) + aux11;
00201   *var = wx * (aux00 - aux11) + aux11;
00202 }
```

5.13.2.12 void intpol_met_3d (float *array[EX][EY][EP]*, int *ip*, int *ix*, int *iy*, double *wp*, double *wx*, double *wy*, double * *var*)

Linear interpolation of 3-D meteorological data.

Definition at line 206 of file libtrac.c.

```
00214               {
00215
00216   double aux00, aux01, aux10, aux11;
00217
00218   /* Interpolate vertically... */
00219   aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
00220     + array[ix][iy][ip + 1];
00221   aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
00222     + array[ix][iy + 1][ip + 1];
00223   aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
00224     + array[ix + 1][iy][ip + 1];
00225   aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
00226     + array[ix + 1][iy + 1][ip + 1];
00227
00228   /* Interpolate horizontally... */
00229   aux00 = wy * (aux00 - aux01) + aux01;
00230   aux11 = wy * (aux10 - aux11) + aux11;
00231   *var = wx * (aux00 - aux11) + aux11;
00232 }
```

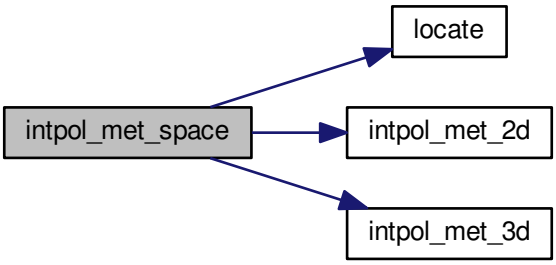
5.13.2.13 void intpol_met_space (met_t * *met*, double *p*, double *lon*, double *lat*, double * *ps*, double * *t*, double * *u*, double * *v*, double * *w*, double * *h2o*, double * *o3*)

Spatial interpolation of meteorological data.

Definition at line 236 of file libtrac.c.

```
00247               {
00248
00249   double wp, wx, wy;
00250
00251   int ip, ix, iy;
00252
00253   /* Check longitude... */
00254   if (met->lon[met->nx - 1] > 180 && lon < 0)
00255     lon += 360;
00256
00257   /* Get indices... */
00258   ip = locate(met->p, met->np, p);
00259   ix = locate(met->lon, met->nx, lon);
00260   iy = locate(met->lat, met->ny, lat);
00261
00262   /* Get weights... */
00263   wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
00264   wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
00265   wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);
00266
00267   /* Interpolate... */
00268   if (ps != NULL)
00269     intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
00270   if (t != NULL)
00271     intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
00272   if (u != NULL)
00273     intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
00274   if (v != NULL)
00275     intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
00276   if (w != NULL)
00277     intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
00278   if (h2o != NULL)
00279     intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
00280   if (o3 != NULL)
00281     intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
00282 }
```

Here is the call graph for this function:



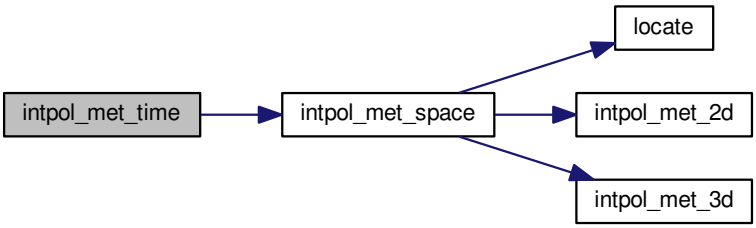
5.13.2.14 void intpol_met_time (met_t * *met0*, met_t * *met1*, double *ts*, double *p*, double *lon*, double *lat*, double * *ps*, double * *t*, double * *u*, double * *v*, double * *w*, double * *h2o*, double * *o3*)

Temporal interpolation of meteorological data.

Definition at line 286 of file libtrac.c.

```
00299              {
00300
00301   double h2o0, h2o1, o30, o31, ps0, ps1, t0, t1, u0, u1, v0, v1, w0, w1, wt;
00302
00303   /* Spatial interpolation... */
00304   intpol_met_space(met0, p, lon, lat,
00305                    ps == NULL ? NULL : &ps0,
00306                    t == NULL ? NULL : &t0,
00307                    u == NULL ? NULL : &u0,
00308                    v == NULL ? NULL : &v0,
00309                    w == NULL ? NULL : &w0,
00310                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
00311   intpol_met_space(met1, p, lon, lat,
00312                    ps == NULL ? NULL : &ps1,
00313                    t == NULL ? NULL : &t1,
00314                    u == NULL ? NULL : &u1,
00315                    v == NULL ? NULL : &v1,
00316                    w == NULL ? NULL : &w1,
00317                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);
00318
00319   /* Get weighting factor... */
00320   wt = (met1->time - ts) / (met1->time - met0->time);
00321
00322   /* Interpolate... */
00323   if (ps != NULL)
00324     *ps = wt * (ps0 - ps1) + ps1;
00325   if (t != NULL)
00326     *t = wt * (t0 - t1) + t1;
00327   if (u != NULL)
00328     *u = wt * (u0 - u1) + u1;
00329   if (v != NULL)
00330     *v = wt * (v0 - v1) + v1;
00331   if (w != NULL)
00332     *w = wt * (w0 - w1) + w1;
00333   if (h2o != NULL)
00334     *h2o = wt * (h2o0 - h2o1) + h2o1;
00335   if (o3 != NULL)
00336     *o3 = wt * (o30 - o31) + o31;
00337 }
```

Here is the call graph for this function:



5.13.2.15 void jsec2time (double *jsec*, int * *year*, int * *mon*, int * *day*, int * *hour*, int * *min*, int * *sec*, double * *remain*)

Convert seconds to date.

Definition at line 341 of file libtrac.c.

```
00349                   {
00350
00351   struct tm t0, *t1;
00352
00353   time_t jsec0;
00354
00355   t0.tm_year = 100;
00356   t0.tm_mon = 0;
00357   t0.tm_mday = 1;
00358   t0.tm_hour = 0;
00359   t0.tm_min = 0;
00360   t0.tm_sec = 0;
00361
00362   jsec0 = (time_t) jsec + timegm(&t0);
00363   t1 = gmtime(&jsec0);
00364
00365   *year = t1->tm_year + 1900;
00366   *mon = t1->tm_mon + 1;
00367   *day = t1->tm_mday;
00368   *hour = t1->tm_hour;
00369   *min = t1->tm_min;
00370   *sec = t1->tm_sec;
00371   *remain = jsec - floor(jsec);
00372 }
```

5.13.2.16 int locate (double * *xx*, int *n*, double *x*)

Find array index.

Definition at line 376 of file libtrac.c.

```
00379             {
00380
00381   int i, ilo, ihi;
00382
00383   ilo = 0;
00384   ihi = n - 1;
00385   i = (ihi + ilo) >> 1;
00386
00387   if (xx[i] < xx[i + 1])
00388     while (ihi > ilo + 1) {
00389       i = (ihi + ilo) >> 1;
00390       if (xx[i] > x)
```

```
00391         ihi = i;
00392       else
00393         ilo = i;
00394     } else
00395       while (ihi > ilo + 1) {
00396         i = (ihi + ilo) >> 1;
00397         if (xx[i] <= x)
00398           ihi = i;
00399         else
00400           ilo = i;
00401       }
00402
00403     return ilo;
00404 }
```

5.13.2.17 void read_atm (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*)

Read atmospheric data.

Definition at line 408 of file libtrac.c.

```
00411                 {
00412
00413   FILE *in;
00414
00415   char line[LEN], *tok;
00416
00417   int iq;
00418
00419   /* Init... */
00420   atm->np = 0;
00421
00422   /* Write info... */
00423   printf("Read atmospheric data: %s\n", filename);
00424
00425   /* Open file... */
00426   if (!(in = fopen(filename, "r")))
00427     ERRMSG("Cannot open file!");
00428
00429   /* Read line... */
00430   while (fgets(line, LEN, in)) {
00431
00432     /* Read data... */
00433     TOK(line, tok, "%lg", atm->time[atm->np]);
00434     TOK(NULL, tok, "%lg", atm->p[atm->np]);
00435     TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00436     TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00437     for (iq = 0; iq < ctl->nq; iq++)
00438       TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00439
00440     /* Convert altitude to pressure... */
00441     atm->p[atm->np] = P(atm->p[atm->np]);
00442
00443     /* Increment data point counter... */
00444     if ((++atm->np) > NP)
00445       ERRMSG("Too many data points!");
00446   }
00447
00448   /* Close file... */
00449   fclose(in);
00450
00451   /* Check number of points... */
00452   if (atm->np < 1)
00453     ERRMSG("Can not read any data!");
00454 }
```

5.13.2.18 void read_ctl (const char * *filename*, int *argc*, char * *argv[]*, ctl_t * *ctl*)

Read control parameters.

Definition at line 458 of file libtrac.c.

```
00462               {
00463
00464   int ip, iq;
00465
00466   /* Write info... */
00467   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00468          "(executable: %s | compiled: %s, %s)\n\n",
00469          argv[0], __DATE__, __TIME__);
00470
00471   /* Initialize quantity indices... */
00472   ctl->qnt_ens = -1;
00473   ctl->qnt_m = -1;
00474   ctl->qnt_r = -1;
00475   ctl->qnt_rho = -1;
00476   ctl->qnt_ps = -1;
00477   ctl->qnt_p = -1;
00478   ctl->qnt_t = -1;
00479   ctl->qnt_u = -1;
00480   ctl->qnt_v = -1;
00481   ctl->qnt_w = -1;
00482   ctl->qnt_h2o = -1;
00483   ctl->qnt_o3 = -1;
00484   ctl->qnt_theta = -1;
00485   ctl->qnt_pv = -1;
00486   ctl->qnt_tice = -1;
00487   ctl->qnt_tsts = -1;
00488   ctl->qnt_tnat = -1;
00489   ctl->qnt_stat = -1;
00490
00491   /* Read quantities... */
00492   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00493   for (iq = 0; iq < ctl->nq; iq++) {
00494
00495     /* Read quantity name and format... */
00496     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
00497     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
00498              ctl->qnt_format[iq]);
00499
00500     /* Try to identify quantity... */
00501     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00502       ctl->qnt_ens = iq;
00503       sprintf(ctl->qnt_unit[iq], "-");
00504     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00505       ctl->qnt_m = iq;
00506       sprintf(ctl->qnt_unit[iq], "kg");
00507     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
00508       ctl->qnt_r = iq;
00509       sprintf(ctl->qnt_unit[iq], "m");
00510     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00511       ctl->qnt_rho = iq;
00512       sprintf(ctl->qnt_unit[iq], "kg/m^3");
00513     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00514       ctl->qnt_ps = iq;
00515       sprintf(ctl->qnt_unit[iq], "hPa");
00516     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00517       ctl->qnt_p = iq;
00518       sprintf(ctl->qnt_unit[iq], "hPa");
00519     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00520       ctl->qnt_t = iq;
00521       sprintf(ctl->qnt_unit[iq], "K");
00522     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00523       ctl->qnt_u = iq;
00524       sprintf(ctl->qnt_unit[iq], "m/s");
00525     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00526       ctl->qnt_v = iq;
00527       sprintf(ctl->qnt_unit[iq], "m/s");
00528     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00529       ctl->qnt_w = iq;
00530       sprintf(ctl->qnt_unit[iq], "hPa/s");
00531     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00532       ctl->qnt_h2o = iq;
00533       sprintf(ctl->qnt_unit[iq], "1");
00534     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00535       ctl->qnt_o3 = iq;
00536       sprintf(ctl->qnt_unit[iq], "1");
00537     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00538       ctl->qnt_theta = iq;
00539       sprintf(ctl->qnt_unit[iq], "K");
00540     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00541       ctl->qnt_pv = iq;
00542       sprintf(ctl->qnt_unit[iq], "PVU");
00543     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00544       ctl->qnt_tice = iq;
00545       sprintf(ctl->qnt_unit[iq], "K");
00546     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00547       ctl->qnt_tsts = iq;
00548       sprintf(ctl->qnt_unit[iq], "K");
```

```
00549     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00550       ctl->qnt_tnat = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00553       ctl->qnt_stat = iq;
00554       sprintf(ctl->qnt_unit[iq], "-");
00555     } else
00556       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00557   }
00558
00559   /* Time steps of simulation... */
00560   ctl->direction =
00561     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00562   if (ctl->direction != -1 && ctl->direction != 1)
00563     ERRMSG("Set DIRECTION to -1 or 1!");
00564   ctl->t_start =
00565     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00566   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00567   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00568
00569   /* Meteorological data... */
00570   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00571   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00572   if (ctl->met_np > EP)
00573     ERRMSG("Too many levels!");
00574   for (ip = 0; ip < ctl->met_np; ip++)
00575     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00576
00577   /* Isosurface parameters... */
00578   ctl->isosurf
00579     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00580   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00581
00582   /* Diffusion parameters... */
00583   ctl->turb_dx_trop
00584     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00585   ctl->turb_dx_strat
00586     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00587   ctl->turb_dz_trop
00588     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00589   ctl->turb_dz_strat
00590     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00591   ctl->turb_meso =
00592     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00593
00594   /* Life time of particles... */
00595   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00596   ctl->tdec_strat =
00597     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00598
00599   /* PSC analysis... */
00600   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00601   ctl->psc_hno3 =
00602     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00603
00604   /* Output of atmospheric data... */
00605   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00606   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00607   ctl->atm_dt_out =
00608     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00609   ctl->atm_filter =
00610     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00611
00612   /* Output of CSI data... */
00613   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00614   ctl->csi_dt_out =
00615     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
00616   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00617            ctl->csi_obsfile);
00618   ctl->csi_obsmin =
00619     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00620   ctl->csi_modmin =
00621     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00622   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
00623   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
00624   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00625   ctl->csi_lon0 =
00626     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
00627   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00628   ctl->csi_nx =
00629     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
00630   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
00631   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
00632   ctl->csi_ny =
00633     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
```

```
00634
00635   /* Output of ensemble data... */
00636   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
00637
00638   /* Output of grid data... */
00639   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00640            ctl->grid_basename);
00641   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00642   ctl->grid_dt_out =
00643     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00644   ctl->grid_sparse =
00645     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00646   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
00647   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00648   ctl->grid_nz =
00649     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00650   ctl->grid_lon0 =
00651     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
00652   ctl->grid_lon1 =
00653     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00654   ctl->grid_nx =
00655     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00656   ctl->grid_lat0 =
00657     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
00658   ctl->grid_lat1 =
00659     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00660   ctl->grid_ny =
00661     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00662
00663   /* Output of profile data... */
00664   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00665            ctl->prof_basename);
00666   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
00667   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00668   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00669   ctl->prof_nz =
00670     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00671   ctl->prof_lon0 =
00672     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
00673   ctl->prof_lon1 =
00674     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00675   ctl->prof_nx =
00676     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00677   ctl->prof_lat0 =
00678     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
00679   ctl->prof_lat1 =
00680     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00681   ctl->prof_ny =
00682     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00683
00684   /* Output of station data... */
00685   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00686            ctl->stat_basename);
00687   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
00688   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
00689   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00690 }
```

Here is the call graph for this function:

5.13.2.19 void read_met (ctl_t * *ctl*, char * *filename*, met_t * *met*)

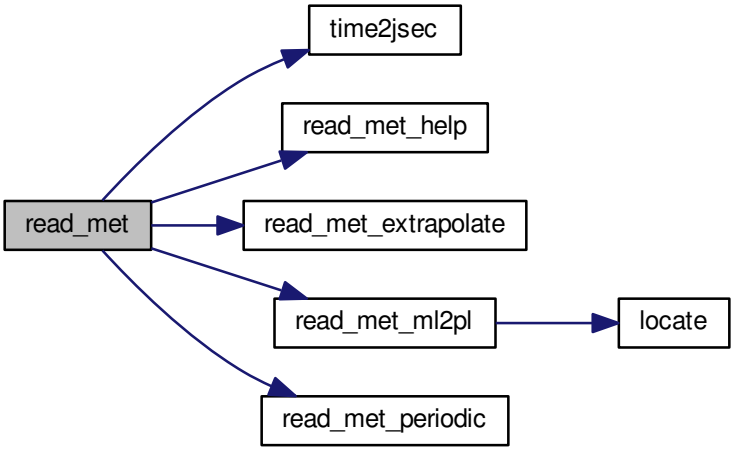
Read meteorological data file.

Definition at line 694 of file libtrac.c.

```
00697               {
00698
00699   char tstr[10];
00700
00701   static float help[EX * EY];
00702
00703   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00704
00705   size_t np, nx, ny;
00706
00707   /* Write info... */
00708   printf("Read meteorological data: %s\n", filename);
00709
00710   /* Get time from filename... */
00711   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00712   year = atoi(tstr);
00713   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00714   mon = atoi(tstr);
00715   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00716   day = atoi(tstr);
00717   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00718   hour = atoi(tstr);
00719   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00720
00721   /* Open netCDF file... */
00722   NC(nc_open(filename, NC_NOWRITE, &ncid));
00723
00724   /* Get dimensions... */
00725   NC(nc_inq_dimid(ncid, "lon", &dimid));
00726   NC(nc_inq_dimlen(ncid, dimid, &nx));
00727   if (nx > EX)
00728     ERRMSG("Too many longitudes!");
00729
00730   NC(nc_inq_dimid(ncid, "lat", &dimid));
00731   NC(nc_inq_dimlen(ncid, dimid, &ny));
00732   if (ny > EY)
00733     ERRMSG("Too many latitudes!");
00734
00735   NC(nc_inq_dimid(ncid, "lev", &dimid));
00736   NC(nc_inq_dimlen(ncid, dimid, &np));
00737   if (np > EP)
00738     ERRMSG("Too many levels!");
00739
00740   /* Store dimensions... */
00741   met->np = (int) np;
00742   met->nx = (int) nx;
00743   met->ny = (int) ny;
00744
00745   /* Get horizontal grid... */
00746   NC(nc_inq_varid(ncid, "lon", &varid));
00747   NC(nc_get_var_double(ncid, varid, met->lon));
00748   NC(nc_inq_varid(ncid, "lat", &varid));
00749   NC(nc_get_var_double(ncid, varid, met->lat));
00750
00751   /* Read meteorological data... */
00752   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00753   read_met_help(ncid, "u", "U", met, met->u, 1.0);
00754   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00755   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00756   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00757   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00758
00759   /* Meteo data on pressure levels... */
00760   if (ctl->met_np <= 0) {
00761
00762     /* Read pressure levels from file... */
00763     NC(nc_inq_varid(ncid, "lev", &varid));
00764     NC(nc_get_var_double(ncid, varid, met->p));
00765     for (ip = 0; ip < met->np; ip++)
00766       met->p[ip] /= 100.;
00767
00768     /* Extrapolate data for lower boundary... */
00769     read_met_extrapolate(met);
00770   }
00771
00772   /* Meteo data on model levels... */
00773   else {
00774
00775     /* Read pressure data from file... */
00776     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00777
00778     /* Interpolate from model levels to pressure levels... */
00779     read_met_ml2pl(ctl, met, met->t);
00780     read_met_ml2pl(ctl, met, met->u);
00781     read_met_ml2pl(ctl, met, met->v);
00782     read_met_ml2pl(ctl, met, met->w);
00783     read_met_ml2pl(ctl, met, met->h2o);
```

```
00784     read_met_ml2pl(ctl, met, met->o3);
00785
00786     /* Set pressure levels... */
00787     met->np = ctl->met_np;
00788     for (ip = 0; ip < met->np; ip++)
00789       met->p[ip] = ctl->met_p[ip];
00790   }
00791
00792   /* Check ordering of pressure levels... */
00793   for (ip = 1; ip < met->np; ip++)
00794     if (met->p[ip - 1] < met->p[ip])
00795       ERRMSG("Pressure levels must be descending!");
00796
00797   /* Read surface pressure... */
00798   if (nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00799     NC(nc_get_var_float(ncid, varid, help));
00800     for (iy = 0; iy < met->ny; iy++)
00801       for (ix = 0; ix < met->nx; ix++)
00802         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00803   } else if (nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00804     NC(nc_get_var_float(ncid, varid, help));
00805     for (iy = 0; iy < met->ny; iy++)
00806       for (ix = 0; ix < met->nx; ix++)
00807         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00808   } else
00809     for (ix = 0; ix < met->nx; ix++)
00810       for (iy = 0; iy < met->ny; iy++)
00811         met->ps[ix][iy] = met->p[0];
00812
00813   /* Create periodic boundary conditions... */
00814   read_met_periodic(met);
00815
00816   /* Close file... */
00817   NC(nc_close(ncid));
00818 }
```

Here is the call graph for this function:



5.13.2.20 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 822 of file libtrac.c.

```
00823               {
00824
00825   int ip, ip0, ix, iy;
00826
00827   /* Loop over columns... */
00828   for (ix = 0; ix < met->nx; ix++)
00829     for (iy = 0; iy < met->ny; iy++) {
00830
00831       /* Find lowest valid data point... */
00832       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00833         if (!gsl_finite(met->t[ix][iy][ip0])
00834             || !gsl_finite(met->u[ix][iy][ip0])
00835             || !gsl_finite(met->v[ix][iy][ip0])
00836             || !gsl_finite(met->w[ix][iy][ip0]))
00837           break;
00838
00839       /* Extrapolate... */
00840       for (ip = ip0; ip >= 0; ip--) {
00841         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00842         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00843         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00844         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00845         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00846         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00847       }
00848     }
00849 }
```

5.13.2.21 void read_met_help (int *ncid*, char * *varname*, char * *varname2*, met_t * *met*, float *dest[EX][EY][EP]*, float *scl*)

Read and convert variable from meteorological data file.

Definition at line 853 of file libtrac.c.

```
00859             {
00860
00861   static float help[EX * EY * EP];
00862
00863   int ip, ix, iy, n = 0, varid;
00864
00865   /* Check if variable exists... */
00866   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00867     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00868       return;
00869
00870   /* Read data... */
00871   NC(nc_get_var_float(ncid, varid, help));
00872
00873   /* Copy and check data... */
00874   for (ip = 0; ip < met->np; ip++)
00875     for (iy = 0; iy < met->ny; iy++)
00876       for (ix = 0; ix < met->nx; ix++) {
00877         dest[ix][iy][ip] = scl * help[n++];
00878         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00879           dest[ix][iy][ip] = GSL_NAN;
00880       }
00881 }
```

5.13.2.22 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

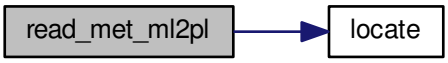
Convert meteorological data from model levels to pressure levels.

Definition at line 885 of file libtrac.c.

```
00888                         {
00889
00890   double aux[EP], p[EP], pt;
00891
00892   int ip, ip2, ix, iy;
00893
00894   /* Loop over columns... */
00895   for (ix = 0; ix < met->nx; ix++)
00896     for (iy = 0; iy < met->ny; iy++) {
00897
```

```
00898       /* Copy pressure profile... */
00899       for (ip = 0; ip < met->np; ip++)
00900         p[ip] = met->pl[ix][iy][ip];
00901
00902       /* Interpolate... */
00903       for (ip = 0; ip < ctl->met_np; ip++) {
00904         pt = ctl->met_p[ip];
00905         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00906           pt = p[0];
00907         else if ((pt > p[met->np - 1] && p[1] > p[0])
00908                  || (pt < p[met->np - 1] && p[1] < p[0]))
00909           pt = p[met->np - 1];
00910         ip2 = locate(p, met->np, pt);
00911         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00912                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00913       }
00914
00915       /* Copy data... */
00916       for (ip = 0; ip < ctl->met_np; ip++)
00917         var[ix][iy][ip] = (float) aux[ip];
00918     }
00919 }
```

Here is the call graph for this function:



5.13.2.23 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 923 of file libtrac.c.

```
00924               {
00925
00926   int ip, iy;
00927
00928   /* Check longitudes... */
00929   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00930              + met->lon[1] - met->lon[0] - 360) < 0.01))
00931     return;
00932
00933   /* Increase longitude counter... */
00934   if ((++met->nx) > EX)
00935     ERRMSG("Cannot create periodic boundary conditions!");
00936
00937   /* Set longitude... */
00938   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
00939
00940   /* Loop over latitudes and pressure levels... */
00941   for (iy = 0; iy < met->ny; iy++)
00942     for (ip = 0; ip < met->np; ip++) {
00943       met->ps[met->nx - 1][iy] = met->ps[0][iy];
00944       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
00945       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
00946       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
00947       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
00948       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
00949       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
00950     }
00951 }
```

5.13.2.24 double scan_ctl (const char * *filename*, int *argc*, char * *argv[]*, const char * *varname*, int *arridx*, const char * *defvalue*, char * *value*)

Read a control parameter from file or command line.

Definition at line 955 of file libtrac.c.

```
00962               {
00963
00964   FILE *in = NULL;
00965
00966   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
00967     msg[LEN], rvarname[LEN], rval[LEN];
00968
00969   int contain = 0, i;
00970
00971   /* Open file... */
00972   if (filename[strlen(filename) - 1] != '-')
00973     if (!(in = fopen(filename, "r")))
00974       ERRMSG("Cannot open file!");
00975
00976   /* Set full variable name... */
00977   if (arridx >= 0) {
00978     sprintf(fullname1, "%s[%d]", varname, arridx);
00979     sprintf(fullname2, "%s[*]", varname);
00980   } else {
00981     sprintf(fullname1, "%s", varname);
00982     sprintf(fullname2, "%s", varname);
00983   }
00984
00985   /* Read data... */
00986   if (in != NULL)
00987     while (fgets(line, LEN, in))
00988       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
00989         if (strcasecmp(rvarname, fullname1) == 0 ||
00990             strcasecmp(rvarname, fullname2) == 0) {
00991           contain = 1;
00992           break;
00993         }
00994   for (i = 1; i < argc - 1; i++)
00995     if (strcasecmp(argv[i], fullname1) == 0 ||
00996         strcasecmp(argv[i], fullname2) == 0) {
00997       sprintf(rval, "%s", argv[i + 1]);
00998       contain = 1;
00999       break;
01000     }
01001
01002   /* Close file... */
01003   if (in != NULL)
01004     fclose(in);
01005
01006   /* Check for missing variables... */
01007   if (!contain) {
01008     if (strlen(defvalue) > 0)
01009       sprintf(rval, "%s", defvalue);
01010     else {
01011       sprintf(msg, "Missing variable %s!\n", fullname1);
01012       ERRMSG(msg);
01013     }
01014   }
01015
01016   /* Write info... */
01017   printf("%s = %s\n", fullname1, rval);
01018
01019   /* Return values... */
01020   if (value != NULL)
01021     sprintf(value, "%s", rval);
01022   return atof(rval);
01023 }
```

5.13.2.25 void time2jsec (int *year*, int *mon*, int *day*, int *hour*, int *min*, int *sec*, double *remain*, double * *jsec*)

Convert date to seconds.

Definition at line 1027 of file libtrac.c.

```
01035                {
01036
01037   struct tm t0, t1;
01038
01039   t0.tm_year = 100;
01040   t0.tm_mon = 0;
01041   t0.tm_mday = 1;
01042   t0.tm_hour = 0;
01043   t0.tm_min = 0;
01044   t0.tm_sec = 0;
01045
01046   t1.tm_year = year - 1900;
01047   t1.tm_mon = mon - 1;
01048   t1.tm_mday = day;
01049   t1.tm_hour = hour;
01050   t1.tm_min = min;
01051   t1.tm_sec = sec;
01052
01053   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01054 }
```

5.13.2.26 void timer (const char * *name*, int *id*, int *mode*)

Measure wall-clock time.

Definition at line 1058 of file libtrac.c.

```
01061            {
01062
01063   static double starttime[NTIMER], runtime[NTIMER];
01064
01065   /* Check id... */
01066   if (id < 0 || id >= NTIMER)
01067     ERRMSG("Too many timers!");
01068
01069   /* Start timer... */
01070   if (mode == 1) {
01071     if (starttime[id] <= 0)
01072       starttime[id] = omp_get_wtime();
01073     else
01074       ERRMSG("Timer already started!");
01075   }
01076
01077   /* Stop timer... */
01078   else if (mode == 2) {
01079     if (starttime[id] > 0) {
01080       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01081       starttime[id] = -1;
01082     } else
01083       ERRMSG("Timer not started!");
01084   }
01085
01086   /* Print timer... */
01087   else if (mode == 3)
01088     printf("%s = %g s\n", name, runtime[id]);
01089 }
```

5.13.2.27 double tropopause (double *t*, double *lat*)

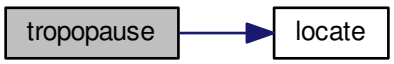
Definition at line 1093 of file libtrac.c.

```
01095              {
01096
01097   static double doys[12]
01098   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01099
01100   static double lats[73]
01101     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01102     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01103     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01104     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01105     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01106     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01107     75, 77.5, 80, 82.5, 85, 87.5, 90
01108   };
```

```
01109
01110   static double tps[12][73]
01111     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01112          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01113          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01114          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01115          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01116          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01117          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01118          275.3, 275.6, 275.4, 274.1, 273.5},
01119    {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01120     300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01121     150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01122     98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01123     98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01124     220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01125     284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01126     287.5, 286.2, 285.8},
01127    {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01128     297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01129     161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01130     100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01131     99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01132     186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01133     279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01134     304.3, 304.9, 306, 306.6, 306.2, 306},
01135    {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01136     290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01137     195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01138     102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01139     99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01140     148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01141     263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01142     315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01143    {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01144     260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01145     205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01146     101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01147     102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01148     165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01149     273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01150     325.3, 325.8, 325.8},
01151    {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01152     222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01153     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01154     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01155     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01156     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01157     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01158     308.5, 312.2, 313.1, 313.3},
01159    {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01160     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01161     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01162     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01163     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01164     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01165     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01166     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01167    {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01168     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01169     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01170     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01171     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01172     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01173     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01174     278.2, 282.6, 287.4, 290.9, 292.5, 293},
01175    {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01176     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01177     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01178     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01179     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01180     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01181     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01182     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01183    {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01184     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01185     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01186     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01187     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01188     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01189     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01190     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01191     305.1},
01192    {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01193     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01194     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01195     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
```

```
01196     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01197     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01198     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01199     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01200    {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01201     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01202     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01203     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01204     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01205     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01206     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01207     281.7, 281.1, 281.2}
01208    };
01209
01210    double doy, p0, p1, pt;
01211
01212    int imon, ilat;
01213
01214    /* Get day of year... */
01215    doy = fmod(t / 86400., 365.25);
01216    while (doy < 0)
01217      doy += 365.25;
01218
01219    /* Get indices... */
01220    imon = locate(doys, 12, doy);
01221    ilat = locate(lats, 73, lat);
01222
01223    /* Get tropopause pressure... */
01224    p0 = LIN(lats[ilat], tps[imon][ilat],
01225             lats[ilat + 1], tps[imon][ilat + 1], lat);
01226    p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01227             lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01228    pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01229
01230    /* Return tropopause pressure... */
01231    return pt;
01232  }
```

Here is the call graph for this function:



5.13.2.28 void write_atm (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

Write atmospheric data.

Definition at line 1236 of file libtrac.c.

```
01240             {
01241
01242    FILE *in, *out;
01243
01244    char line[LEN];
01245
01246    double r, t0, t1;
01247
01248    int ip, iq, year, mon, day, hour, min, sec;
01249
01250    /* Set time interval for output... */
01251    t0 = t - 0.5 * ctl->dt_mod;
01252    t1 = t + 0.5 * ctl->dt_mod;
01253
01254    /* Check if gnuplot output is requested... */
01255    if (ctl->atm_gpfile[0] != '-') {
```

```
01256
01257     /* Write info... */
01258     printf("Plot atmospheric data: %s.png\n", filename);
01259
01260     /* Create gnuplot pipe... */
01261     if (!(out = popen("gnuplot", "w")))
01262       ERRMSG("Cannot create pipe to gnuplot!");
01263
01264     /* Set plot filename... */
01265     fprintf(out, "set out \"%s.png\"\n", filename);
01266
01267     /* Set time string... */
01268     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01269     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01270             year, mon, day, hour, min);
01271
01272     /* Dump gnuplot file to pipe... */
01273     if (!(in = fopen(ctl->atm_gpfile, "r")))
01274       ERRMSG("Cannot open file!");
01275     while (fgets(line, LEN, in))
01276       fprintf(out, "%s", line);
01277     fclose(in);
01278   }
01279
01280   else {
01281
01282     /* Write info... */
01283     printf("Write atmospheric data: %s\n", filename);
01284
01285     /* Create file... */
01286     if (!(out = fopen(filename, "w")))
01287       ERRMSG("Cannot create file!");
01288   }
01289
01290   /* Write header... */
01291   fprintf(out,
01292           "# $1 = time [s]\n"
01293           "# $2 = altitude [km]\n"
01294           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01295   for (iq = 0; iq < ctl->nq; iq++)
01296     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01297             ctl->qnt_unit[iq]);
01298   fprintf(out, "\n");
01299
01300   /* Write data... */
01301   for (ip = 0; ip < atm->np; ip++) {
01302
01303     /* Check time... */
01304     if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01305       continue;
01306
01307     /* Write output... */
01308     fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01309             atm->lon[ip], atm->lat[ip]);
01310     for (iq = 0; iq < ctl->nq; iq++) {
01311       fprintf(out, " ");
01312       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01313     }
01314     fprintf(out, "\n");
01315   }
01316
01317   /* Close file... */
01318   fclose(out);
01319 }
```

Here is the call graph for this function:

5.13.2.29 void write_csi (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

Write CSI data.

Definition at line 1323 of file libtrac.c.

```
01327           {
01328
01329   static FILE *in, *out;
01330
01331   static char line[LEN];
01332
01333   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01334     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01335
01336   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01337
01338   /* Init... */
01339   if (!init) {
01340     init = 1;
01341
01342     /* Check quantity index for mass... */
01343     if (ctl->qnt_m < 0)
01344       ERRMSG("Need quantity mass to analyze CSI!");
01345
01346     /* Open observation data file... */
01347     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01348     if (!(in = fopen(ctl->csi_obsfile, "r")))
01349       ERRMSG("Cannot open file!");
01350
01351     /* Create new file... */
01352     printf("Write CSI data: %s\n", filename);
01353     if (!(out = fopen(filename, "w")))
01354       ERRMSG("Cannot create file!");
01355
01356     /* Write header... */
01357     fprintf(out,
01358             "# $1 = time [s]\n"
01359             "# $2 = number of hits (cx)\n"
01360             "# $3 = number of misses (cy)\n"
01361             "# $4 = number of false alarms (cz)\n"
01362             "# $5 = number of observations (cx + cy)\n"
01363             "# $6 = number of forecasts (cx + cz)\n"
01364             "# $7 = bias (forecasts/observations) [%%]\n"
01365             "# $8 = probability of detection (POD) [%%]\n"
01366             "# $9 = false alarm rate (FAR) [%%]\n"
01367             "# $10 = critical success index (CSI) [%%]\n\n");
01368   }
01369
01370   /* Set time interval... */
01371   t0 = t - 0.5 * ctl->dt_mod;
01372   t1 = t + 0.5 * ctl->dt_mod;
01373
01374   /* Initialize grid cells... */
01375   for (ix = 0; ix < ctl->csi_nx; ix++)
01376     for (iy = 0; iy < ctl->csi_ny; iy++)
01377       for (iz = 0; iz < ctl->csi_nz; iz++)
01378         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01379
01380   /* Read data... */
01381   while (fgets(line, LEN, in)) {
01382
01383     /* Read data... */
01384     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01385         5)
01386       continue;
01387
01388     /* Check time... */
01389     if (rt < t0)
01390       continue;
01391     if (rt > t1)
01392       break;
01393
01394     /* Calculate indices... */
01395     ix = (int) ((rlon - ctl->csi_lon0)
01396                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01397     iy = (int) ((rlat - ctl->csi_lat0)
01398                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01399     iz = (int) ((rz - ctl->csi_z0)
01400                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01401
01402     /* Check indices... */
01403     if (ix < 0 || ix >= ctl->csi_nx ||
```

```
01404         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01405       continue;
01406
01407     /* Get mean observation index... */
01408     obsmean[ix][iy][iz] += robs;
01409     obscount[ix][iy][iz]++;
01410   }
01411
01412   /* Analyze model data... */
01413   for (ip = 0; ip < atm->np; ip++) {
01414
01415     /* Check time... */
01416     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01417       continue;
01418
01419     /* Get indices... */
01420     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01421                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01422     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01423                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01424     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01425                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01426
01427     /* Check indices... */
01428     if (ix < 0 || ix >= ctl->csi_nx ||
01429         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01430       continue;
01431
01432     /* Get total mass in grid cell... */
01433     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01434   }
01435
01436   /* Analyze all grid cells... */
01437   for (ix = 0; ix < ctl->csi_nx; ix++)
01438     for (iy = 0; iy < ctl->csi_ny; iy++)
01439       for (iz = 0; iz < ctl->csi_nz; iz++) {
01440
01441         /* Calculate mean observation index... */
01442         if (obscount[ix][iy][iz] > 0)
01443           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01444
01445         /* Calculate column density... */
01446         if (modmean[ix][iy][iz] > 0) {
01447           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01448           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01449           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01450           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01451             * cos(lat * M_PI / 180.);
01452           modmean[ix][iy][iz] /= (1e6 * area);
01453         }
01454
01455         /* Calculate CSI... */
01456         if (obscount[ix][iy][iz] > 0) {
01457           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01458               modmean[ix][iy][iz] >= ctl->csi_modmin)
01459             cx++;
01460           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01461                    modmean[ix][iy][iz] < ctl->csi_modmin)
01462             cy++;
01463           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01464                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01465             cz++;
01466         }
01467       }
01468
01469   /* Write output... */
01470   if (fmod(t, ctl->csi_dt_out) == 0) {
01471
01472     /* Write... */
01473     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01474             t, cx, cy, cz, cx + cy, cx + cz,
01475             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01476             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01477             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01478             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01479
01480     /* Set counters to zero... */
01481     cx = cy = cz = 0;
01482   }
01483
01484   /* Close file... */
01485   if (t == ctl->t_stop)
01486     fclose(out);
01487 }
```

5.13.2.30 void write_ens (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

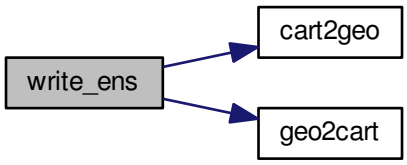
Write ensemble data.

Definition at line 1491 of file libtrac.c.

```
01495           {
01496
01497   static FILE *out;
01498
01499   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01500     t0, t1, x[NENS][3], xm[3];
01501
01502   static int init, ip, iq;
01503
01504   static size_t i, n;
01505
01506   /* Init... */
01507   if (!init) {
01508     init = 1;
01509
01510     /* Check quantities... */
01511     if (ctl->qnt_ens < 0)
01512       ERRMSG("Missing ensemble IDs!");
01513
01514     /* Create new file... */
01515     printf("Write ensemble data: %s\n", filename);
01516     if (!(out = fopen(filename, "w")))
01517       ERRMSG("Cannot create file!");
01518
01519     /* Write header... */
01520     fprintf(out,
01521             "# $1 = time [s]\n"
01522             "# $2 = altitude [km]\n"
01523             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01524     for (iq = 0; iq < ctl->nq; iq++)
01525       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01526               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01527     for (iq = 0; iq < ctl->nq; iq++)
01528       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01529               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01530     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01531   }
01532
01533   /* Set time interval... */
01534   t0 = t - 0.5 * ctl->dt_mod;
01535   t1 = t + 0.5 * ctl->dt_mod;
01536
01537   /* Init... */
01538   ens = GSL_NAN;
01539   n = 0;
01540
01541   /* Loop over air parcels... */
01542   for (ip = 0; ip < atm->np; ip++) {
01543
01544     /* Check time... */
01545     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01546       continue;
01547
01548     /* Check ensemble id... */
01549     if (atm->q[ctl->qnt_ens][ip] != ens) {
01550
01551       /* Write results... */
01552       if (n > 0) {
01553
01554         /* Get mean position... */
01555         xm[0] = xm[1] = xm[2] = 0;
01556         for (i = 0; i < n; i++) {
01557           xm[0] += x[i][0] / (double) n;
01558           xm[1] += x[i][1] / (double) n;
01559           xm[2] += x[i][2] / (double) n;
01560         }
01561         cart2geo(xm, &dummy, &lon, &lat);
01562         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01563                 lat);
01564
01565         /* Get quantity statistics... */
01566         for (iq = 0; iq < ctl->nq; iq++) {
01567           fprintf(out, " ");
01568           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01569         }
01570         for (iq = 0; iq < ctl->nq; iq++) {
01571           fprintf(out, " ");
```

```
01572           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01573         }
01574         fprintf(out, " %lu\n", n);
01575       }
01576
01577       /* Init new ensemble... */
01578       ens = atm->q[ctl->qnt_ens][ip];
01579       n = 0;
01580     }
01581
01582     /* Save data... */
01583     p[n] = atm->p[ip];
01584     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01585     for (iq = 0; iq < ctl->nq; iq++)
01586       q[iq][n] = atm->q[iq][ip];
01587     if ((++n) >= NENS)
01588       ERRMSG("Too many data points!");
01589   }
01590
01591   /* Write results... */
01592   if (n > 0) {
01593
01594     /* Get mean position... */
01595     xm[0] = xm[1] = xm[2] = 0;
01596     for (i = 0; i < n; i++) {
01597       xm[0] += x[i][0] / (double) n;
01598       xm[1] += x[i][1] / (double) n;
01599       xm[2] += x[i][2] / (double) n;
01600     }
01601     cart2geo(xm, &dummy, &lon, &lat);
01602     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01603
01604     /* Get quantity statistics... */
01605     for (iq = 0; iq < ctl->nq; iq++) {
01606       fprintf(out, " ");
01607       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01608     }
01609     for (iq = 0; iq < ctl->nq; iq++) {
01610       fprintf(out, " ");
01611       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01612     }
01613     fprintf(out, " %lu\n", n);
01614   }
01615
01616   /* Close file... */
01617   if (t == ctl->t_stop)
01618     fclose(out);
01619 }
```

Here is the call graph for this function:



5.13.2.31 void write_grid (const char * *filename*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

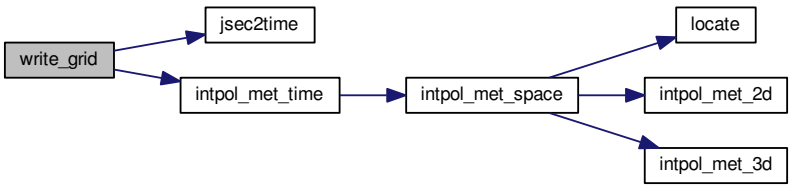
Write gridded data.

Definition at line 1623 of file libtrac.c.

```
01629           {
01630
01631   FILE *in, *out;
01632
01633   char line[LEN];
01634
01635   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01636     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01637
01638   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01639
01640   /* Check dimensions... */
01641   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01642     ERRMSG("Grid dimensions too large!");
01643
01644   /* Check quantity index for mass... */
01645   if (ctl->qnt_m < 0)
01646     ERRMSG("Need quantity mass to write grid data!");
01647
01648   /* Set time interval for output... */
01649   t0 = t - 0.5 * ctl->dt_mod;
01650   t1 = t + 0.5 * ctl->dt_mod;
01651
01652   /* Set grid box size... */
01653   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01654   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01655   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01656
01657   /* Initialize grid... */
01658   for (ix = 0; ix < ctl->grid_nx; ix++)
01659     for (iy = 0; iy < ctl->grid_ny; iy++)
01660       for (iz = 0; iz < ctl->grid_nz; iz++)
01661         grid_m[ix][iy][iz] = 0;
01662
01663   /* Average data... */
01664   for (ip = 0; ip < atm->np; ip++)
01665     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01666
01667       /* Get index... */
01668       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01669       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01670       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01671
01672       /* Check indices... */
01673       if (ix < 0 || ix >= ctl->grid_nx ||
01674           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01675         continue;
01676
01677       /* Add mass... */
01678       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01679     }
01680
01681   /* Check if gnuplot output is requested... */
01682   if (ctl->grid_gpfile[0] != '-') {
01683
01684     /* Write info... */
01685     printf("Plot grid data: %s.png\n", filename);
01686
01687     /* Create gnuplot pipe... */
01688     if (!(out = popen("gnuplot", "w")))
01689       ERRMSG("Cannot create pipe to gnuplot!");
01690
01691     /* Set plot filename... */
01692     fprintf(out, "set out \"%s.png\"\n", filename);
01693
01694     /* Set time string... */
01695     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01696     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01697             year, mon, day, hour, min);
01698
01699     /* Dump gnuplot file to pipe... */
01700     if (!(in = fopen(ctl->grid_gpfile, "r")))
01701       ERRMSG("Cannot open file!");
01702     while (fgets(line, LEN, in))
01703       fprintf(out, "%s", line);
01704     fclose(in);
01705   }
01706
01707   else {
01708
01709     /* Write info... */
01710     printf("Write grid data: %s\n", filename);
01711
01712     /* Create file... */
01713     if (!(out = fopen(filename, "w")))
01714       ERRMSG("Cannot create file!");
01715   }
```

```
01716
01717   /* Write header... */
01718   fprintf(out,
01719           "# $1 = time [s]\n"
01720           "# $2 = altitude [km]\n"
01721           "# $3 = longitude [deg]\n"
01722           "# $4 = latitude [deg]\n"
01723           "# $5 = surface area [km^2]\n"
01724           "# $6 = layer width [km]\n"
01725           "# $7 = temperature [K]\n"
01726           "# $8 = column density [kg/m^2]\n"
01727           "# $9 = mass mixing ratio [1]\n\n");
01728
01729   /* Write data... */
01730   for (ix = 0; ix < ctl->grid_nx; ix++) {
01731     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01732       fprintf(out, "\n");
01733     for (iy = 0; iy < ctl->grid_ny; iy++) {
01734       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01735         fprintf(out, "\n");
01736       for (iz = 0; iz < ctl->grid_nz; iz++)
01737         if (!ctl->grid_sparse
01738             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01739
01740           /* Set coordinates... */
01741           z = ctl->grid_z0 + dz * (iz + 0.5);
01742           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01743           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01744
01745           /* Get pressure and temperature... */
01746           press = P(z);
01747           intpol_met_time(met0, met1, t, press, lon, lat,
01748                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01749
01750           /* Calculate surface area... */
01751           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01752             * cos(lat * M_PI / 180.);
01753
01754           /* Calculate column density... */
01755           cd = grid_m[ix][iy][iz] / (1e6 * area);
01756
01757           /* Calculate mass mixing ratio... */
01758           rho_air = 100. * press / (287.058 * temp);
01759           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01760
01761           /* Write output... */
01762           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01763                   t, z, lon, lat, area, dz, temp, cd, mmr);
01764         }
01765     }
01766   }
01767
01768   /* Close file... */
01769   fclose(out);
01770 }
```

Here is the call graph for this function:



5.13.2.32 void write_prof (const char * *filename*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

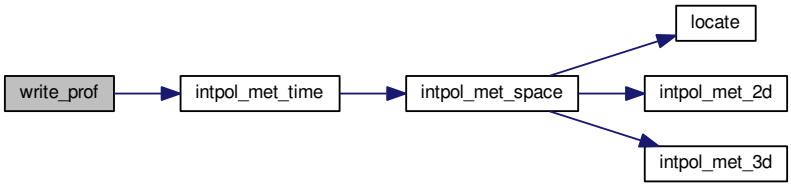
Write profile data.

Definition at line 1774 of file libtrac.c.

```
01780           {
01781
01782   static FILE *in, *out;
01783
01784   static char line[LEN];
01785
01786   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01787     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01788     press, temp, rho_air, mmr, h2o, o3;
01789
01790   static int init, obscount[GX][GY], ip, ix, iy, iz;
01791
01792   /* Init... */
01793   if (!init) {
01794     init = 1;
01795
01796     /* Check quantity index for mass... */
01797     if (ctl->qnt_m < 0)
01798       ERRMSG("Need quantity mass!");
01799
01800     /* Check dimensions... */
01801     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01802       ERRMSG("Grid dimensions too large!");
01803
01804     /* Open observation data file... */
01805     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01806     if (!(in = fopen(ctl->prof_obsfile, "r")))
01807       ERRMSG("Cannot open file!");
01808
01809     /* Create new file... */
01810     printf("Write profile data: %s\n", filename);
01811     if (!(out = fopen(filename, "w")))
01812       ERRMSG("Cannot create file!");
01813
01814     /* Write header... */
01815     fprintf(out,
01816             "# $1  = time [s]\n"
01817             "# $2  = altitude [km]\n"
01818             "# $3  = longitude [deg]\n"
01819             "# $4  = latitude [deg]\n"
01820             "# $5  = pressure [hPa]\n"
01821             "# $6  = temperature [K]\n"
01822             "# $7  = mass mixing ratio [1]\n"
01823             "# $8  = H2O volume mixing ratio [1]\n"
01824             "# $9  = O3 volume mixing ratio [1]\n"
01825             "# $10 = mean BT index [K]\n");
01826
01827     /* Set grid box size... */
01828     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01829     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01830     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01831   }
01832
01833   /* Set time interval... */
01834   t0 = t - 0.5 * ctl->dt_mod;
01835   t1 = t + 0.5 * ctl->dt_mod;
01836
01837   /* Initialize... */
01838   for (ix = 0; ix < ctl->prof_nx; ix++)
01839     for (iy = 0; iy < ctl->prof_ny; iy++) {
01840       obsmean[ix][iy] = 0;
01841       obscount[ix][iy] = 0;
01842       tmean[ix][iy] = 0;
01843       for (iz = 0; iz < ctl->prof_nz; iz++)
01844         mass[ix][iy][iz] = 0;
01845     }
01846
01847   /* Read data... */
01848   while (fgets(line, LEN, in)) {
01849
01850     /* Read data... */
01851     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01852       continue;
01853
01854     /* Check time... */
01855     if (rt < t0)
01856       continue;
01857     if (rt > t1)
01858       break;
01859
01860     /* Calculate indices... */
01861     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01862     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01863
01864     /* Check indices... */
01865     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01866       continue;
```

```
01867
01868     /* Get mean observation index... */
01869     obsmean[ix][iy] += robs;
01870     tmean[ix][iy] += rt;
01871     obscount[ix][iy]++;
01872   }
01873
01874   /* Analyze model data... */
01875   for (ip = 0; ip < atm->np; ip++) {
01876
01877     /* Check time... */
01878     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01879       continue;
01880
01881     /* Get indices... */
01882     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01883     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01884     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01885
01886     /* Check indices... */
01887     if (ix < 0 || ix >= ctl->prof_nx ||
01888         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01889       continue;
01890
01891     /* Get total mass in grid cell... */
01892     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01893   }
01894
01895   /* Extract profiles... */
01896   for (ix = 0; ix < ctl->prof_nx; ix++)
01897     for (iy = 0; iy < ctl->prof_ny; iy++)
01898       if (obscount[ix][iy] > 0) {
01899
01900         /* Write output... */
01901         fprintf(out, "\n");
01902
01903         /* Loop over altitudes... */
01904         for (iz = 0; iz < ctl->prof_nz; iz++) {
01905
01906           /* Set coordinates... */
01907           z = ctl->prof_z0 + dz * (iz + 0.5);
01908           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01909           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01910
01911           /* Get meteorological data... */
01912           press = P(z);
01913           intpol_met_time(met0, met1, t, press, lon, lat,
01914                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
01915
01916           /* Calculate mass mixing ratio... */
01917           rho_air = 100. * press / (287.058 * temp);
01918           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
01919             * cos(lat * M_PI / 180.);
01920           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
01921
01922           /* Write output... */
01923           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
01924                   tmean[ix][iy] / obscount[ix][iy],
01925                   z, lon, lat, press, temp, mmr, h2o, o3,
01926                   obsmean[ix][iy] / obscount[ix][iy]);
01927         }
01928       }
01929
01930   /* Close file... */
01931   if (t == ctl->t_stop)
01932     fclose(out);
01933 }
```

Here is the call graph for this function:



5.13.2.33 void write_station (const char * *filename*, ctl_t * *ctl*, atm_t * *atm*, double *t*)

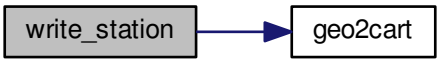
Write station data.

Definition at line 1937 of file libtrac.c.

```
01941            {
01942
01943   static FILE *out;
01944
01945   static double rmax2, t0, t1, x0[3], x1[3];
01946
01947   static int init, ip, iq;
01948
01949   /* Init... */
01950   if (!init) {
01951     init = 1;
01952
01953     /* Write info... */
01954     printf("Write station data: %s\n", filename);
01955
01956     /* Create new file... */
01957     if (!(out = fopen(filename, "w")))
01958       ERRMSG("Cannot create file!");
01959
01960     /* Write header... */
01961     fprintf(out,
01962             "# $1 = time [s]\n"
01963             "# $2 = altitude [km]\n"
01964             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01965     for (iq = 0; iq < ctl->nq; iq++)
01966       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
01967               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01968     fprintf(out, "\n");
01969
01970     /* Set geolocation and search radius... */
01971     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
01972     rmax2 = gsl_pow_2(ctl->stat_r);
01973   }
01974
01975   /* Set time interval for output... */
01976   t0 = t - 0.5 * ctl->dt_mod;
01977   t1 = t + 0.5 * ctl->dt_mod;
01978
01979   /* Loop over air parcels... */
01980   for (ip = 0; ip < atm->np; ip++) {
01981
01982     /* Check time... */
01983     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01984       continue;
01985
01986     /* Check station flag... */
01987     if (ctl->qnt_stat >= 0)
01988       if (atm->q[ctl->qnt_stat][ip])
01989         continue;
01990
01991     /* Get Cartesian coordinates... */
01992     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
01993
01994     /* Check horizontal distance... */
```

```
01995     if (DIST2(x0, x1) > rmax2)
01996       continue;
01997
01998     /* Set station flag... */
01999     if (ctl->qnt_stat >= 0)
02000       atm->q[ctl->qnt_stat][ip] = 1;
02001
02002     /* Write data... */
02003     fprintf(out, "%.2f %g %g %g",
02004             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02005     for (iq = 0; iq < ctl->nq; iq++) {
02006       fprintf(out, " ");
02007       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02008     }
02009     fprintf(out, "\n");
02010   }
02011
02012   /* Close file... */
02013   if (t == ctl->t_stop)
02014     fclose(out);
02015 }
```

Here is the call graph for this function:



5.14 libtrac.h

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00034 #include <ctype.h>
00035 #include <gsl/gsl_const_mksa.h>
00036 #include <gsl/gsl_math.h>
00037 #include <gsl/gsl_randist.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <gsl/gsl_sort.h>
00040 #include <gsl/gsl_statistics.h>
00041 #include <math.h>
00042 #include <netcdf.h>
00043 #include <omp.h>
00044 #include <stdio.h>
00045 #include <stdlib.h>
00046 #include <string.h>
00047 #include <time.h>
00048 #include <sys/time.h>
00049
00050 /* ------------------------------------------------------------
00051    Macros...
00052    ------------------------------------------------------------ */
00053
```

```
00055 #define ALLOC(ptr, type, n)                                \
00056   if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)       \
00057     ERRMSG("Out of memory!");
00058
00060 #define DIST(a, b) sqrt(DIST2(a, b))
00061
00063 #define DIST2(a, b)                                                \
00064   ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00065
00067 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00068
00070 #define ERRMSG(msg) {                                              \
00071     printf("\nError (%s, %s, l%d): %s\n\n",                        \
00072            __FILE__, __func__, __LINE__, msg);                     \
00073     exit(EXIT_FAILURE);                                            \
00074   }
00075
00077 #define LIN(x0, y0, x1, y1, x)                  \
00078   ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00079
00081 #define NC(cmd) {                                  \
00082     if((cmd)!=NC_NOERR)                            \
00083       ERRMSG(nc_strerror(cmd));                    \
00084   }
00085
00087 #define NORM(a) sqrt(DOTP(a, a))
00088
00090 #define PRINT(format, var)                                         \
00091   printf("Print (%s, %s, l%d): %s= "format"\n",                    \
00092          __FILE__, __func__, __LINE__, #var, var);
00093
00095 #define P(z) (P0*exp(-(z)/H0))
00096
00098 #define TOK(line, tok, format, var) {                              \
00099     if(((tok)=strtok((line), " \t"))) {                            \
00100       if(sscanf(tok, format, &(var))!=1) continue;                 \
00101     } else ERRMSG("Error while reading!");                         \
00102   }
00103
00105 #define Z(p) (H0*log(P0/(p)))
00106
00108 #define START_TIMER(id) timer(#id, id, 1)
00109
00111 #define STOP_TIMER(id) timer(#id, id, 2)
00112
00114 #define PRINT_TIMER(id) timer(#id, id, 3)
00115
00116 /* ------------------------------------------------------------
00117    Constants...
00118    ------------------------------------------------------------ */
00119
00121 #define G0 9.80665
00122
00124 #define H0 7.0
00125
00127 #define P0 1013.25
00128
00130 #define RE 6367.421
00131
00132 /* ------------------------------------------------------------
00133    Dimensions...
00134    ------------------------------------------------------------ */
00135
00137 #define LEN 5000
00138
00140 #define NP 10000000
00141
00143 #define NQ 10
00144
00146 #define EP 73
00147
00149 #define EX 721
00150
00152 #define EY 361
00153
00155 #define GX 720
00156
00158 #define GY 360
00159
00161 #define GZ 100
00162
00164 #define NENS 2000
00165
00167 #define NTHREADS 128
00168
00170 #define NTIMER 20
00171
```

```
00172 /* ----------------------------------------------------------
00173    Structs...
00174    ---------------------------------------------------------- */
00175
00177 typedef struct {
00178
00180   int nq;
00181
00183   char qnt_name[NQ][LEN];
00184
00186   char qnt_unit[NQ][LEN];
00187
00189   char qnt_format[NQ][LEN];
00190
00192   int qnt_ens;
00193
00195   int qnt_m;
00196
00198   int qnt_rho;
00199
00201   int qnt_r;
00202
00204   int qnt_ps;
00205
00207   int qnt_p;
00208
00210   int qnt_t;
00211
00213   int qnt_u;
00214
00216   int qnt_v;
00217
00219   int qnt_w;
00220
00222   int qnt_h2o;
00223
00225   int qnt_o3;
00226
00228   int qnt_theta;
00229
00231   int qnt_pv;
00232
00234   int qnt_tice;
00235
00237   int qnt_tsts;
00238
00240   int qnt_tnat;
00241
00243   int qnt_stat;
00244
00246   int direction;
00247
00249   double t_start;
00250
00252   double t_stop;
00253
00255   double dt_mod;
00256
00258   double dt_met;
00259
00261   int met_np;
00262
00264   double met_p[EP];
00265
00268   int isosurf;
00269
00271   char balloon[LEN];
00272
00274   double turb_dx_trop;
00275
00277   double turb_dx_strat;
00278
00280   double turb_dz_trop;
00281
00283   double turb_dz_strat;
00284
00286   double turb_meso;
00287
00289   double tdec_trop;
00290
00292   double tdec_strat;
00293
00295   double psc_h2o;
00296
00298   double psc_hno3;
00299
00301   char atm_basename[LEN];
```

```
00302
00304   char atm_gpfile[LEN];
00305
00307   double atm_dt_out;
00308
00310   int atm_filter;
00311
00313   char csi_basename[LEN];
00314
00316   double csi_dt_out;
00317
00319   char csi_obsfile[LEN];
00320
00322   double csi_obsmin;
00323
00325   double csi_modmin;
00326
00328   int csi_nz;
00329
00331   double csi_z0;
00332
00334   double csi_z1;
00335
00337   int csi_nx;
00338
00340   double csi_lon0;
00341
00343   double csi_lon1;
00344
00346   int csi_ny;
00347
00349   double csi_lat0;
00350
00352   double csi_lat1;
00353
00355   char grid_basename[LEN];
00356
00358   char grid_gpfile[LEN];
00359
00361   double grid_dt_out;
00362
00364   int grid_sparse;
00365
00367   int grid_nz;
00368
00370   double grid_z0;
00371
00373   double grid_z1;
00374
00376   int grid_nx;
00377
00379   double grid_lon0;
00380
00382   double grid_lon1;
00383
00385   int grid_ny;
00386
00388   double grid_lat0;
00389
00391   double grid_lat1;
00392
00394   char prof_basename[LEN];
00395
00397   char prof_obsfile[LEN];
00398
00400   int prof_nz;
00401
00403   double prof_z0;
00404
00406   double prof_z1;
00407
00409   int prof_nx;
00410
00412   double prof_lon0;
00413
00415   double prof_lon1;
00416
00418   int prof_ny;
00419
00421   double prof_lat0;
00422
00424   double prof_lat1;
00425
00427   char ens_basename[LEN];
00428
00430   char stat_basename[LEN];
00431
```

```
00433   double stat_lon;
00434
00436   double stat_lat;
00437
00439   double stat_r;
00440
00441 } ctl_t;
00442
00444 typedef struct {
00445
00447   int np;
00448
00450   double time[NP];
00451
00453   double p[NP];
00454
00456   double lon[NP];
00457
00459   double lat[NP];
00460
00462   double q[NQ][NP];
00463
00465   double up[NP];
00466
00468   double vp[NP];
00469
00471   double wp[NP];
00472
00473 } atm_t;
00474
00476 typedef struct {
00477
00479   double time;
00480
00482   int nx;
00483
00485   int ny;
00486
00488   int np;
00489
00491   double lon[EX];
00492
00494   double lat[EY];
00495
00497   double p[EP];
00498
00500   double ps[EX][EY];
00501
00503   float pl[EX][EY][EP];
00504
00506   float t[EX][EY][EP];
00507
00509   float u[EX][EY][EP];
00510
00512   float v[EX][EY][EP];
00513
00515   float w[EX][EY][EP];
00516
00518   float h2o[EX][EY][EP];
00519
00521   float o3[EX][EY][EP];
00522
00523 } met_t;
00524
00525 /* ------------------------------------------------------------
00526    Functions...
00527    ------------------------------------------------------------ */
00528
00530 void cart2geo(
00531   double *x,
00532   double *z,
00533   double *lon,
00534   double *lat);
00535
00537 double deg2dx(
00538   double dlon,
00539   double lat);
00540
00542 double deg2dy(
00543   double dlat);
00544
00546 double dp2dz(
00547   double dp,
00548   double p);
00549
00551 double dx2deg(
00552   double dx,
```

```
00553   double lat);
00554
00556 double dy2deg(
00557   double dy);
00558
00560 double dz2dp(
00561   double dz,
00562   double p);
00563
00565 void geo2cart(
00566   double z,
00567   double lon,
00568   double lat,
00569   double *x);
00570
00572 void get_met(
00573   ctl_t * ctl,
00574   char *metbase,
00575   double t,
00576   met_t * met0,
00577   met_t * met1);
00578
00580 void get_met_help(
00581   double t,
00582   int direct,
00583   char *metbase,
00584   double dt_met,
00585   char *filename);
00586
00588 void intpol_met_2d(
00589   double array[EX][EY],
00590   int ix,
00591   int iy,
00592   double wx,
00593   double wy,
00594   double *var);
00595
00597 void intpol_met_3d(
00598   float array[EX][EY][EP],
00599   int ip,
00600   int ix,
00601   int iy,
00602   double wp,
00603   double wx,
00604   double wy,
00605   double *var);
00606
00608 void intpol_met_space(
00609   met_t * met,
00610   double p,
00611   double lon,
00612   double lat,
00613   double *ps,
00614   double *t,
00615   double *u,
00616   double *v,
00617   double *w,
00618   double *h2o,
00619   double *o3);
00620
00622 void intpol_met_time(
00623   met_t * met0,
00624   met_t * met1,
00625   double ts,
00626   double p,
00627   double lon,
00628   double lat,
00629   double *ps,
00630   double *t,
00631   double *u,
00632   double *v,
00633   double *w,
00634   double *h2o,
00635   double *o3);
00636
00638 void jsec2time(
00639   double jsec,
00640   int *year,
00641   int *mon,
00642   int *day,
00643   int *hour,
00644   int *min,
00645   int *sec,
00646   double *remain);
00647
00649 int locate(
00650   double *xx,
```

```
00651   int n,
00652   double x);
00653
00655 void read_atm(
00656   const char *filename,
00657   ctl_t * ctl,
00658   atm_t * atm);
00659
00661 void read_ctl(
00662   const char *filename,
00663   int argc,
00664   char *argv[],
00665   ctl_t * ctl);
00666
00668 void read_met(
00669   ctl_t * ctl,
00670   char *filename,
00671   met_t * met);
00672
00674 void read_met_extrapolate(
00675   met_t * met);
00676
00678 void read_met_help(
00679   int ncid,
00680   char *varname,
00681   char *varname2,
00682   met_t * met,
00683   float dest[EX][EY][EP],
00684   float scl);
00685
00687 void read_met_ml2pl(
00688   ctl_t * ctl,
00689   met_t * met,
00690   float var[EX][EY][EP]);
00691
00693 void read_met_periodic(
00694   met_t * met);
00695
00697 double scan_ctl(
00698   const char *filename,
00699   int argc,
00700   char *argv[],
00701   const char *varname,
00702   int arridx,
00703   const char *defvalue,
00704   char *value);
00705
00707 void time2jsec(
00708   int year,
00709   int mon,
00710   int day,
00711   int hour,
00712   int min,
00713   int sec,
00714   double remain,
00715   double *jsec);
00716
00718 void timer(
00719   const char *name,
00720   int id,
00721   int mode);
00722
00723 /* Get tropopause pressure... */
00724 double tropopause(
00725   double t,
00726   double lat);
00727
00729 void write_atm(
00730   const char *filename,
00731   ctl_t * ctl,
00732   atm_t * atm,
00733   double t);
00734
00736 void write_csi(
00737   const char *filename,
00738   ctl_t * ctl,
00739   atm_t * atm,
00740   double t);
00741
00743 void write_ens(
00744   const char *filename,
00745   ctl_t * ctl,
00746   atm_t * atm,
00747   double t);
00748
00750 void write_grid(
00751   const char *filename,
```

```
00752   ctl_t * ctl,
00753   met_t * met0,
00754   met_t * met1,
00755   atm_t * atm,
00756   double t);
00757
00759 void write_prof(
00760   const char *filename,
00761   ctl_t * ctl,
00762   met_t * met0,
00763   met_t * met1,
00764   atm_t * atm,
00765   double t);
00766
00768 void write_station(
00769   const char *filename,
00770   ctl_t * ctl,
00771   atm_t * atm,
00772   double t);
```

5.15 match.c File Reference

Calculate deviations between two trajectories.

Functions

- int main (int argc, char ∗argv[ ])

5.15.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

5.15.2 Function Documentation

5.15.2.1 int main (int *argc,* char ∗ *argv[ ]*)

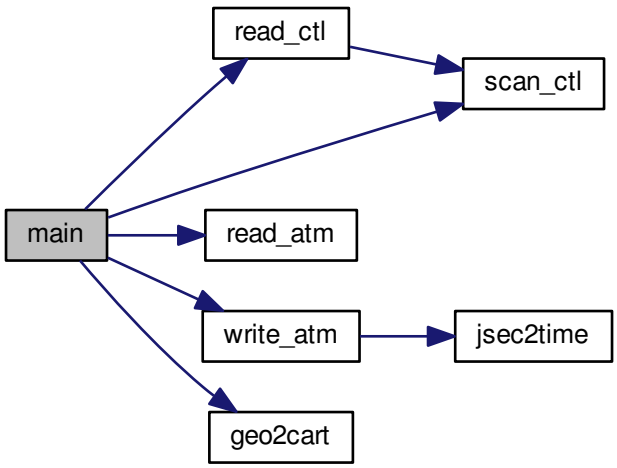
Definition at line 28 of file match.c.

```
00030                {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm1, *atm2, *atm3;
00035
00036   FILE *out;
00037
00038   char filename[LEN];
00039
00040   double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00041
00042   int filter, ip1, ip2, iq, n;
00043
00044   /* Allocate... */
00045   ALLOC(atm1, atm_t, 1);
00046   ALLOC(atm2, atm_t, 1);
00047   ALLOC(atm3, atm_t, 1);
00048
00049   /* Check arguments... */
00050   if (argc < 5)
00051     ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00052
00053   /* Read control parameters... */
00054   read_ctl(argv[1], argc, argv, &ctl);
```

```
00055   filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
00056   filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00057
00058   /* Read atmospheric data... */
00059   read_atm(argv[2], &ctl, atm1);
00060   read_atm(argv[3], &ctl, atm2);
00061
00062   /* Write info... */
00063   printf("Write transport deviations: %s\n", argv[4]);
00064
00065   /* Create output file... */
00066   if (!(out = fopen(argv[4], "w")))
00067     ERRMSG("Cannot create file!");
00068
00069   /* Write header... */
00070   fprintf(out,
00071           "# $1 = time [s]\n"
00072           "# $2 = altitude [km]\n"
00073           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00074   for (iq = 0; iq < ctl.nq; iq++)
00075     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00076             ctl.qnt_unit[iq]);
00077   fprintf(out,
00078           "# $%d = trajectory time [s]\n"
00079           "# $%d = vertical length of trajectory [km]\n"
00080           "# $%d = horizontal length of trajectory [km]\n"
00081           "# $%d = vertical deviation [km]\n"
00082           "# $%d = horizontal deviation [km]\n",
00083           5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
00084   for (iq = 0; iq < ctl.nq; iq++)
00085     fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,
00086             ctl.qnt_name[iq], ctl.qnt_unit[iq]);
00087   fprintf(out, "\n");
00088
00089   /* Filtering of reference time series... */
00090   if (filter) {
00091
00092     /* Copy data... */
00093     memcpy(atm3, atm2, sizeof(atm_t));
00094
00095     /* Loop over data points... */
00096     for (ip1 = 0; ip1 < atm2->np; ip1++) {
00097       n = 0;
00098       atm2->p[ip1] = 0;
00099       for (iq = 0; iq < ctl.nq; iq++)
00100         atm2->q[iq][ip1] = 0;
00101       for (ip2 = 0; ip2 < atm2->np; ip2++)
00102         if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
00103           atm2->p[ip1] += atm3->p[ip2];
00104           for (iq = 0; iq < ctl.nq; iq++)
00105             atm2->q[iq][ip1] += atm3->q[iq][ip2];
00106           n++;
00107         }
00108       atm2->p[ip1] /= n;
00109       for (iq = 0; iq < ctl.nq; iq++)
00110         atm2->q[iq][ip1] /= n;
00111     }
00112
00113     /* Write filtered data... */
00114     sprintf(filename, "%s.filt", argv[3]);
00115     write_atm(filename, &ctl, atm2, 0);
00116   }
00117
00118   /* Loop over air parcels (reference data)... */
00119   for (ip2 = 0; ip2 < atm2->np; ip2++) {
00120
00121     /* Get trajectory length... */
00122     if (ip2 > 0) {
00123       geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00124       geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00125       lh += DIST(x1, x2);
00126       lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
00127       lt = fabs(atm2->time[ip2] - atm2->time[0]);
00128     }
00129
00130     /* Init... */
00131     n = 0;
00132     dh = 0;
00133     dv = 0;
00134     for (iq = 0; iq < ctl.nq; iq++)
00135       dq[iq] = 0;
00136     geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00137
00138     /* Find corresponding time step (test data)... */
00139     for (ip1 = 0; ip1 < atm1->np; ip1++)
00140       if (fabs(atm1->time[ip1] - atm2->time[ip2])
00141           < (filter ? filter_dt : 0.1)) {
```

```
00142
00143         /* Calculate deviations... */
00144         geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00145         dh += DIST(x1, x2);
00146         dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00147         for (iq = 0; iq < ctl.nq; iq++)
00148           dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00149         n++;
00150       }
00151
00152     /* Write output... */
00153     if (n > 0) {
00154       fprintf(out, "%.2f %.4f %.4f %.4f",
00155               atm2->time[ip2], Z(atm2->p[ip2]),
00156               atm2->lon[ip2], atm2->lat[ip2]);
00157       for (iq = 0; iq < ctl.nq; iq++) {
00158         fprintf(out, " ");
00159         fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00160       }
00161       fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
00162       for (iq = 0; iq < ctl.nq; iq++) {
00163         fprintf(out, " ");
00164         fprintf(out, ctl.qnt_format[iq], dq[iq] / n);
00165       }
00166       fprintf(out, "\n");
00167     }
00168   }
00169
00170   /* Close file... */
00171   fclose(out);
00172
00173   /* Free... */
00174   free(atm1);
00175   free(atm2);
00176   free(atm3);
00177
00178   return EXIT_SUCCESS;
00179 }
```

Here is the call graph for this function:



5.16 match.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
```

```
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026 #include <gsl/gsl_sort.h>
00027
00028 int main(
00029   int argc,
00030   char *argv[]) {
00031
00032   ctl_t ctl;
00033
00034   atm_t *atm1, *atm2, *atm3;
00035
00036   FILE *out;
00037
00038   char filename[LEN];
00039
00040   double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00041
00042   int filter, ip1, ip2, iq, n;
00043
00044   /* Allocate... */
00045   ALLOC(atm1, atm_t, 1);
00046   ALLOC(atm2, atm_t, 1);
00047   ALLOC(atm3, atm_t, 1);
00048
00049   /* Check arguments... */
00050   if (argc < 5)
00051     ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00052
00053   /* Read control parameters... */
00054   read_ctl(argv[1], argc, argv, &ctl);
00055   filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
00056   filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00057
00058   /* Read atmospheric data... */
00059   read_atm(argv[2], &ctl, atm1);
00060   read_atm(argv[3], &ctl, atm2);
00061
00062   /* Write info... */
00063   printf("Write transport deviations: %s\n", argv[4]);
00064
00065   /* Create output file... */
00066   if (!(out = fopen(argv[4], "w")))
00067     ERRMSG("Cannot create file!");
00068
00069   /* Write header... */
00070   fprintf(out,
00071           "# $1 = time [s]\n"
00072           "# $2 = altitude [km]\n"
00073           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00074   for (iq = 0; iq < ctl.nq; iq++)
00075     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00076             ctl.qnt_unit[iq]);
00077   fprintf(out,
00078           "# $%d = trajectory time [s]\n"
00079           "# $%d = vertical length of trajectory [km]\n"
00080           "# $%d = horizontal length of trajectory [km]\n"
00081           "# $%d = vertical deviation [km]\n"
00082           "# $%d = horizontal deviation [km]\n",
00083           5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
00084   for (iq = 0; iq < ctl.nq; iq++)
00085     fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,
00086             ctl.qnt_name[iq], ctl.qnt_unit[iq]);
00087   fprintf(out, "\n");
00088
00089   /* Filtering of reference time series... */
00090   if (filter) {
00091
00092     /* Copy data... */
00093     memcpy(atm3, atm2, sizeof(atm_t));
00094
00095     /* Loop over data points... */
```

```
00096     for (ip1 = 0; ip1 < atm2->np; ip1++) {
00097       n = 0;
00098       atm2->p[ip1] = 0;
00099       for (iq = 0; iq < ctl.nq; iq++)
00100         atm2->q[iq][ip1] = 0;
00101       for (ip2 = 0; ip2 < atm2->np; ip2++)
00102         if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
00103           atm2->p[ip1] += atm3->p[ip2];
00104           for (iq = 0; iq < ctl.nq; iq++)
00105             atm2->q[iq][ip1] += atm3->q[iq][ip2];
00106           n++;
00107         }
00108       atm2->p[ip1] /= n;
00109       for (iq = 0; iq < ctl.nq; iq++)
00110         atm2->q[iq][ip1] /= n;
00111     }
00112
00113     /* Write filtered data... */
00114     sprintf(filename, "%s.filt", argv[3]);
00115     write_atm(filename, &ctl, atm2, 0);
00116   }
00117
00118   /* Loop over air parcels (reference data)... */
00119   for (ip2 = 0; ip2 < atm2->np; ip2++) {
00120
00121     /* Get trajectory length... */
00122     if (ip2 > 0) {
00123       geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00124       geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00125       lh += DIST(x1, x2);
00126       lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
00127       lt = fabs(atm2->time[ip2] - atm2->time[0]);
00128     }
00129
00130     /* Init... */
00131     n = 0;
00132     dh = 0;
00133     dv = 0;
00134     for (iq = 0; iq < ctl.nq; iq++)
00135       dq[iq] = 0;
00136     geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00137
00138     /* Find corresponding time step (test data)... */
00139     for (ip1 = 0; ip1 < atm1->np; ip1++)
00140       if (fabs(atm1->time[ip1] - atm2->time[ip2])
00141           < (filter ? filter_dt : 0.1)) {
00142
00143         /* Calculate deviations... */
00144         geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00145         dh += DIST(x1, x2);
00146         dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00147         for (iq = 0; iq < ctl.nq; iq++)
00148           dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00149         n++;
00150       }
00151
00152     /* Write output... */
00153     if (n > 0) {
00154       fprintf(out, "%.2f %.4f %.4f %.4f",
00155               atm2->time[ip2], Z(atm2->p[ip2]),
00156               atm2->lon[ip2], atm2->lat[ip2]);
00157       for (iq = 0; iq < ctl.nq; iq++) {
00158         fprintf(out, " ");
00159         fprintf(out, ctl.qnt_format[iq], atm2->q[iq][ip2]);
00160       }
00161       fprintf(out, " %.2f %g %g %g %g", lt, lv, lh, dv / n, dh / n);
00162       for (iq = 0; iq < ctl.nq; iq++) {
00163         fprintf(out, " ");
00164         fprintf(out, ctl.qnt_format[iq], dq[iq] / n);
00165       }
00166       fprintf(out, "\n");
00167     }
00168   }
00169
00170   /* Close file... */
00171   fclose(out);
00172
00173   /* Free... */
00174   free(atm1);
00175   free(atm2);
00176   free(atm3);
00177
00178   return EXIT_SUCCESS;
00179 }
```

5.17 met_map.c File Reference

Extract global map from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.17.1 Detailed Description

Extract global map from meteorological data.

Definition in file met_map.c.

5.17.2 Function Documentation

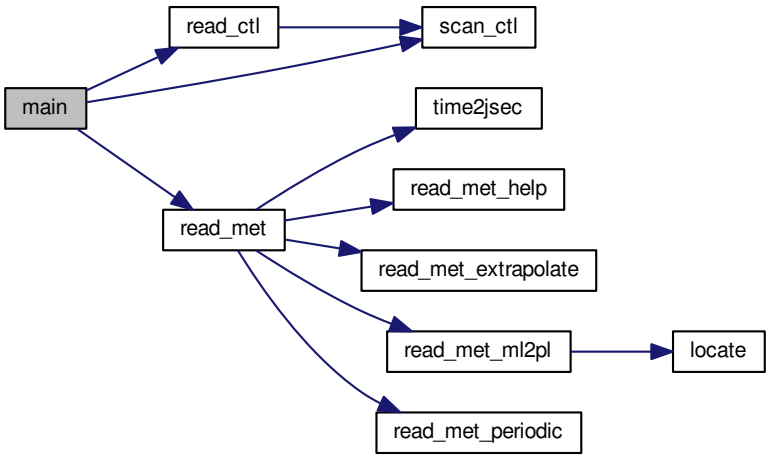
5.17.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 27 of file met_map.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *in, *out;
00036
00037   static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
00038     um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00039
00040   static int i, ip, ip2, ix, iy, np[EX][EY];
00041
00042   /* Allocate... */
00043   ALLOC(met, met_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 4)
00047     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00052
00053   /* Loop over files... */
00054   for (i = 3; i < argc; i++) {
00055
00056     /* Read meteorological data... */
00057     if (!(in = fopen(argv[i], "r")))
00058       continue;
00059     else
00060       fclose(in);
00061     read_met(&ctl, argv[i], met);
00062
00063     /* Find nearest pressure level... */
00064     for (ip2 = 0; ip2 < met->np; ip2++) {
00065       dz = fabs(Z(met->p[ip2]) - z);
00066       if (dz < dzmin) {
00067         dzmin = dz;
00068         ip = ip2;
00069       }
00070     }
00071
00072     /* Average data... */
00073     for (ix = 0; ix < met->nx; ix++)
00074       for (iy = 0; iy < met->ny; iy++) {
00075         timem[ix][iy] += met->time;
00076         tm[ix][iy] += met->t[ix][iy][ip];
```

```
00077         um[ix][iy] += met->u[ix][iy][ip];
00078         vm[ix][iy] += met->v[ix][iy][ip];
00079         wm[ix][iy] += met->w[ix][iy][ip];
00080         h2om[ix][iy] += met->h2o[ix][iy][ip];
00081         o3m[ix][iy] += met->o3[ix][iy][ip];
00082         psm[ix][iy] += met->ps[ix][iy];
00083         np[ix][iy]++;
00084       }
00085   }
00086
00087   /* Create output file... */
00088   printf("Write meteorological data file: %s\n", argv[2]);
00089   if (!(out = fopen(argv[2], "w")))
00090     ERRMSG("Cannot create file!");
00091
00092   /* Write header... */
00093   fprintf(out,
00094           "# $1  = time [s]\n"
00095           "# $2  = altitude [km]\n"
00096           "# $3  = longitude [deg]\n"
00097           "# $4  = latitude [deg]\n"
00098           "# $5  = pressure [hPa]\n"
00099           "# $6  = temperature [K]\n"
00100           "# $7  = zonal wind [m/s]\n"
00101           "# $8  = meridional wind [m/s]\n"
00102           "# $9  = vertical wind [hPa/s]\n"
00103           "# $10 = H2O volume mixing ratio [1]\n"
00104           "# $11 = O3 volume mixing ratio [1]\n"
00105           "# $12 = surface pressure [hPa]\n");
00106
00107   /* Write data... */
00108   for (iy = 0; iy < met->ny; iy++) {
00109     fprintf(out, "\n");
00110     for (ix = 0; ix < met->nx; ix++)
00111       if (met->lon[ix] >= 180)
00112         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00113                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00114                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00115                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00116                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00117                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00118                 psm[ix][iy] / np[ix][iy]);
00119     for (ix = 0; ix < met->nx; ix++)
00120       if (met->lon[ix] <= 180)
00121         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00122                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00123                 met->lon[ix], met->lat[iy], met->p[ip],
00124                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00125                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00126                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00127                 psm[ix][iy] / np[ix][iy]);
00128   }
00129
00130   /* Close file... */
00131   fclose(out);
00132
00133   /* Free... */
00134   free(met);
00135
00136   return EXIT_SUCCESS;
00137 }
```

Here is the call graph for this function:



5.18 met_map.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *in, *out;
00036
00037   static double dz, dzmin = 1e10, z, timem[EX][EY], psm[EX][EY], tm[EX][EY],
00038     um[EX][EY], vm[EX][EY], wm[EX][EY], h2om[EX][EY], o3m[EX][EY];
00039
00040   static int i, ip, ip2, ix, iy, np[EX][EY];
00041
00042   /* Allocate... */
00043   ALLOC(met, met_t, 1);
00044
00045   /* Check arguments... */
00046   if (argc < 4)
00047     ERRMSG("Give parameters: <ctl> <map.tab> <met0> [ <met1> ... ]");
00048
00049   /* Read control parameters... */
00050   read_ctl(argv[1], argc, argv, &ctl);
00051   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00052
```

```
00053   /* Loop over files... */
00054   for (i = 3; i < argc; i++) {
00055
00056     /* Read meteorological data... */
00057     if (!(in = fopen(argv[i], "r")))
00058       continue;
00059     else
00060       fclose(in);
00061     read_met(&ctl, argv[i], met);
00062
00063     /* Find nearest pressure level... */
00064     for (ip2 = 0; ip2 < met->np; ip2++) {
00065       dz = fabs(Z(met->p[ip2]) - z);
00066       if (dz < dzmin) {
00067         dzmin = dz;
00068         ip = ip2;
00069       }
00070     }
00071
00072     /* Average data... */
00073     for (ix = 0; ix < met->nx; ix++)
00074       for (iy = 0; iy < met->ny; iy++) {
00075         timem[ix][iy] += met->time;
00076         tm[ix][iy] += met->t[ix][iy][ip];
00077         um[ix][iy] += met->u[ix][iy][ip];
00078         vm[ix][iy] += met->v[ix][iy][ip];
00079         wm[ix][iy] += met->w[ix][iy][ip];
00080         h2om[ix][iy] += met->h2o[ix][iy][ip];
00081         o3m[ix][iy] += met->o3[ix][iy][ip];
00082         psm[ix][iy] += met->ps[ix][iy];
00083         np[ix][iy]++;
00084       }
00085   }
00086
00087   /* Create output file... */
00088   printf("Write meteorological data file: %s\n", argv[2]);
00089   if (!(out = fopen(argv[2], "w")))
00090     ERRMSG("Cannot create file!");
00091
00092   /* Write header... */
00093   fprintf(out,
00094           "# $1  = time [s]\n"
00095           "# $2  = altitude [km]\n"
00096           "# $3  = longitude [deg]\n"
00097           "# $4  = latitude [deg]\n"
00098           "# $5  = pressure [hPa]\n"
00099           "# $6  = temperature [K]\n"
00100           "# $7  = zonal wind [m/s]\n"
00101           "# $8  = meridional wind [m/s]\n"
00102           "# $9  = vertical wind [hPa/s]\n"
00103           "# $10 = H2O volume mixing ratio [1]\n"
00104           "# $11 = O3 volume mixing ratio [1]\n"
00105           "# $12 = surface pressure [hPa]\n");
00106
00107   /* Write data... */
00108   for (iy = 0; iy < met->ny; iy++) {
00109     fprintf(out, "\n");
00110     for (ix = 0; ix < met->nx; ix++)
00111       if (met->lon[ix] >= 180)
00112         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00113                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00114                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00115                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00116                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00117                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00118                 psm[ix][iy] / np[ix][iy]);
00119     for (ix = 0; ix < met->nx; ix++)
00120       if (met->lon[ix] <= 180)
00121         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00122                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00123                 met->lon[ix], met->lat[iy], met->p[ip],
00124                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00125                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00126                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00127                 psm[ix][iy] / np[ix][iy]);
00128   }
00129
00130   /* Close file... */
00131   fclose(out);
00132
00133   /* Free... */
00134   free(met);
00135
00136   return EXIT_SUCCESS;
00137 }
```

5.19 met_prof.c File Reference

Extract vertical profile from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.19.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met_prof.c.

5.19.2 Function Documentation

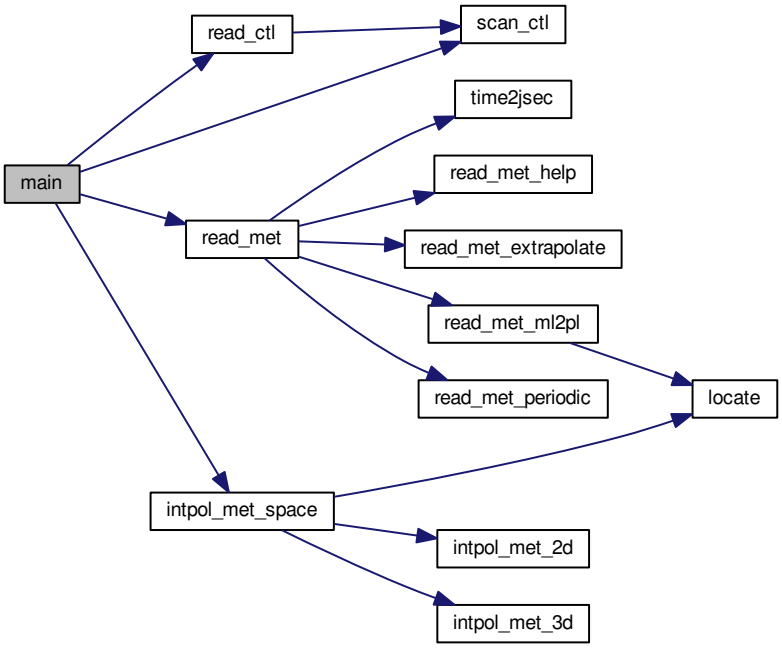
5.19.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 38 of file met_prof.c.

```
00040                {
00041
00042   ctl_t ctl;
00043
00044   met_t *met;
00045
00046   FILE *in, *out;
00047
00048   static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ],
00050     w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
00051
00052   static int i, iz, np[NZ];
00053
00054   /* Allocate... */
00055   ALLOC(met, met_t, 1);
00056
00057   /* Check arguments... */
00058   if (argc < 4)
00059     ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00060
00061   /* Read control parameters... */
00062   read_ctl(argv[1], argc, argv, &ctl);
00063   z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
00064   z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
00065   dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
00066   lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
00067   lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
00068   dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
00069   lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
00070   lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
00071   dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00072
00073   /* Loop over input files... */
00074   for (i = 3; i < argc; i++) {
00075
00076     /* Read meteorological data... */
00077     if (!(in = fopen(argv[i], "r")))
00078       continue;
00079     else
00080       fclose(in);
00081     read_met(&ctl, argv[i], met);
00082
00083     /* Average... */
00084     for (z = z0; z <= z1; z += dz) {
00085       iz = (int) ((z - z0) / dz);
00086       if (iz < 0 || iz > NZ)
00087         ERRMSG("Too many altitudes!");
```

```
00088       for (lon = lon0; lon <= lon1; lon += dlon)
00089         for (lat = lat0; lat <= lat1; lat += dlat) {
00090           intpol_met_space(met, P(z), lon, lat, &ps,
00091                            &t, &u, &v, &w, &h2o, &o3);
00092           if (gsl_finite(t) && gsl_finite(u)
00093               && gsl_finite(v) && gsl_finite(w)) {
00094             timem[iz] += met->time;
00095             lonm[iz] += lon;
00096             latm[iz] += lat;
00097             tm[iz] += t;
00098             um[iz] += u;
00099             vm[iz] += v;
00100            wm[iz] += w;
00101            h2om[iz] += h2o;
00102            o3m[iz] += o3;
00103            psm[iz] += ps;
00104            np[iz]++;
00105          }
00106       }
00107   }
00108 }
00109
00110 /* Normalize... */
00111 for (z = z0; z <= z1; z += dz) {
00112   iz = (int) ((z - z0) / dz);
00113   if (np[iz] > 0) {
00114     timem[iz] /= np[iz];
00115     lonm[iz] /= np[iz];
00116     latm[iz] /= np[iz];
00117     tm[iz] /= np[iz];
00118     um[iz] /= np[iz];
00119     vm[iz] /= np[iz];
00120     wm[iz] /= np[iz];
00121     h2om[iz] /= np[iz];
00122     o3m[iz] /= np[iz];
00123     psm[iz] /= np[iz];
00124   } else {
00125     timem[iz] = GSL_NAN;
00126     lonm[iz] = GSL_NAN;
00127     latm[iz] = GSL_NAN;
00128     tm[iz] = GSL_NAN;
00129     um[iz] = GSL_NAN;
00130     vm[iz] = GSL_NAN;
00131     wm[iz] = GSL_NAN;
00132     h2om[iz] = GSL_NAN;
00133     o3m[iz] = GSL_NAN;
00134     psm[iz] = GSL_NAN;
00135   }
00136 }
00137
00138 /* Create output file... */
00139 printf("Write meteorological data file: %s\n", argv[2]);
00140 if (!(out = fopen(argv[2], "w")))
00141   ERRMSG("Cannot create file!");
00142
00143 /* Write header... */
00144 fprintf(out,
00145         "# $1  = time [s]\n"
00146         "# $2  = altitude [km]\n"
00147         "# $3  = longitude [deg]\n"
00148         "# $4  = latitude [deg]\n"
00149         "# $5  = pressure [hPa]\n"
00150         "# $6  = temperature [K]\n"
00151         "# $7  = zonal wind [m/s]\n"
00152         "# $8  = meridional wind [m/s]\n"
00153         "# $9  = vertical wind [hPa/s]\n"
00154         "# $10 = H2O volume mixing ratio [1]\n"
00155         "# $11 = O3 volume mixing ratio [1]\n"
00156         "# $12 = surface pressure [hPa]\n\n");
00157
00158 /* Write data... */
00159 for (z = z0; z <= z1; z += dz) {
00160   iz = (int) ((z - z0) / dz);
00161   fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00162           timem[iz], z, lonm[iz], latm[iz], P(z), tm[iz],
00163           um[iz], vm[iz], wm[iz], h2om[iz], o3m[iz], psm[iz]);
00164 }
00165
00166 /* Close file... */
00167 fclose(out);
00168
00169 /* Free... */
00170 free(met);
00171
00172 return EXIT_SUCCESS;
00173 }
```

Here is the call graph for this function:



5.20 met_prof.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* ------------------------------------------------------------
00028    Dimensions...
00029    ------------------------------------------------------------ */
00030
00031 /* Maximum number of altitudes. */
00032 #define NZ 1000
00033
00034 /* ------------------------------------------------------------
00035    Main...
00036    ------------------------------------------------------------ */
00037
00038 int main(
00039   int argc,
00040   char *argv[]) {
00041
00042   ctl_t ctl;
00043
```

```
00044   met_t *met;
00045
00046   FILE *in, *out;
00047
00048   static double timem[NZ], z, z0, z1, dz, lon, lon0, lon1, dlon, lonm[NZ],
00049     lat, lat0, lat1, dlat, latm[NZ], t, tm[NZ], u, um[NZ], v, vm[NZ],
00050     w, wm[NZ], h2o, h2om[NZ], o3, o3m[NZ], ps, psm[NZ];
00051
00052   static int i, iz, np[NZ];
00053
00054   /* Allocate... */
00055   ALLOC(met, met_t, 1);
00056
00057   /* Check arguments... */
00058   if (argc < 4)
00059     ERRMSG("Give parameters: <ctl> <prof.tab> <met0> [ <met1> ... ]");
00060
00061   /* Read control parameters... */
00062   read_ctl(argv[1], argc, argv, &ctl);
00063   z0 = scan_ctl(argv[1], argc, argv, "Z0", -1, "0", NULL);
00064   z1 = scan_ctl(argv[1], argc, argv, "Z1", -1, "60", NULL);
00065   dz = scan_ctl(argv[1], argc, argv, "DZ", -1, "1", NULL);
00066   lon0 = scan_ctl(argv[1], argc, argv, "LON0", -1, "0", NULL);
00067   lon1 = scan_ctl(argv[1], argc, argv, "LON1", -1, "0", NULL);
00068   dlon = scan_ctl(argv[1], argc, argv, "DLON", -1, "1", NULL);
00069   lat0 = scan_ctl(argv[1], argc, argv, "LAT0", -1, "0", NULL);
00070   lat1 = scan_ctl(argv[1], argc, argv, "LAT1", -1, "0", NULL);
00071   dlat = scan_ctl(argv[1], argc, argv, "DLAT", -1, "1", NULL);
00072
00073   /* Loop over input files... */
00074   for (i = 3; i < argc; i++) {
00075
00076     /* Read meteorological data... */
00077     if (!(in = fopen(argv[i], "r")))
00078       continue;
00079     else
00080       fclose(in);
00081     read_met(&ctl, argv[i], met);
00082
00083     /* Average... */
00084     for (z = z0; z <= z1; z += dz) {
00085       iz = (int) ((z - z0) / dz);
00086       if (iz < 0 || iz > NZ)
00087         ERRMSG("Too many altitudes!");
00088       for (lon = lon0; lon <= lon1; lon += dlon)
00089         for (lat = lat0; lat <= lat1; lat += dlat) {
00090           intpol_met_space(met, P(z), lon, lat, &ps,
00091                            &t, &u, &v, &w, &h2o, &o3);
00092           if (gsl_finite(t) && gsl_finite(u)
00093               && gsl_finite(v) && gsl_finite(w)) {
00094             timem[iz] += met->time;
00095             lonm[iz] += lon;
00096             latm[iz] += lat;
00097             tm[iz] += t;
00098             um[iz] += u;
00099             vm[iz] += v;
00100             wm[iz] += w;
00101             h2om[iz] += h2o;
00102             o3m[iz] += o3;
00103             psm[iz] += ps;
00104             np[iz]++;
00105           }
00106         }
00107     }
00108   }
00109
00110   /* Normalize... */
00111   for (z = z0; z <= z1; z += dz) {
00112     iz = (int) ((z - z0) / dz);
00113     if (np[iz] > 0) {
00114       timem[iz] /= np[iz];
00115       lonm[iz] /= np[iz];
00116       latm[iz] /= np[iz];
00117       tm[iz] /= np[iz];
00118       um[iz] /= np[iz];
00119       vm[iz] /= np[iz];
00120       wm[iz] /= np[iz];
00121       h2om[iz] /= np[iz];
00122       o3m[iz] /= np[iz];
00123       psm[iz] /= np[iz];
00124     } else {
00125       timem[iz] = GSL_NAN;
00126       lonm[iz] = GSL_NAN;
00127       latm[iz] = GSL_NAN;
00128       tm[iz] = GSL_NAN;
00129       um[iz] = GSL_NAN;
00130       vm[iz] = GSL_NAN;
```

```
00131       wm[iz] = GSL_NAN;
00132       h2om[iz] = GSL_NAN;
00133       o3m[iz] = GSL_NAN;
00134       psm[iz] = GSL_NAN;
00135     }
00136   }
00137
00138   /* Create output file... */
00139   printf("Write meteorological data file: %s\n", argv[2]);
00140   if (!(out = fopen(argv[2], "w")))
00141     ERRMSG("Cannot create file!");
00142
00143   /* Write header... */
00144   fprintf(out,
00145           "# $1  = time [s]\n"
00146           "# $2  = altitude [km]\n"
00147           "# $3  = longitude [deg]\n"
00148           "# $4  = latitude [deg]\n"
00149           "# $5  = pressure [hPa]\n"
00150           "# $6  = temperature [K]\n"
00151           "# $7  = zonal wind [m/s]\n"
00152           "# $8  = meridional wind [m/s]\n"
00153           "# $9  = vertical wind [hPa/s]\n"
00154           "# $10 = H2O volume mixing ratio [1]\n"
00155           "# $11 = O3 volume mixing ratio [1]\n"
00156           "# $12 = surface pressure [hPa]\n\n");
00157
00158   /* Write data... */
00159   for (z = z0; z <= z1; z += dz) {
00160     iz = (int) ((z - z0) / dz);
00161     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00162             timem[iz], z, lonm[iz], latm[iz], P(z), tm[iz],
00163             um[iz], vm[iz], wm[iz], h2om[iz], o3m[iz], psm[iz]);
00164   }
00165
00166   /* Close file... */
00167   fclose(out);
00168
00169   /* Free... */
00170   free(met);
00171
00172   return EXIT_SUCCESS;
00173 }
```

5.21 met_sample.c File Reference

Sample meteorological data at given geolocations.

Functions

- int main (int argc, char ∗argv[ ])

5.21.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met_sample.c.

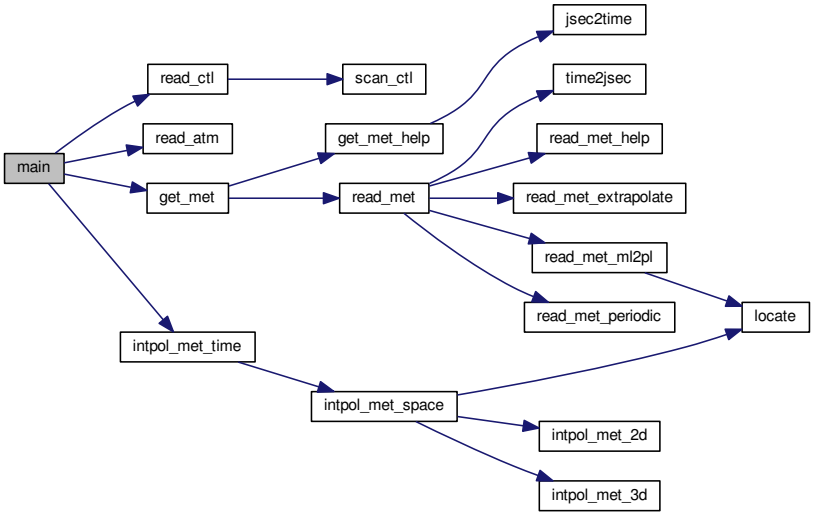
5.21.2 Function Documentation

5.21.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 31 of file met_sample.c.

```
00033                {
00034
00035   ctl_t ctl;
00036
00037   atm_t *atm;
00038
00039   met_t *met0, *met1;
00040
00041   FILE *out;
00042
00043   double t, u, v, w, h2o, o3;
00044
00045   int ip;
00046
00047   /* Check arguments... */
00048   if (argc < 4)
00049     ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00050
00051   /* Allocate... */
00052   ALLOC(atm, atm_t, 1);
00053   ALLOC(met0, met_t, 1);
00054   ALLOC(met1, met_t, 1);
00055
00056   /* Read control parameters... */
00057   read_ctl(argv[1], argc, argv, &ctl);
00058
00059   /* Read atmospheric data... */
00060   read_atm(argv[3], &ctl, atm);
00061
00062   /* Create output file... */
00063   printf("Write meteorological data file: %s\n", argv[4]);
00064   if (!(out = fopen(argv[4], "w")))
00065     ERRMSG("Cannot create file!");
00066
00067   /* Write header... */
00068   fprintf(out,
00069           "# $1  = time [s]\n"
00070           "# $2  = altitude [km]\n"
00071           "# $3  = longitude [deg]\n"
00072           "# $4  = latitude [deg]\n"
00073           "# $5  = pressure [hPa]\n"
00074           "# $6  = temperature [K]\n"
00075           "# $7  = zonal wind [m/s]\n"
00076           "# $8  = meridional wind [m/s]\n"
00077           "# $9  = vertical wind [hPa/s]\n"
00078           "# $10 = H2O volume mixing ratio [1]\n"
00079           "# $11 = O3 volume mixing ratio [1]\n\n");
00080
00081   /* Loop over air parcels... */
00082   for (ip = 0; ip < atm->np; ip++) {
00083
00084     /* Get meteorological data... */
00085     get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00086
00087     /* Interpolate meteorological data... */
00088     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00089                     atm->lat[ip], NULL, &t, &u, &v, &w, &h2o, &o3);
00090
00091     /* Write data... */
00092     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n",
00093             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00094             atm->p[ip], t, u, v, w, h2o, o3);
00095   }
00096
00097   /* Close file... */
00098   fclose(out);
00099
00100   /* Free... */
00101   free(atm);
00102   free(met0);
00103   free(met1);
00104
00105   return EXIT_SUCCESS;
00106 }
```

Here is the call graph for this function:



5.22 met_sample.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* ------------------------------------------------------------
00028    Main...
00029    ------------------------------------------------------------ */
00030
00031 int main(
00032   int argc,
00033   char *argv[]) {
00034
00035   ctl_t ctl;
00036
00037   atm_t *atm;
00038
00039   met_t *met0, *met1;
00040
00041   FILE *out;
00042
00043   double t, u, v, w, h2o, o3;
00044
00045   int ip;
00046
00047   /* Check arguments... */
00048   if (argc < 4)
00049     ERRMSG("Give parameters: <ctl> <metbase> <atm_in> <sample.tab>");
00050
00051   /* Allocate... */
```

```
00052   ALLOC(atm, atm_t, 1);
00053   ALLOC(met0, met_t, 1);
00054   ALLOC(met1, met_t, 1);
00055
00056   /* Read control parameters... */
00057   read_ctl(argv[1], argc, argv, &ctl);
00058
00059   /* Read atmospheric data... */
00060   read_atm(argv[3], &ctl, atm);
00061
00062   /* Create output file... */
00063   printf("Write meteorological data file: %s\n", argv[4]);
00064   if (!(out = fopen(argv[4], "w")))
00065     ERRMSG("Cannot create file!");
00066
00067   /* Write header... */
00068   fprintf(out,
00069           "# $1  = time [s]\n"
00070           "# $2  = altitude [km]\n"
00071           "# $3  = longitude [deg]\n"
00072           "# $4  = latitude [deg]\n"
00073           "# $5  = pressure [hPa]\n"
00074           "# $6  = temperature [K]\n"
00075           "# $7  = zonal wind [m/s]\n"
00076           "# $8  = meridional wind [m/s]\n"
00077           "# $9  = vertical wind [hPa/s]\n"
00078           "# $10 = H2O volume mixing ratio [1]\n"
00079           "# $11 = O3 volume mixing ratio [1]\n\n");
00080
00081   /* Loop over air parcels... */
00082   for (ip = 0; ip < atm->np; ip++) {
00083
00084     /* Get meteorological data... */
00085     get_met(&ctl, argv[2], atm->time[ip], met0, met1);
00086
00087     /* Interpolate meteorological data... */
00088     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00089                     atm->lat[ip], NULL, &t, &u, &v, &w, &h2o, &o3);
00090
00091     /* Write data... */
00092     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g\n",
00093             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip],
00094             atm->p[ip], t, u, v, w, h2o, o3);
00095   }
00096
00097   /* Close file... */
00098   fclose(out);
00099
00100   /* Free... */
00101   free(atm);
00102   free(met0);
00103   free(met1);
00104
00105   return EXIT_SUCCESS;
00106 }
```

5.23 met_zm.c File Reference

Extract zonal mean from meteorological data.

Functions

- int main (int argc, char *argv[])

5.23.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met_zm.c.

5.23.2 Function Documentation

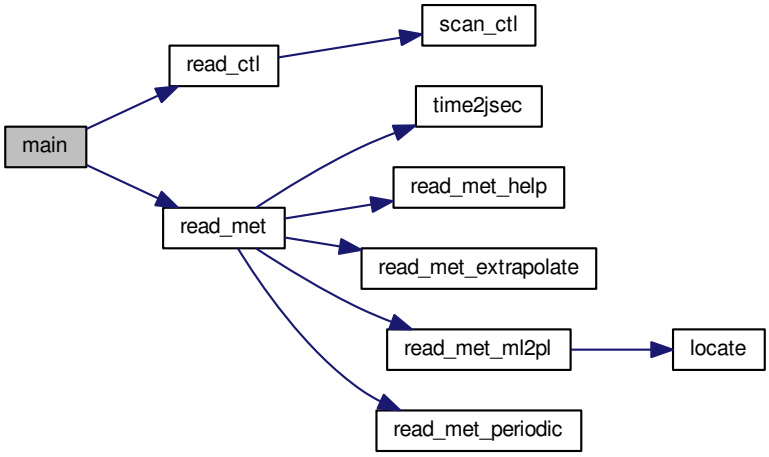
5.23.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file met_zm.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *in, *out;
00036
00037   static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
00038     vm[EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],
00039     psm2[EP][EY], tm2[EP][EY], um2[EP][EY], vm2[EP][EY], vhm2[EP][EY],
00040     wm2[EP][EY], h2om2[EP][EY], o3m2[EP][EY];
00041
00042   static int i, ip, ix, iy, np[EP][EY];
00043
00044   /* Allocate... */
00045   ALLOC(met, met_t, 1);
00046
00047   /* Check arguments... */
00048   if (argc < 4)
00049     ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00050
00051   /* Read control parameters... */
00052   read_ctl(argv[1], argc, argv, &ctl);
00053
00054   /* Loop over files... */
00055   for (i = 3; i < argc; i++) {
00056
00057     /* Read meteorological data... */
00058     if (!(in = fopen(argv[i], "r")))
00059       continue;
00060     else
00061       fclose(in);
00062     read_met(&ctl, argv[i], met);
00063
00064     /* Average data... */
00065     for (ix = 0; ix < met->nx; ix++)
00066       for (iy = 0; iy < met->ny; iy++)
00067         for (ip = 0; ip < met->np; ip++) {
00068           timem[ip][iy] += met->time;
00069           tm[ip][iy] += met->t[ix][iy][ip];
00070           um[ip][iy] += met->u[ix][iy][ip];
00071           vm[ip][iy] += met->v[ix][iy][ip];
00072           vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00073                               + gsl_pow_2(met->v[ix][iy][ip]));
00074           wm[ip][iy] += met->w[ix][iy][ip];
00075           h2om[ip][iy] += met->h2o[ix][iy][ip];
00076           o3m[ip][iy] += met->o3[ix][iy][ip];
00077           psm[ip][iy] += met->ps[ix][iy];
00078           tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00079           um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00080           vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
00081           vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00082             + gsl_pow_2(met->v[ix][iy][ip]);
00083           wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00084           h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00085           o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
00086           psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00087           np[ip][iy]++;
00088         }
00089   }
00090
00091   /* Create output file... */
00092   printf("Write meteorological data file: %s\n", argv[2]);
00093   if (!(out = fopen(argv[2], "w")))
00094     ERRMSG("Cannot create file!");
00095
00096   /* Write header... */
00097   fprintf(out,
00098           "# $1  = time [s]\n"
00099           "# $2  = altitude [km]\n"
00100           "# $3  = latitude [deg]\n"
00101           "# $4  = temperature mean [K]\n"
00102           "# $5  = temperature standard deviation [K]\n"
00103           "# $6  = zonal wind mean [m/s]\n"
00104           "# $7  = zonal wind standard deviation [m/s]\n"
```

```
00105           "# $8  = meridional wind mean [m/s]\n"
00106           "# $9  = meridional wind standard deviation [m/s]\n"
00107           "# $10 = horizontal wind mean [m/s]\n"
00108           "# $11 = horizontal wind standard deviation [m/s]\n"
00109           "# $12 = vertical wind mean [hPa/s]\n"
00110           "# $13 = vertical wind standard deviation [hPa/s]\n"
00111           "# $14 = H2O vmr mean [1]\n"
00112           "# $15 = H2O vmr standard deviation [1]\n"
00113           "# $16 = O3 vmr mean [1]\n"
00114           "# $17 = O3 vmr standard deviation [1]\n"
00115           "# $18 = surface pressure mean [hPa]\n"
00116           "# $19 = surface pressure standard deviation [hPa]\n");
00117
00118   /* Write data... */
00119   for (iy = 0; iy < met->ny; iy++) {
00120     fprintf(out, "\n");
00121     for (ip = 0; ip < met->np; ip++)
00122       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g"
00123               " %g %g %g %g %g %g %g %g\n",
00124               timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
00125               tm[ip][iy] / np[ip][iy],
00126               sqrt(tm2[ip][iy] / np[ip][iy] -
00127                    gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00128               um[ip][iy] / np[ip][iy],
00129               sqrt(um2[ip][iy] / np[ip][iy] -
00130                    gsl_pow_2(um[ip][iy] / np[ip][iy])),
00131               vm[ip][iy] / np[ip][iy],
00132               sqrt(vm2[ip][iy] / np[ip][iy] -
00133                    gsl_pow_2(vm[ip][iy] / np[ip][iy])),
00134               vhm[ip][iy] / np[ip][iy],
00135               sqrt(vhm2[ip][iy] / np[ip][iy] -
00136                    gsl_pow_2(vhm[ip][iy] / np[ip][iy])),
00137               wm[ip][iy] / np[ip][iy],
00138               sqrt(wm2[ip][iy] / np[ip][iy] -
00139                    gsl_pow_2(wm[ip][iy] / np[ip][iy])),
00140               h2om[ip][iy] / np[ip][iy],
00141               sqrt(h2om2[ip][iy] / np[ip][iy] -
00142                    gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
00143               o3m[ip][iy] / np[ip][iy],
00144               sqrt(o3m2[ip][iy] / np[ip][iy] -
00145                    gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
00146               psm[ip][iy] / np[ip][iy],
00147               sqrt(psm2[ip][iy] / np[ip][iy] -
00148                    gsl_pow_2(psm[ip][iy] / np[ip][iy])));
00149   }
00150
00151   /* Close file... */
00152   fclose(out);
00153
00154   /* Free... */
00155   free(met);
00156
00157   return EXIT_SUCCESS;
00158 }
```

Here is the call graph for this function:



5.24 met_zm.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *in, *out;
00036
00037   static double timem[EP][EY], psm[EP][EY], tm[EP][EY], um[EP][EY],
00038     vm[EP][EY], vhm[EP][EY], wm[EP][EY], h2om[EP][EY], o3m[EP][EY],
00039     psm2[EP][EY], tm2[EP][EY], um2[EP][EY], vm2[EP][EY], vhm2[EP][EY],
00040     wm2[EP][EY], h2om2[EP][EY], o3m2[EP][EY];
00041
00042   static int i, ip, ix, iy, np[EP][EY];
00043
00044   /* Allocate... */
00045   ALLOC(met, met_t, 1);
00046
00047   /* Check arguments... */
00048   if (argc < 4)
00049     ERRMSG("Give parameters: <ctl> <zm.tab> <met0> [ <met1> ... ]");
00050
00051   /* Read control parameters... */
00052   read_ctl(argv[1], argc, argv, &ctl);
```

```
00053
00054   /* Loop over files... */
00055   for (i = 3; i < argc; i++) {
00056
00057     /* Read meteorological data... */
00058     if (!(in = fopen(argv[i], "r")))
00059       continue;
00060     else
00061       fclose(in);
00062     read_met(&ctl, argv[i], met);
00063
00064     /* Average data... */
00065     for (ix = 0; ix < met->nx; ix++)
00066       for (iy = 0; iy < met->ny; iy++)
00067         for (ip = 0; ip < met->np; ip++) {
00068           timem[ip][iy] += met->time;
00069           tm[ip][iy] += met->t[ix][iy][ip];
00070           um[ip][iy] += met->u[ix][iy][ip];
00071           vm[ip][iy] += met->v[ix][iy][ip];
00072           vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00073                               + gsl_pow_2(met->v[ix][iy][ip]));
00074           wm[ip][iy] += met->w[ix][iy][ip];
00075           h2om[ip][iy] += met->h2o[ix][iy][ip];
00076           o3m[ip][iy] += met->o3[ix][iy][ip];
00077           psm[ip][iy] += met->ps[ix][iy];
00078           tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00079           um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00080           vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
00081           vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00082             + gsl_pow_2(met->v[ix][iy][ip]);
00083           wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00084           h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00085           o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
00086           psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00087           np[ip][iy]++;
00088         }
00089   }
00090
00091   /* Create output file... */
00092   printf("Write meteorological data file: %s\n", argv[2]);
00093   if (!(out = fopen(argv[2], "w")))
00094     ERRMSG("Cannot create file!");
00095
00096   /* Write header... */
00097   fprintf(out,
00098           "# $1  = time [s]\n"
00099           "# $2  = altitude [km]\n"
00100           "# $3  = latitude [deg]\n"
00101           "# $4  = temperature mean [K]\n"
00102           "# $5  = temperature standard deviation [K]\n"
00103           "# $6  = zonal wind mean [m/s]\n"
00104           "# $7  = zonal wind standard deviation [m/s]\n"
00105           "# $8  = meridional wind mean [m/s]\n"
00106           "# $9  = meridional wind standard deviation [m/s]\n"
00107           "# $10 = horizontal wind mean [m/s]\n"
00108           "# $11 = horizontal wind standard deviation [m/s]\n"
00109           "# $12 = vertical wind mean [hPa/s]\n"
00110           "# $13 = vertical wind standard deviation [hPa/s]\n"
00111           "# $14 = H2O vmr mean [1]\n"
00112           "# $15 = H2O vmr standard deviation [1]\n"
00113           "# $16 = O3 vmr mean [1]\n"
00114           "# $17 = O3 vmr standard deviation [1]\n"
00115           "# $18 = surface pressure mean [hPa]\n"
00116           "# $19 = surface pressure standard deviation [hPa]\n");
00117
00118   /* Write data... */
00119   for (iy = 0; iy < met->ny; iy++) {
00120     fprintf(out, "\n");
00121     for (ip = 0; ip < met->np; ip++)
00122       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g"
00123               " %g %g %g %g %g %g %g %g\n",
00124               timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
00125               tm[ip][iy] / np[ip][iy],
00126               sqrt(tm2[ip][iy] / np[ip][iy] -
00127                    gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00128               um[ip][iy] / np[ip][iy],
00129               sqrt(um2[ip][iy] / np[ip][iy] -
00130                    gsl_pow_2(um[ip][iy] / np[ip][iy])),
00131               vm[ip][iy] / np[ip][iy],
00132               sqrt(vm2[ip][iy] / np[ip][iy] -
00133                    gsl_pow_2(vm[ip][iy] / np[ip][iy])),
00134               vhm[ip][iy] / np[ip][iy],
00135               sqrt(vhm2[ip][iy] / np[ip][iy] -
00136                    gsl_pow_2(vhm[ip][iy] / np[ip][iy])),
00137               wm[ip][iy] / np[ip][iy],
00138               sqrt(wm2[ip][iy] / np[ip][iy] -
00139                    gsl_pow_2(wm[ip][iy] / np[ip][iy])),
```

```
00140               h2om[ip][iy] / np[ip][iy],
00141               sqrt(h2om2[ip][iy] / np[ip][iy] -
00142                    gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
00143               o3m[ip][iy] / np[ip][iy],
00144               sqrt(o3m2[ip][iy] / np[ip][iy] -
00145                    gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
00146               psm[ip][iy] / np[ip][iy],
00147               sqrt(psm2[ip][iy] / np[ip][iy] -
00148                    gsl_pow_2(psm[ip][iy] / np[ip][iy])));
00149   }
00150
00151   /* Close file... */
00152   fclose(out);
00153
00154   /* Free... */
00155   free(met);
00156
00157   return EXIT_SUCCESS;
00158 }
```

5.25 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

Functions

- int main (int argc, char ∗argv[ ])

5.25.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

5.25.2 Function Documentation

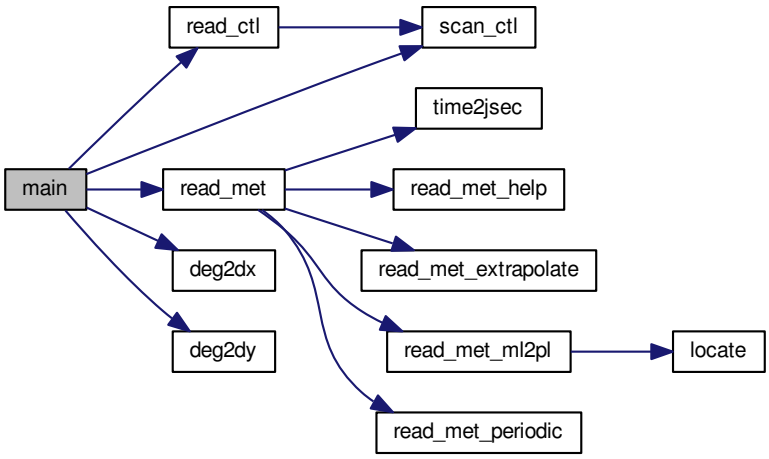
5.25.2.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 8 of file smago.c.

```
00010                {
00011
00012   ctl_t ctl;
00013
00014   met_t *met;
00015
00016   FILE *out;
00017
00018   static double dz, dzmin = 1e10, z, t, s, ls2, k[EX][EY], c = 0.15;
00019
00020   static int ip, ip2, ix, iy;
00021
00022   /* Allocate... */
00023   ALLOC(met, met_t, 1);
00024
00025   /* Check arguments... */
00026   if (argc < 4)
00027     ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00028
00029   /* Read control parameters... */
00030   read_ctl(argv[1], argc, argv, &ctl);
00031   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00032
00033   /* Read meteorological data... */
00034   read_met(&ctl, argv[3], met);
```

```
00035
00036   /* Find nearest pressure level... */
00037   for (ip2 = 0; ip2 < met->np; ip2++) {
00038     dz = fabs(Z(met->p[ip2]) - z);
00039     if (dz < dzmin) {
00040       dzmin = dz;
00041       ip = ip2;
00042     }
00043   }
00044
00045   /* Write info... */
00046   printf("Analyze %g hPa...\n", met->p[ip]);
00047
00048   /* Calculate horizontal diffusion coefficients... */
00049   for (ix = 1; ix < met->nx - 1; ix++)
00050     for (iy = 1; iy < met->ny - 1; iy++) {
00051       t = 0.5 * ((met->u[ix + 1][iy][ip] - met->u[ix - 1][iy][ip])
00052                  / (1000. *
00053                     deg2dx(met->lon[ix + 1] - met->lon[ix - 1], met->
      lat[iy]))
00054                  - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
00055                  / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1])));
00056       s = 0.5 * ((met->u[ix][iy + 1][ip] - met->u[ix][iy - 1][ip])
00057                  / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]))
00058                  + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00059                  / (1000. *
00060                     deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00061                            met->lat[iy])));
00062       ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
00063       if (fabs(met->lat[iy]) > 80)
00064         ls2 *= (90. - fabs(met->lat[iy])) / 10.;
00065       k[ix][iy] = ls2 * sqrt(2.0 * (gsl_pow_2(t) + gsl_pow_2(s)));
00066     }
00067
00068   /* Create output file... */
00069   printf("Write data file: %s\n", argv[2]);
00070   if (!(out = fopen(argv[2], "w")))
00071     ERRMSG("Cannot create file!");
00072
00073   /* Write header... */
00074   fprintf(out,
00075           "# $1 = longitude [deg]\n"
00076           "# $2 = latitude [deg]\n"
00077           "# $3 = zonal wind [m/s]\n"
00078           "# $4 = meridional wind [m/s]\n"
00079           "# $5 = horizontal diffusivity [m^2/s]\n");
00080
00081   /* Write data... */
00082   for (iy = 0; iy < met->ny; iy++) {
00083     fprintf(out, "\n");
00084     for (ix = 0; ix < met->nx; ix++)
00085       if (met->lon[ix] >= 180)
00086         fprintf(out, "%g %g %g %g %g\n",
00087                 met->lon[ix] - 360.0, met->lat[iy],
00088                 met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00089     for (ix = 0; ix < met->nx; ix++)
00090       if (met->lon[ix] <= 180)
00091         fprintf(out, "%g %g %g %g %g\n",
00092                 met->lon[ix], met->lat[iy],
00093                 met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00094   }
00095
00096   /* Close file... */
00097   fclose(out);
00098
00099   /* Free... */
00100   free(met);
00101
00102   return EXIT_SUCCESS;
00103 }
```

Here is the call graph for this function:



5.26 smago.c

```
00001
00006 #include "libtrac.h"
00007
00008 int main(
00009   int argc,
00010   char *argv[]) {
00011
00012   ctl_t ctl;
00013
00014   met_t *met;
00015
00016   FILE *out;
00017
00018   static double dz, dzmin = 1e10, z, t, s, ls2, k[EX][EY], c = 0.15;
00019
00020   static int ip, ip2, ix, iy;
00021
00022   /* Allocate... */
00023   ALLOC(met, met_t, 1);
00024
00025   /* Check arguments... */
00026   if (argc < 4)
00027     ERRMSG("Give parameters: <ctl> <map.tab> <met>");
00028
00029   /* Read control parameters... */
00030   read_ctl(argv[1], argc, argv, &ctl);
00031   z = scan_ctl(argv[1], argc, argv, "Z", -1, "", NULL);
00032
00033   /* Read meteorological data... */
00034   read_met(&ctl, argv[3], met);
00035
00036   /* Find nearest pressure level... */
00037   for (ip2 = 0; ip2 < met->np; ip2++) {
00038     dz = fabs(Z(met->p[ip2]) - z);
00039     if (dz < dzmin) {
00040       dzmin = dz;
00041       ip = ip2;
00042     }
00043   }
00044
00045   /* Write info... */
00046   printf("Analyze %g hPa...\n", met->p[ip]);
00047
00048   /* Calculate horizontal diffusion coefficients... */
00049   for (ix = 1; ix < met->nx - 1; ix++)
00050     for (iy = 1; iy < met->ny - 1; iy++) {
00051       t = 0.5 * ((met->u[ix + 1][iy][ip] - met->u[ix - 1][iy][ip])
```

```
00052                   / (1000. *
00053                      deg2dx(met->lon[ix + 1] - met->lon[ix - 1], met->
    lat[iy]))
00054                   - (met->v[ix][iy + 1][ip] - met->v[ix][iy - 1][ip])
00055                   / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1])));
00056       s = 0.5 * ((met->u[ix][iy + 1][ip] - met->u[ix][iy - 1][ip])
00057                  / (1000. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]))
00058                  + (met->v[ix + 1][iy][ip] - met->v[ix - 1][iy][ip])
00059                  / (1000. *
00060                     deg2dx(met->lon[ix + 1] - met->lon[ix - 1],
00061                            met->lat[iy])));
00062       ls2 = gsl_pow_2(c * 500. * deg2dy(met->lat[iy + 1] - met->lat[iy - 1]));
00063       if (fabs(met->lat[iy]) > 80)
00064         ls2 *= (90. - fabs(met->lat[iy])) / 10.;
00065       k[ix][iy] = ls2 * sqrt(2.0 * (gsl_pow_2(t) + gsl_pow_2(s)));
00066     }
00067
00068   /* Create output file... */
00069   printf("Write data file: %s\n", argv[2]);
00070   if (!(out = fopen(argv[2], "w")))
00071     ERRMSG("Cannot create file!");
00072
00073   /* Write header... */
00074   fprintf(out,
00075           "# $1 = longitude [deg]\n"
00076           "# $2 = latitude [deg]\n"
00077           "# $3 = zonal wind [m/s]\n"
00078           "# $4 = meridional wind [m/s]\n"
00079           "# $5 = horizontal diffusivity [m^2/s]\n");
00080
00081   /* Write data... */
00082   for (iy = 0; iy < met->ny; iy++) {
00083     fprintf(out, "\n");
00084     for (ix = 0; ix < met->nx; ix++)
00085       if (met->lon[ix] >= 180)
00086         fprintf(out, "%g %g %g %g %g\n",
00087                 met->lon[ix] - 360.0, met->lat[iy],
00088                 met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00089     for (ix = 0; ix < met->nx; ix++)
00090       if (met->lon[ix] <= 180)
00091         fprintf(out, "%g %g %g %g %g\n",
00092                 met->lon[ix], met->lat[iy],
00093                 met->u[ix][iy][ip], met->v[ix][iy][ip], k[ix][iy]);
00094   }
00095
00096   /* Close file... */
00097   fclose(out);
00098
00099   /* Free... */
00100   free(met);
00101
00102   return EXIT_SUCCESS;
00103 }
```

5.27 split.c File Reference

Split air parcels into a larger number of parcels.

Functions

- int main (int argc, char *argv[])

5.27.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

5.27.2 Function Documentation

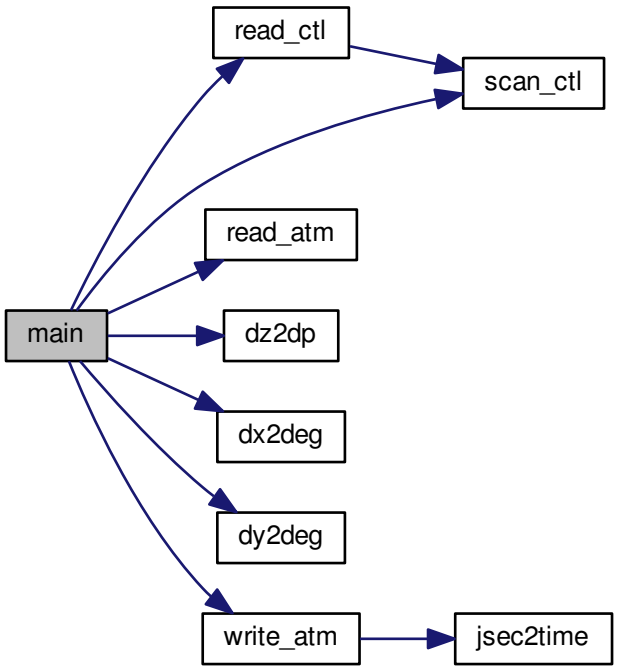
5.27.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file split.c.

```
00029                {
00030
00031   atm_t *atm, *atm2;
00032
00033   ctl_t ctl;
00034
00035   gsl_rng *rng;
00036
00037   double m, mtot = 0, dt, dx, dz, mmax = 0,
00038     t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00039
00040   int i, ip, iq, n;
00041
00042   /* Allocate... */
00043   ALLOC(atm, atm_t, 1);
00044   ALLOC(atm2, atm_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052   n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00053   m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00054   dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00055   t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00056   t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00057   dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00058   z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00059   z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00060   dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00061   lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00062   lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00063   lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00064   lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00065
00066   /* Init random number generator... */
00067   gsl_rng_env_setup();
00068   rng = gsl_rng_alloc(gsl_rng_default);
00069
00070   /* Read atmospheric data... */
00071   read_atm(argv[2], &ctl, atm);
00072
00073   /* Get total and maximum mass... */
00074   if (ctl.qnt_m >= 0)
00075     for (ip = 0; ip < atm->np; ip++) {
00076       mtot += atm->q[ctl.qnt_m][ip];
00077       mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078     }
00079   if (m > 0)
00080     mtot = m;
00081
00082   /* Loop over air parcels... */
00083   for (i = 0; i < n; i++) {
00084
00085     /* Select air parcel... */
00086     if (ctl.qnt_m >= 0)
00087       do {
00088         ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00089       } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
00090     else
00091       ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00092
00093     /* Set time... */
00094     if (t1 > t0)
00095       atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00096     else
00097       atm2->time[atm2->np] = atm->time[ip]
00098         + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
00100    /* Set vertical position... */
00101    if (z1 > z0)
00102      atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00103    else
00104      atm2->p[atm2->np] = atm->p[ip]
```

```
00105         + dz2dp(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00106
00107     /* Set horizontal position... */
00108     if (lon1 > lon0 && lat1 > lat0) {
00109       atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00110       atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00111     } else {
00112       atm2->lon[atm2->np] = atm->lon[ip]
00113         + gsl_ran_gaussian_ziggurat(rng, dx2deg(dx, atm->lat[ip]) / 2.3548);
00114       atm2->lat[atm2->np] = atm->lat[ip]
00115         + gsl_ran_gaussian_ziggurat(rng, dy2deg(dx) / 2.3548);
00116     }
00117
00118     /* Copy quantities... */
00119     for (iq = 0; iq < ctl.nq; iq++)
00120       atm2->q[iq][atm2->np] = atm->q[iq][ip];
00121
00122     /* Adjust mass... */
00123     if (ctl.qnt_m >= 0)
00124       atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
00126     /* Increment particle counter... */
00127     if ((++atm2->np) >= NP)
00128       ERRMSG("Too many air parcels!");
00129   }
00130
00131   /* Save data and close file... */
00132   write_atm(argv[3], &ctl, atm2, atm->time[0]);
00133
00134   /* Free... */
00135   free(atm);
00136   free(atm2);
00137
00138   return EXIT_SUCCESS;
00139 }
```

Here is the call graph for this function:



5.28 split.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   atm_t *atm, *atm2;
00032
00033   ctl_t ctl;
00034
00035   gsl_rng *rng;
00036
00037   double m, mtot = 0, dt, dx, dz, mmax = 0,
00038     t0, t1, z0, z1, lon0, lon1, lat0, lat1;
00039
00040   int i, ip, iq, n;
00041
00042   /* Allocate... */
00043   ALLOC(atm, atm_t, 1);
00044   ALLOC(atm2, atm_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <atm_in> <atm_out>");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052   n = (int) scan_ctl(argv[1], argc, argv, "SPLIT_N", -1, "", NULL);
00053   m = scan_ctl(argv[1], argc, argv, "SPLIT_M", -1, "-999", NULL);
00054   dt = scan_ctl(argv[1], argc, argv, "SPLIT_DT", -1, "0", NULL);
00055   t0 = scan_ctl(argv[1], argc, argv, "SPLIT_T0", -1, "0", NULL);
00056   t1 = scan_ctl(argv[1], argc, argv, "SPLIT_T1", -1, "0", NULL);
00057   dz = scan_ctl(argv[1], argc, argv, "SPLIT_DZ", -1, "0", NULL);
00058   z0 = scan_ctl(argv[1], argc, argv, "SPLIT_Z0", -1, "0", NULL);
00059   z1 = scan_ctl(argv[1], argc, argv, "SPLIT_Z1", -1, "0", NULL);
00060   dx = scan_ctl(argv[1], argc, argv, "SPLIT_DX", -1, "0", NULL);
00061   lon0 = scan_ctl(argv[1], argc, argv, "SPLIT_LON0", -1, "0", NULL);
00062   lon1 = scan_ctl(argv[1], argc, argv, "SPLIT_LON1", -1, "0", NULL);
00063   lat0 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT0", -1, "0", NULL);
00064   lat1 = scan_ctl(argv[1], argc, argv, "SPLIT_LAT1", -1, "0", NULL);
00065
00066   /* Init random number generator... */
00067   gsl_rng_env_setup();
00068   rng = gsl_rng_alloc(gsl_rng_default);
00069
00070   /* Read atmospheric data... */
00071   read_atm(argv[2], &ctl, atm);
00072
00073   /* Get total and maximum mass... */
00074   if (ctl.qnt_m >= 0)
00075     for (ip = 0; ip < atm->np; ip++) {
00076       mtot += atm->q[ctl.qnt_m][ip];
00077       mmax = GSL_MAX(mmax, atm->q[ctl.qnt_m][ip]);
00078     }
00079   if (m > 0)
00080     mtot = m;
00081
00082   /* Loop over air parcels... */
00083   for (i = 0; i < n; i++) {
00084
00085     /* Select air parcel... */
00086     if (ctl.qnt_m >= 0)
00087       do {
00088         ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00089       } while (gsl_rng_uniform(rng) > atm->q[ctl.qnt_m][ip] / mmax);
```

```
00090     else
00091       ip = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00092
00093     /* Set time... */
00094     if (t1 > t0)
00095       atm2->time[atm2->np] = t0 + (t1 - t0) * gsl_rng_uniform_pos(rng);
00096     else
00097       atm2->time[atm2->np] = atm->time[ip]
00098         + gsl_ran_gaussian_ziggurat(rng, dt / 2.3548);
00099
00100    /* Set vertical position... */
00101    if (z1 > z0)
00102      atm2->p[atm2->np] = P(z0 + (z1 - z0) * gsl_rng_uniform_pos(rng));
00103    else
00104      atm2->p[atm2->np] = atm->p[ip]
00105        + dz2dp(gsl_ran_gaussian_ziggurat(rng, dz / 2.3548), atm->p[ip]);
00106
00107    /* Set horizontal position... */
00108    if (lon1 > lon0 && lat1 > lat0) {
00109      atm2->lon[atm2->np] = lon0 + (lon1 - lon0) * gsl_rng_uniform_pos(rng);
00110      atm2->lat[atm2->np] = lat0 + (lat1 - lat0) * gsl_rng_uniform_pos(rng);
00111    } else {
00112      atm2->lon[atm2->np] = atm->lon[ip]
00113        + gsl_ran_gaussian_ziggurat(rng, dx2deg(dx, atm->lat[ip]) / 2.3548);
00114      atm2->lat[atm2->np] = atm->lat[ip]
00115        + gsl_ran_gaussian_ziggurat(rng, dy2deg(dx) / 2.3548);
00116    }
00117
00118    /* Copy quantities... */
00119    for (iq = 0; iq < ctl.nq; iq++)
00120      atm2->q[iq][atm2->np] = atm->q[iq][ip];
00121
00122    /* Adjust mass... */
00123    if (ctl.qnt_m >= 0)
00124      atm2->q[ctl.qnt_m][atm2->np] = mtot / n;
00125
00126    /* Increment particle counter... */
00127    if ((++atm2->np) >= NP)
00128      ERRMSG("Too many air parcels!");
00129  }
00130
00131  /* Save data and close file... */
00132  write_atm(argv[3], &ctl, atm2, atm->time[0]);
00133
00134  /* Free... */
00135  free(atm);
00136  free(atm2);
00137
00138  return EXIT_SUCCESS;
00139 }
```

5.29 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

- int main (int argc, char *argv[])

5.29.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

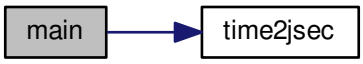
5.29.2 Function Documentation

5.29.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file time2jsec.c.

```
00029                {
00030
00031   double jsec, remain;
00032
00033   int day, hour, min, mon, sec, year;
00034
00035   /* Check arguments... */
00036   if (argc < 8)
00037     ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00038
00039   /* Read arguments... */
00040   year = atoi(argv[1]);
00041   mon = atoi(argv[2]);
00042   day = atoi(argv[3]);
00043   hour = atoi(argv[4]);
00044   min = atoi(argv[5]);
00045   sec = atoi(argv[6]);
00046   remain = atof(argv[7]);
00047
00048   /* Convert... */
00049   time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
00050   printf("%.2f\n", jsec);
00051
00052   return EXIT_SUCCESS;
00053 }
```

Here is the call graph for this function:



5.30 time2jsec.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 int main(
00028   int argc,
00029   char *argv[]) {
```

```
00030
00031   double jsec, remain;
00032
00033   int day, hour, min, mon, sec, year;
00034
00035   /* Check arguments... */
00036   if (argc < 8)
00037     ERRMSG("Give parameters: <year> <mon> <day> <hour> <min> <sec> <remain>");
00038
00039   /* Read arguments... */
00040   year = atoi(argv[1]);
00041   mon = atoi(argv[2]);
00042   day = atoi(argv[3]);
00043   hour = atoi(argv[4]);
00044   min = atoi(argv[5]);
00045   sec = atoi(argv[6]);
00046   remain = atof(argv[7]);
00047
00048   /* Convert... */
00049   time2jsec(year, mon, day, hour, min, sec, remain, &jsec);
00050   printf("%.2f\n", jsec);
00051
00052   return EXIT_SUCCESS;
00053 }
```

5.31 trac.c File Reference

Lagrangian particle dispersion model.

Functions

- void init_simtime (ctl_t *ctl, atm_t *atm)

  Set simulation time interval.
- void module_advection (met_t *met0, met_t *met1, atm_t *atm, int ip, double dt)

  Calculate advection of air parcels.
- void module_decay (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, double dt)

  Calculate exponential decay of particle mass.
- void module_diffusion_meso (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, double dt, gsl_rng *rng)

  Calculate mesoscale diffusion.
- void module_diffusion_turb (ctl_t *ctl, atm_t *atm, int ip, double dt, gsl_rng *rng)

  Calculate turbulent diffusion.
- void module_isosurf (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip)

  Force air parcels to stay on isosurface.
- void module_meteo (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip)

  Interpolate meteorological data for air parcel positions.
- void module_position (met_t *met0, met_t *met1, atm_t *atm, int ip)

  Check position of air parcels.
- void module_sedi (ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, int ip, double dt)

  Calculate sedimentation of air parcels.
- void write_output (const char *dirname, ctl_t *ctl, met_t *met0, met_t *met1, atm_t *atm, double t)

  Write simulation output.
- int main (int argc, char *argv[])

5.31.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.31.2 Function Documentation

5.31.2.1 void init_simtime (ctl_t * *ctl*, atm_t * *atm*)

Set simulation time interval.

Definition at line 398 of file trac.c.

```
00400                {
00401
00402   /* Set inital and final time... */
00403   if (ctl->direction == 1) {
00404     if (ctl->t_start < -1e99)
00405       ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00406     if (ctl->t_stop < -1e99)
00407       ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00408   } else if (ctl->direction == -1) {
00409     if (ctl->t_stop < -1e99)
00410       ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00411     if (ctl->t_start < -1e99)
00412       ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00413   }
00414
00415   /* Check time... */
00416   if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
00417     ERRMSG("Nothing to do!");
00418 }
```

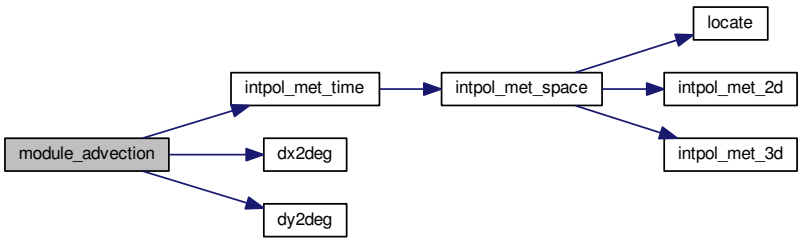
5.31.2.2 void module_advection (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate advection of air parcels.

Definition at line 422 of file trac.c.

```
00427              {
00428
00429   double v[3], xm[3];
00430
00431   /* Interpolate meteorological data... */
00432   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00433                   atm->lon[ip], atm->lat[ip], NULL, NULL,
00434                   &v[0], &v[1], &v[2], NULL, NULL);
00435
00436   /* Get position of the mid point... */
00437   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00438   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00439   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00440
00441   /* Interpolate meteorological data for mid point... */
00442   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00443                   xm[2], xm[0], xm[1], NULL, NULL,
00444                   &v[0], &v[1], &v[2], NULL, NULL);
00445
00446   /* Save new position... */
00447   atm->time[ip] += dt;
00448   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00449   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00450   atm->p[ip] += dt * v[2];
00451 }
```

Here is the call graph for this function:



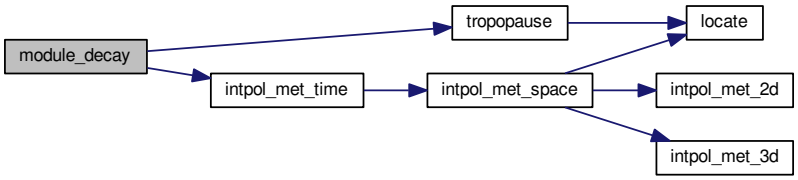
5.31.2.3 void module_decay (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate exponential decay of particle mass.

Definition at line 455 of file trac.c.

```
00461             {
00462
00463   double ps, pt, tdec;
00464
00465   /* Check lifetime values... */
00466   if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
      qnt_m < 0)
00467     return;
00468
00469   /* Set constant lifetime... */
00470   if (ctl->tdec_trop == ctl->tdec_strat)
00471     tdec = ctl->tdec_trop;
00472
00473   /* Set altitude-dependent lifetime... */
00474   else {
00475
00476     /* Get surface pressure... */
00477     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00478                     atm->lon[ip], atm->lat[ip], &ps, NULL,
00479                     NULL, NULL, NULL, NULL, NULL);
00480
00481     /* Get tropopause pressure... */
00482     pt = tropopause(atm->time[ip], atm->lat[ip]);
00483
00484     /* Set lifetime... */
00485     if (atm->p[ip] <= pt)
00486       tdec = ctl->tdec_strat;
00487     else
00488       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00489   }
00490
00491   /* Calculate exponential decay... */
00492   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00493 }
```

Here is the call graph for this function:



5.31.2.4 void module_diffusion_meso (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*, gsl_rng * *rng*)

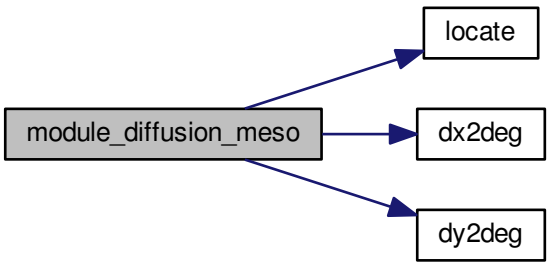
Calculate mesoscale diffusion.

Definition at line 497 of file trac.c.

```
00504                 {
00505
00506   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00507
00508   int ix, iy, iz;
00509
00510   /* Calculate mesoscale velocity fluctuations... */
00511   if (ctl->turb_meso > 0) {
00512
00513     /* Get indices... */
00514     ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00515     iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00516     iz = locate(met0->p, met0->np, atm->p[ip]);
00517
00518     /* Collect local wind data... */
00519     u[0] = met0->u[ix][iy][iz];
00520     u[1] = met0->u[ix + 1][iy][iz];
00521     u[2] = met0->u[ix][iy + 1][iz];
00522     u[3] = met0->u[ix + 1][iy + 1][iz];
00523     u[4] = met0->u[ix][iy][iz + 1];
00524     u[5] = met0->u[ix + 1][iy][iz + 1];
00525     u[6] = met0->u[ix][iy + 1][iz + 1];
00526     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00527
00528     v[0] = met0->v[ix][iy][iz];
00529     v[1] = met0->v[ix + 1][iy][iz];
00530     v[2] = met0->v[ix][iy + 1][iz];
00531     v[3] = met0->v[ix + 1][iy + 1][iz];
00532     v[4] = met0->v[ix][iy][iz + 1];
00533     v[5] = met0->v[ix + 1][iy][iz + 1];
00534     v[6] = met0->v[ix][iy + 1][iz + 1];
00535     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00536
00537     w[0] = met0->w[ix][iy][iz];
00538     w[1] = met0->w[ix + 1][iy][iz];
00539     w[2] = met0->w[ix][iy + 1][iz];
00540     w[3] = met0->w[ix + 1][iy + 1][iz];
00541     w[4] = met0->w[ix][iy][iz + 1];
00542     w[5] = met0->w[ix + 1][iy][iz + 1];
00543     w[6] = met0->w[ix][iy + 1][iz + 1];
00544     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00545
00546     /* Get indices... */
00547     ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00548     iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00549     iz = locate(met1->p, met1->np, atm->p[ip]);
00550
00551     /* Collect local wind data... */
00552     u[8] = met1->u[ix][iy][iz];
00553     u[9] = met1->u[ix + 1][iy][iz];
00554     u[10] = met1->u[ix][iy + 1][iz];
00555     u[11] = met1->u[ix + 1][iy + 1][iz];
00556     u[12] = met1->u[ix][iy][iz + 1];
00557     u[13] = met1->u[ix + 1][iy][iz + 1];
00558     u[14] = met1->u[ix][iy + 1][iz + 1];
00559     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00560
00561     v[8] = met1->v[ix][iy][iz];
00562     v[9] = met1->v[ix + 1][iy][iz];
00563     v[10] = met1->v[ix][iy + 1][iz];
00564     v[11] = met1->v[ix + 1][iy + 1][iz];
00565     v[12] = met1->v[ix][iy][iz + 1];
00566     v[13] = met1->v[ix + 1][iy][iz + 1];
00567     v[14] = met1->v[ix][iy + 1][iz + 1];
00568     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00569
00570     w[8] = met1->w[ix][iy][iz];
00571     w[9] = met1->w[ix + 1][iy][iz];
00572     w[10] = met1->w[ix][iy + 1][iz];
00573     w[11] = met1->w[ix + 1][iy + 1][iz];
00574     w[12] = met1->w[ix][iy][iz + 1];
00575     w[13] = met1->w[ix + 1][iy][iz + 1];
00576     w[14] = met1->w[ix][iy + 1][iz + 1];
00577     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00578
00579     /* Get standard deviations of local wind data... */
00580     usig = gsl_stats_sd(u, 1, 16);
00581     vsig = gsl_stats_sd(v, 1, 16);
00582     wsig = gsl_stats_sd(w, 1, 16);
00583
00584     /* Set temporal correlations for mesoscale fluctuations... */
00585     r = 1 - 2 * fabs(dt) / ctl->dt_met;
00586     rs = sqrt(1 - r * r);
00587
00588     /* Calculate mesoscale wind fluctuations... */
00589     atm->up[ip] =
00590       r * atm->up[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
```

```
00591                                                   ctl->turb_meso * usig);
00592     atm->vp[ip] =
00593       r * atm->vp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00594                                                   ctl->turb_meso * vsig);
00595     atm->wp[ip] =
00596       r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00597                                                   ctl->turb_meso * wsig);
00598
00599     /* Calculate air parcel displacement... */
00600     atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00601     atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00602     atm->p[ip] += atm->wp[ip] * dt;
00603   }
00604 }
```

Here is the call graph for this function:



5.31.2.5 void module_diffusion_turb (ctl_t * *ctl*, atm_t * *atm*, int *ip*, double *dt*, gsl_rng * *rng*)

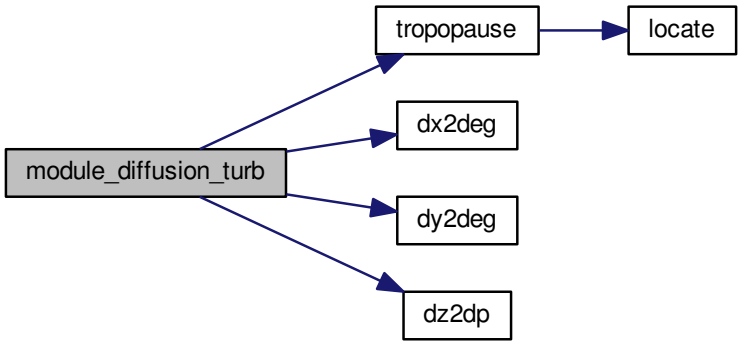
Calculate turbulent diffusion.

Definition at line 608 of file trac.c.

```
00613                 {
00614
00615   double dx, dz, pt, p0, p1, w;
00616
00617   /* Get tropopause pressure... */
00618   pt = tropopause(atm->time[ip], atm->lat[ip]);
00619
00620   /* Get weighting factor... */
00621   p1 = pt * 0.866877899;
00622   p0 = pt / 0.866877899;
00623   if (atm->p[ip] > p0)
00624     w = 1;
00625   else if (atm->p[ip] < p1)
00626     w = 0;
00627   else
00628     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00629
00630   /* Set diffusivitiy... */
00631   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00632   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00633
00634   /* Horizontal turbulent diffusion... */
00635   if (dx > 0) {
00636     atm->lon[ip]
00637       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00638                 / 1000., atm->lat[ip]);
00639     atm->lat[ip]
00640       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00641                 / 1000.);
00642   }
```

```
00643
00644   /* Vertical turbulent diffusion... */
00645   if (dz > 0)
00646     atm->p[ip]
00647       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00648                / 1000., atm->p[ip]);
00649 }
```

Here is the call graph for this function:



5.31.2.6 void module_isosurf (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*)

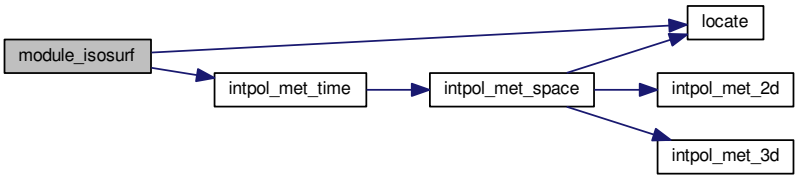
Force air parcels to stay on isosurface.

Definition at line 653 of file trac.c.

```
00658           {
00659
00660   static double *iso, *ps, t, *ts;
00661
00662   static int idx, ip2, n, nb = 100000;
00663
00664   FILE *in;
00665
00666   char line[LEN];
00667
00668   /* Check control parameter... */
00669   if (ctl->isosurf < 1 || ctl->isosurf > 4)
00670     return;
00671
00672   /* Initialize... */
00673   if (ip < 0) {
00674
00675     /* Allocate... */
00676     ALLOC(iso, double,
00677           NP);
00678     ALLOC(ps, double,
00679           nb);
00680     ALLOC(ts, double,
00681           nb);
00682
00683     /* Save pressure... */
00684     if (ctl->isosurf == 1)
00685       for (ip2 = 0; ip2 < atm->np; ip2++)
00686         iso[ip2] = atm->p[ip2];
00687
00688     /* Save density... */
00689     else if (ctl->isosurf == 2)
00690       for (ip2 = 0; ip2 < atm->np; ip2++) {
00691         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
```

```
00692                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00693                         NULL, NULL, NULL);
00694         iso[ip2] = atm->p[ip2] / t;
00695       }
00696
00697     /* Save potential temperature... */
00698     else if (ctl->isosurf == 3)
00699       for (ip2 = 0; ip2 < atm->np; ip2++) {
00700         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00701                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00702                         NULL, NULL, NULL);
00703         iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00704       }
00705
00706     /* Read balloon pressure data... */
00707     else if (ctl->isosurf == 4) {
00708
00709       /* Write info... */
00710       printf("Read balloon pressure data: %s\n", ctl->balloon);
00711
00712       /* Open file... */
00713       if (!(in = fopen(ctl->balloon, "r")))
00714         ERRMSG("Cannot open file!");
00715
00716       /* Read pressure time series... */
00717       while (fgets(line, LEN, in))
00718         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00719           if ((++n) > 100000)
00720             ERRMSG("Too many data points!");
00721
00722       /* Check number of points... */
00723       if (n < 1)
00724         ERRMSG("Could not read any data!");
00725
00726       /* Close file... */
00727       fclose(in);
00728     }
00729
00730     /* Leave initialization... */
00731     return;
00732   }
00733
00734   /* Restore pressure... */
00735   if (ctl->isosurf == 1)
00736     atm->p[ip] = iso[ip];
00737
00738   /* Restore density... */
00739   else if (ctl->isosurf == 2) {
00740     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00741                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00742     atm->p[ip] = iso[ip] * t;
00743   }
00744
00745   /* Restore potential temperature... */
00746   else if (ctl->isosurf == 3) {
00747     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00748                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00749     atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00750   }
00751
00752   /* Interpolate pressure... */
00753   else if (ctl->isosurf == 4) {
00754     if (atm->time[ip] <= ts[0])
00755       atm->p[ip] = ps[0];
00756     else if (atm->time[ip] >= ts[n - 1])
00757       atm->p[ip] = ps[n - 1];
00758     else {
00759       idx = locate(ts, n, atm->time[ip]);
00760       atm->p[ip] = LIN(ts[idx], ps[idx],
00761                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00762     }
00763   }
00764 }
```

Here is the call graph for this function:



5.31.2.7 void module_meteo (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*)

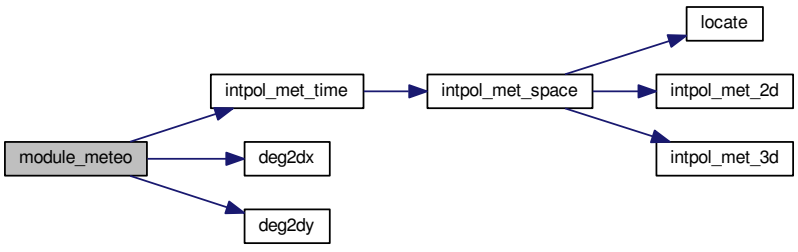
Interpolate meteorological data for air parcel positions.

Definition at line 768 of file trac.c.

```
00773           {
00774
00775   double b, c, ps, p1, p_hno3, p_h2o, t, t1, term1, term2,
00776     u, u1, v, v1, w, x1, x2, h2o, o3, grad, vort;
00777
00778   /* Interpolate meteorological data... */
00779   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00780                   atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00781
00782   /* Set surface pressure... */
00783   if (ctl->qnt_ps >= 0)
00784     atm->q[ctl->qnt_ps][ip] = ps;
00785
00786   /* Set pressure... */
00787   if (ctl->qnt_p >= 0)
00788     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00789
00790   /* Set temperature... */
00791   if (ctl->qnt_t >= 0)
00792     atm->q[ctl->qnt_t][ip] = t;
00793
00794   /* Set zonal wind... */
00795   if (ctl->qnt_u >= 0)
00796     atm->q[ctl->qnt_u][ip] = u;
00797
00798   /* Set meridional wind... */
00799   if (ctl->qnt_v >= 0)
00800     atm->q[ctl->qnt_v][ip] = v;
00801
00802   /* Set vertical velocity... */
00803   if (ctl->qnt_w >= 0)
00804     atm->q[ctl->qnt_w][ip] = w;
00805
00806   /* Set water vapor vmr... */
00807   if (ctl->qnt_h2o >= 0)
00808     atm->q[ctl->qnt_h2o][ip] = h2o;
00809
00810   /* Set ozone vmr... */
00811   if (ctl->qnt_o3 >= 0)
00812     atm->q[ctl->qnt_o3][ip] = o3;
00813
00814   /* Calculate potential temperature... */
00815   if (ctl->qnt_theta >= 0)
00816     atm->q[ctl->qnt_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00817
00818   /* Calculate potential vorticity... */
00819   if (ctl->qnt_pv >= 0) {
00820
00821     /* Absolute vorticity... */
00822     vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00823     if (fabs(atm->lat[ip]) < 89.) {
00824       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00825                       (atm->lon[ip] >=
```

```
00826                       0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00827                      atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00828       vort += (v1 - v) / 1000.
00829         / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00830     }
00831     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00832                     (atm->lat[ip] >=
00833                      0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
00834                     &u1, NULL, NULL, NULL, NULL);
00835     vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00836
00837     /* Potential temperature gradient... */
00838     p1 = 0.85 * atm->p[ip];
00839     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
00840                     atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL, NULL);
00841     grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00842       / (100. * (p1 - atm->p[ip]));
00843
00844     /* Calculate PV... */
00845     atm->q[ctl->qnt_pv][ip] = -1e6 * G0 * vort * grad;
00846   }
00847
00848   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00849   if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00850     atm->q[ctl->qnt_tice][ip] = -2663.5
00851       / (log10(ctl->psc_h2o * atm->p[ip] * 100.) - 12.537);
00852
00853   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00854   if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
00855     p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00856     p_h2o = ctl->psc_h2o * atm->p[ip] / 1.333224;
00857     term1 = 38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o);
00858     term2 = 0.009179 - 0.00088 * log10(p_h2o);
00859     b = term1 / term2;
00860     c = -11397.0 / term2;
00861     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00862     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00863     if (x1 > 0)
00864       atm->q[ctl->qnt_tnat][ip] = x1;
00865     if (x2 > 0)
00866       atm->q[ctl->qnt_tnat][ip] = x2;
00867   }
00868
00869   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00870   if (ctl->qnt_tsts >= 0) {
00871     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00872       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00873     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00874                                        + atm->q[ctl->qnt_tnat][ip]);
00875   }
00876 }
```

Here is the call graph for this function:



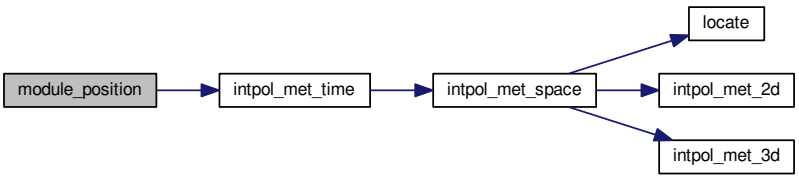
5.31.2.8 void module_position (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*)

Check position of air parcels.

Definition at line 880 of file trac.c.

```
00884           {
00885
00886     double ps;
00887
00888     /* Calculate modulo... */
00889     atm->lon[ip] = fmod(atm->lon[ip], 360);
00890     atm->lat[ip] = fmod(atm->lat[ip], 360);
00891
00892     /* Check latitude... */
00893     while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00894       if (atm->lat[ip] > 90) {
00895         atm->lat[ip] = 180 - atm->lat[ip];
00896         atm->lon[ip] += 180;
00897       }
00898       if (atm->lat[ip] < -90) {
00899         atm->lat[ip] = -180 - atm->lat[ip];
00900         atm->lon[ip] += 180;
00901       }
00902     }
00903
00904     /* Check longitude... */
00905     while (atm->lon[ip] < -180)
00906       atm->lon[ip] += 360;
00907     while (atm->lon[ip] >= 180)
00908       atm->lon[ip] -= 360;
00909
00910     /* Get surface pressure... */
00911     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00912                     atm->lon[ip], atm->lat[ip], &ps, NULL,
00913                     NULL, NULL, NULL, NULL, NULL);
00914
00915     /* Check pressure... */
00916     if (atm->p[ip] > ps)
00917       atm->p[ip] = ps;
00918     else if (atm->p[ip] < met0->p[met0->np - 1])
00919       atm->p[ip] = met0->p[met0->np - 1];
00920   }
```

Here is the call graph for this function:



5.31.2.9 void module_sedi (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

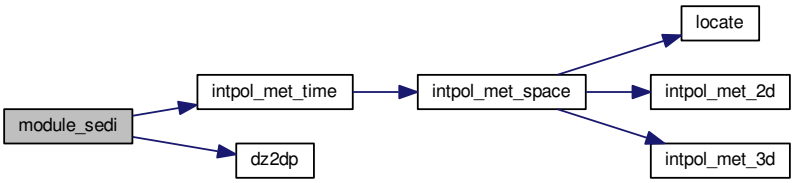
Calculate sedimentation of air parcels.

Definition at line 924 of file trac.c.

```
00930             {
00931
00932     /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00933     const double A = 1.249, B = 0.42, C = 0.87;
00934
00935     /* Specific gas constant for dry air [J/(kg K)]: */
00936     const double R = 287.058;
00937
00938     /* Average mass of an air molecule [kg/molec]: */
00939     const double m = 4.8096e-26;
00940
00941     double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00942
00943     /* Check if parameters are available... */
```

```
00944   if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
00945     return;
00946
00947   /* Convert units... */
00948   p = 100 * atm->p[ip];
00949   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00950   rho_p = atm->q[ctl->qnt_rho][ip];
00951
00952   /* Get temperature... */
00953   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00954                   atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00955
00956   /* Density of dry air... */
00957   rho = p / (R * T);
00958
00959   /* Dynamic viscosity of air... */
00960   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00961
00962   /* Thermal velocity of an air molecule... */
00963   v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00964
00965   /* Mean free path of an air molecule... */
00966   lambda = 2 * eta / (rho * v);
00967
00968   /* Knudsen number for air... */
00969   K = lambda / r_p;
00970
00971   /* Cunningham slip-flow correction... */
00972   G = 1 + K * (A + B * exp(-C / K));
00973
00974   /* Sedimentation (fall) velocity... */
00975   v_p =
00976     2. * gsl_pow_2(r_p) * (rho_p -
00977                            rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00978
00979   /* Calculate pressure change... */
00980   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00981 }
```

Here is the call graph for this function:



5.31.2.10 void write_output (const char * *dirname*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

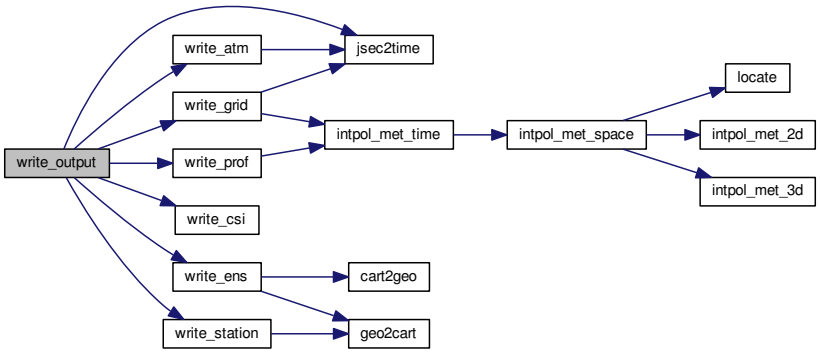
Write simulation output.

Definition at line 985 of file trac.c.

```
00991           {
00992
00993   char filename[LEN];
00994
00995   double r;
00996
00997   int year, mon, day, hour, min, sec;
00998
00999   /* Get time... */
01000   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01001
01002   /* Write atmospheric data... */
```

```
01003   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01004     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01005             dirname, ctl->atm_basename, year, mon, day, hour, min);
01006     write_atm(filename, ctl, atm, t);
01007   }
01008
01009   /* Write CSI data... */
01010   if (ctl->csi_basename[0] != '-') {
01011     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01012     write_csi(filename, ctl, atm, t);
01013   }
01014
01015   /* Write ensemble data... */
01016   if (ctl->ens_basename[0] != '-') {
01017     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01018     write_ens(filename, ctl, atm, t);
01019   }
01020
01021   /* Write gridded data... */
01022   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01023     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01024             dirname, ctl->grid_basename, year, mon, day, hour, min);
01025     write_grid(filename, ctl, met0, met1, atm, t);
01026   }
01027
01028   /* Write profile data... */
01029   if (ctl->prof_basename[0] != '-') {
01030     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01031     write_prof(filename, ctl, met0, met1, atm, t);
01032   }
01033
01034   /* Write station data... */
01035   if (ctl->stat_basename[0] != '-') {
01036     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01037     write_station(filename, ctl, atm, t);
01038   }
01039 }
```

Here is the call graph for this function:



5.31.2.11 int main (int *argc*, char * *argv[ ]*)

Definition at line 160 of file trac.c.

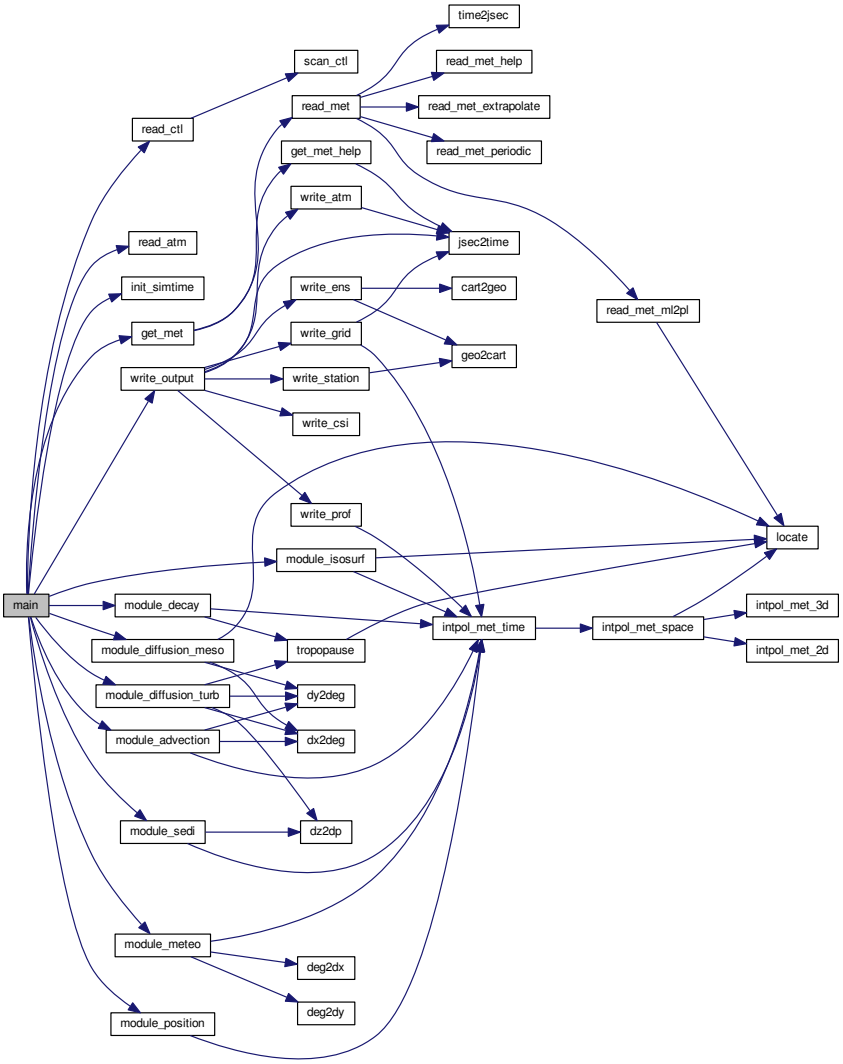
```
00162                {
00163
00164   ctl_t ctl;
00165
00166   atm_t *atm;
00167
00168   met_t *met0, *met1;
00169
00170   gsl_rng *rng[NTHREADS];
00171
```

```
00172   FILE *dirlist;
00173
00174   char dirname[LEN], filename[LEN];
00175
00176   double *dt, t, t0;
00177
00178   int i, ip, ntask = 0, rank = 0, size = 1;
00179
00180 #ifdef MPI
00181   /* Initialize MPI... */
00182   MPI_Init(&argc, &argv);
00183   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00184   MPI_Comm_size(MPI_COMM_WORLD, &size);
00185 #endif
00186
00187   /* Check arguments... */
00188   if (argc < 5)
00189     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00190
00191   /* Open directory list... */
00192   if (!(dirlist = fopen(argv[1], "r")))
00193     ERRMSG("Cannot open directory list!");
00194
00195   /* Loop over directories... */
00196   while (fscanf(dirlist, "%s", dirname) != EOF) {
00197
00198     /* MPI parallelization... */
00199     if ((++ntask) % size != rank)
00200       continue;
00201
00202     /* ------------------------------------------------------------
00203        Initialize model run...
00204        ------------------------------------------------------------ */
00205
00206     /* Set timers... */
00207     START_TIMER(TIMER_TOTAL);
00208     START_TIMER(TIMER_INIT);
00209
00210     /* Allocate... */
00211     ALLOC(atm, atm_t, 1);
00212     ALLOC(met0, met_t, 1);
00213     ALLOC(met1, met_t, 1);
00214     ALLOC(dt, double,
00215           NP);
00216
00217     /* Read control parameters... */
00218     sprintf(filename, "%s/%s", dirname, argv[2]);
00219     read_ctl(filename, argc, argv, &ctl);
00220
00221     /* Initialize random number generators... */
00222     gsl_rng_env_setup();
00223     for (i = 0; i < NTHREADS; i++)
00224       rng[i] = gsl_rng_alloc(gsl_rng_default);
00225
00226     /* Read atmospheric data... */
00227     sprintf(filename, "%s/%s", dirname, argv[3]);
00228     read_atm(filename, &ctl, atm);
00229
00230     /* Get simulation time interval... */
00231     init_simtime(&ctl, atm);
00232
00233     /* Get rounded start time... */
00234     if (ctl.direction == 1)
00235       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00236     else
00237       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00238
00239     /* Set timers... */
00240     STOP_TIMER(TIMER_INIT);
00241
00242     /* ------------------------------------------------------------
00243        Loop over timesteps...
00244        ------------------------------------------------------------ */
00245
00246     /* Loop over timesteps... */
00247     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00248          t += ctl.direction * ctl.dt_mod) {
00249
00250       /* Adjust length of final time step... */
00251       if (ctl.direction * (t - ctl.t_stop) > 0)
00252         t = ctl.t_stop;
00253
00254       /* Set time steps for air parcels... */
00255       for (ip = 0; ip < atm->np; ip++)
00256         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00257              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00258              && ctl.direction * (atm->time[ip] - t) < 0))
```

```
00259           dt[ip] = t - atm->time[ip];
00260         else
00261           dt[ip] = GSL_NAN;
00262
00263       /* Get meteorological data... */
00264       START_TIMER(TIMER_INPUT);
00265       get_met(&ctl, argv[4], t, met0, met1);
00266       STOP_TIMER(TIMER_INPUT);
00267
00268       /* Initialize isosurface... */
00269       START_TIMER(TIMER_ISOSURF);
00270       if (t == t0)
00271         module_isosurf(&ctl, met0, met1, atm, -1);
00272       STOP_TIMER(TIMER_ISOSURF);
00273
00274       /* Advection... */
00275       START_TIMER(TIMER_ADVECT);
00276 #pragma omp parallel for default(shared) private(ip)
00277       for (ip = 0; ip < atm->np; ip++)
00278         if (gsl_finite(dt[ip]))
00279           module_advection(met0, met1, atm, ip, dt[ip]);
00280       STOP_TIMER(TIMER_ADVECT);
00281
00282       /* Turbulent diffusion... */
00283       START_TIMER(TIMER_DIFFTURB);
00284 #pragma omp parallel for default(shared) private(ip)
00285       for (ip = 0; ip < atm->np; ip++)
00286         if (gsl_finite(dt[ip]))
00287           module_diffusion_turb(&ctl, atm, ip, dt[ip],
00288                                 rng[omp_get_thread_num()]);
00289       STOP_TIMER(TIMER_DIFFTURB);
00290
00291       /* Mesoscale diffusion... */
00292       START_TIMER(TIMER_DIFFMESO);
00293 #pragma omp parallel for default(shared) private(ip)
00294       for (ip = 0; ip < atm->np; ip++)
00295         if (gsl_finite(dt[ip]))
00296           module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00297                                 rng[omp_get_thread_num()]);
00298       STOP_TIMER(TIMER_DIFFMESO);
00299
00300       /* Sedimentation... */
00301       START_TIMER(TIMER_SEDI);
00302 #pragma omp parallel for default(shared) private(ip)
00303       for (ip = 0; ip < atm->np; ip++)
00304         if (gsl_finite(dt[ip]))
00305           module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00306       STOP_TIMER(TIMER_SEDI);
00307
00308       /* Isosurface... */
00309       START_TIMER(TIMER_ISOSURF);
00310 #pragma omp parallel for default(shared) private(ip)
00311       for (ip = 0; ip < atm->np; ip++)
00312         module_isosurf(&ctl, met0, met1, atm, ip);
00313       STOP_TIMER(TIMER_ISOSURF);
00314
00315       /* Position... */
00316       START_TIMER(TIMER_POSITION);
00317 #pragma omp parallel for default(shared) private(ip)
00318       for (ip = 0; ip < atm->np; ip++)
00319         module_position(met0, met1, atm, ip);
00320       STOP_TIMER(TIMER_POSITION);
00321
00322       /* Meteorological data... */
00323       START_TIMER(TIMER_METEO);
00324 #pragma omp parallel for default(shared) private(ip)
00325       for (ip = 0; ip < atm->np; ip++)
00326         module_meteo(&ctl, met0, met1, atm, ip);
00327       STOP_TIMER(TIMER_METEO);
00328
00329       /* Decay... */
00330       START_TIMER(TIMER_DECAY);
00331 #pragma omp parallel for default(shared) private(ip)
00332       for (ip = 0; ip < atm->np; ip++)
00333         if (gsl_finite(dt[ip]))
00334           module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00335       STOP_TIMER(TIMER_DECAY);
00336
00337       /* Write output... */
00338       START_TIMER(TIMER_OUTPUT);
00339       write_output(dirname, &ctl, met0, met1, atm, t);
00340       STOP_TIMER(TIMER_OUTPUT);
00341     }
00342
00343     /* ------------------------------------------------------------
00344        Finalize model run...
00345        ------------------------------------------------------------ */
```

```
00346
00347     /* Report timers... */
00348     STOP_TIMER(TIMER_TOTAL);
00349     PRINT_TIMER(TIMER_TOTAL);
00350     PRINT_TIMER(TIMER_INIT);
00351     PRINT_TIMER(TIMER_INPUT);
00352     PRINT_TIMER(TIMER_OUTPUT);
00353     PRINT_TIMER(TIMER_ADVECT);
00354     PRINT_TIMER(TIMER_DECAY);
00355     PRINT_TIMER(TIMER_DIFFMESO);
00356     PRINT_TIMER(TIMER_DIFFTURB);
00357     PRINT_TIMER(TIMER_ISOSURF);
00358     PRINT_TIMER(TIMER_METEO);
00359     PRINT_TIMER(TIMER_POSITION);
00360     PRINT_TIMER(TIMER_SEDI);
00361
00362     /* Report memory usage... */
00363     printf("MEMORY_ATM = %g MByte\n", 2. * sizeof(atm_t) / 1024. / 1024.);
00364     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00365     printf("MEMORY_DYNAMIC = %g MByte\n",
00366            NP * sizeof(double) / 1024. / 1024.);
00367     printf("MEMORY_STATIC = %g MByte\n",
00368            (((EX + EY) + (2 + NQ) * GX * GY * GZ) * sizeof(double)
00369             + (EX * EY + EX * EY * EP) * sizeof(float)
00370             + (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00371
00372     /* Report problem size... */
00373     printf("SIZE_NP = %d\n", atm->np);
00374     printf("SIZE_TASKS = %d\n", size);
00375     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00376
00377     /* Free random number generators... */
00378     for (i = 0; i < NTHREADS; i++)
00379       gsl_rng_free(rng[i]);
00380
00381     /* Free... */
00382     free(atm);
00383     free(met0);
00384     free(met1);
00385     free(dt);
00386   }
00387
00388 #ifdef MPI
00389   /* Finalize MPI... */
00390   MPI_Finalize();
00391 #endif
00392
00393   return EXIT_SUCCESS;
00394 }
```

Here is the call graph for this function:



5.32 trac.c

```
00001 /*
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copyright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 #ifdef MPI
```

```
00028 #include "mpi.h"
00029 #endif
00030
00031 /* ------------------------------------------------------------
00032    Defines...
00033    ------------------------------------------------------------ */
00034
00036 #define TIMER_TOTAL 0
00037
00039 #define TIMER_INIT 1
00040
00042 #define TIMER_INPUT 2
00043
00045 #define TIMER_OUTPUT 3
00046
00048 #define TIMER_ADVECT 4
00049
00051 #define TIMER_DECAY 5
00052
00054 #define TIMER_DIFFMESO 6
00055
00057 #define TIMER_DIFFTURB 7
00058
00060 #define TIMER_ISOSURF 8
00061
00063 #define TIMER_METEO 9
00064
00066 #define TIMER_POSITION 10
00067
00069 #define TIMER_SEDI 11
00070
00071 /* ------------------------------------------------------------
00072    Functions...
00073    ------------------------------------------------------------ */
00074
00076 void init_simtime(
00077   ctl_t * ctl,
00078   atm_t * atm);
00079
00081 void module_advection(
00082   met_t * met0,
00083   met_t * met1,
00084   atm_t * atm,
00085   int ip,
00086   double dt);
00087
00089 void module_decay(
00090   ctl_t * ctl,
00091   met_t * met0,
00092   met_t * met1,
00093   atm_t * atm,
00094   int ip,
00095   double dt);
00096
00098 void module_diffusion_meso(
00099   ctl_t * ctl,
00100   met_t * met0,
00101   met_t * met1,
00102   atm_t * atm,
00103   int ip,
00104   double dt,
00105   gsl_rng * rng);
00106
00108 void module_diffusion_turb(
00109   ctl_t * ctl,
00110   atm_t * atm,
00111   int ip,
00112   double dt,
00113   gsl_rng * rng);
00114
00116 void module_isosurf(
00117   ctl_t * ctl,
00118   met_t * met0,
00119   met_t * met1,
00120   atm_t * atm,
00121   int ip);
00122
00124 void module_meteo(
00125   ctl_t * ctl,
00126   met_t * met0,
00127   met_t * met1,
00128   atm_t * atm,
00129   int ip);
00130
00132 void module_position(
00133   met_t * met0,
00134   met_t * met1,
```

```
00135   atm_t * atm,
00136   int ip);
00137
00139 void module_sedi(
00140   ctl_t * ctl,
00141   met_t * met0,
00142   met_t * met1,
00143   atm_t * atm,
00144   int ip,
00145   double dt);
00146
00148 void write_output(
00149   const char *dirname,
00150   ctl_t * ctl,
00151   met_t * met0,
00152   met_t * met1,
00153   atm_t * atm,
00154   double t);
00155
00156 /* ------------------------------------------------------------
00157    Main...
00158    ------------------------------------------------------------ */
00159
00160 int main(
00161   int argc,
00162   char *argv[]) {
00163
00164   ctl_t ctl;
00165
00166   atm_t *atm;
00167
00168   met_t *met0, *met1;
00169
00170   gsl_rng *rng[NTHREADS];
00171
00172   FILE *dirlist;
00173
00174   char dirname[LEN], filename[LEN];
00175
00176   double *dt, t, t0;
00177
00178   int i, ip, ntask = 0, rank = 0, size = 1;
00179
00180 #ifdef MPI
00181   /* Initialize MPI... */
00182   MPI_Init(&argc, &argv);
00183   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00184   MPI_Comm_size(MPI_COMM_WORLD, &size);
00185 #endif
00186
00187   /* Check arguments... */
00188   if (argc < 5)
00189     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00190
00191   /* Open directory list... */
00192   if (!(dirlist = fopen(argv[1], "r")))
00193     ERRMSG("Cannot open directory list!");
00194
00195   /* Loop over directories... */
00196   while (fscanf(dirlist, "%s", dirname) != EOF) {
00197
00198     /* MPI parallelization... */
00199     if ((++ntask) % size != rank)
00200       continue;
00201
00202     /* ------------------------------------------------------------
00203        Initialize model run...
00204        ------------------------------------------------------------ */
00205
00206     /* Set timers... */
00207     START_TIMER(TIMER_TOTAL);
00208     START_TIMER(TIMER_INIT);
00209
00210     /* Allocate... */
00211     ALLOC(atm, atm_t, 1);
00212     ALLOC(met0, met_t, 1);
00213     ALLOC(met1, met_t, 1);
00214     ALLOC(dt, double,
00215           NP);
00216
00217     /* Read control parameters... */
00218     sprintf(filename, "%s/%s", dirname, argv[2]);
00219     read_ctl(filename, argc, argv, &ctl);
00220
00221     /* Initialize random number generators... */
00222     gsl_rng_env_setup();
00223     for (i = 0; i < NTHREADS; i++)
```

```
00224       rng[i] = gsl_rng_alloc(gsl_rng_default);
00225
00226     /* Read atmospheric data... */
00227     sprintf(filename, "%s/%s", dirname, argv[3]);
00228     read_atm(filename, &ctl, atm);
00229
00230     /* Get simulation time interval... */
00231     init_simtime(&ctl, atm);
00232
00233     /* Get rounded start time... */
00234     if (ctl.direction == 1)
00235       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00236     else
00237       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00238
00239     /* Set timers... */
00240     STOP_TIMER(TIMER_INIT);
00241
00242     /* ------------------------------------------------------------
00243        Loop over timesteps...
00244        ------------------------------------------------------------ */
00245
00246     /* Loop over timesteps... */
00247     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00248          t += ctl.direction * ctl.dt_mod) {
00249
00250       /* Adjust length of final time step... */
00251       if (ctl.direction * (t - ctl.t_stop) > 0)
00252         t = ctl.t_stop;
00253
00254       /* Set time steps for air parcels... */
00255       for (ip = 0; ip < atm->np; ip++)
00256         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00257              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00258              && ctl.direction * (atm->time[ip] - t) < 0))
00259           dt[ip] = t - atm->time[ip];
00260         else
00261           dt[ip] = GSL_NAN;
00262
00263       /* Get meteorological data... */
00264       START_TIMER(TIMER_INPUT);
00265       get_met(&ctl, argv[4], t, met0, met1);
00266       STOP_TIMER(TIMER_INPUT);
00267
00268       /* Initialize isosurface... */
00269       START_TIMER(TIMER_ISOSURF);
00270       if (t == t0)
00271         module_isosurf(&ctl, met0, met1, atm, -1);
00272       STOP_TIMER(TIMER_ISOSURF);
00273
00274       /* Advection... */
00275       START_TIMER(TIMER_ADVECT);
00276 #pragma omp parallel for default(shared) private(ip)
00277       for (ip = 0; ip < atm->np; ip++)
00278         if (gsl_finite(dt[ip]))
00279           module_advection(met0, met1, atm, ip, dt[ip]);
00280       STOP_TIMER(TIMER_ADVECT);
00281
00282       /* Turbulent diffusion... */
00283       START_TIMER(TIMER_DIFFTURB);
00284 #pragma omp parallel for default(shared) private(ip)
00285       for (ip = 0; ip < atm->np; ip++)
00286         if (gsl_finite(dt[ip]))
00287           module_diffusion_turb(&ctl, atm, ip, dt[ip],
00288                                 rng[omp_get_thread_num()]);
00289       STOP_TIMER(TIMER_DIFFTURB);
00290
00291       /* Mesoscale diffusion... */
00292       START_TIMER(TIMER_DIFFMESO);
00293 #pragma omp parallel for default(shared) private(ip)
00294       for (ip = 0; ip < atm->np; ip++)
00295         if (gsl_finite(dt[ip]))
00296           module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00297                                 rng[omp_get_thread_num()]);
00298       STOP_TIMER(TIMER_DIFFMESO);
00299
00300       /* Sedimentation... */
00301       START_TIMER(TIMER_SEDI);
00302 #pragma omp parallel for default(shared) private(ip)
00303       for (ip = 0; ip < atm->np; ip++)
00304         if (gsl_finite(dt[ip]))
00305           module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00306       STOP_TIMER(TIMER_SEDI);
00307
00308       /* Isosurface... */
00309       START_TIMER(TIMER_ISOSURF);
00310 #pragma omp parallel for default(shared) private(ip)
```

```
00311       for (ip = 0; ip < atm->np; ip++)
00312         module_isosurf(&ctl, met0, met1, atm, ip);
00313       STOP_TIMER(TIMER_ISOSURF);
00314
00315       /* Position... */
00316       START_TIMER(TIMER_POSITION);
00317 #pragma omp parallel for default(shared) private(ip)
00318       for (ip = 0; ip < atm->np; ip++)
00319         module_position(met0, met1, atm, ip);
00320       STOP_TIMER(TIMER_POSITION);
00321
00322       /* Meteorological data... */
00323       START_TIMER(TIMER_METEO);
00324 #pragma omp parallel for default(shared) private(ip)
00325       for (ip = 0; ip < atm->np; ip++)
00326         module_meteo(&ctl, met0, met1, atm, ip);
00327       STOP_TIMER(TIMER_METEO);
00328
00329       /* Decay... */
00330       START_TIMER(TIMER_DECAY);
00331 #pragma omp parallel for default(shared) private(ip)
00332       for (ip = 0; ip < atm->np; ip++)
00333         if (gsl_finite(dt[ip]))
00334           module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00335       STOP_TIMER(TIMER_DECAY);
00336
00337       /* Write output... */
00338       START_TIMER(TIMER_OUTPUT);
00339       write_output(dirname, &ctl, met0, met1, atm, t);
00340       STOP_TIMER(TIMER_OUTPUT);
00341     }
00342
00343     /* ------------------------------------------------------------
00344        Finalize model run...
00345        ------------------------------------------------------------ */
00346
00347     /* Report timers... */
00348     STOP_TIMER(TIMER_TOTAL);
00349     PRINT_TIMER(TIMER_TOTAL);
00350     PRINT_TIMER(TIMER_INIT);
00351     PRINT_TIMER(TIMER_INPUT);
00352     PRINT_TIMER(TIMER_OUTPUT);
00353     PRINT_TIMER(TIMER_ADVECT);
00354     PRINT_TIMER(TIMER_DECAY);
00355     PRINT_TIMER(TIMER_DIFFMESO);
00356     PRINT_TIMER(TIMER_DIFFTURB);
00357     PRINT_TIMER(TIMER_ISOSURF);
00358     PRINT_TIMER(TIMER_METEO);
00359     PRINT_TIMER(TIMER_POSITION);
00360     PRINT_TIMER(TIMER_SEDI);
00361
00362     /* Report memory usage... */
00363     printf("MEMORY_ATM = %g MByte\n", 2. * sizeof(atm_t) / 1024. / 1024.);
00364     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00365     printf("MEMORY_DYNAMIC = %g MByte\n",
00366            NP * sizeof(double) / 1024. / 1024.);
00367     printf("MEMORY_STATIC = %g MByte\n",
00368            (((EX + EY) + (2 + NQ) * GX * GY * GZ) * sizeof(double)
00369             + (EX * EY + EX * EY * EP) * sizeof(float)
00370             + (2 * GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00371
00372     /* Report problem size... */
00373     printf("SIZE_NP = %d\n", atm->np);
00374     printf("SIZE_TASKS = %d\n", size);
00375     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00376
00377     /* Free random number generators... */
00378     for (i = 0; i < NTHREADS; i++)
00379       gsl_rng_free(rng[i]);
00380
00381     /* Free... */
00382     free(atm);
00383     free(met0);
00384     free(met1);
00385     free(dt);
00386   }
00387
00388 #ifdef MPI
00389   /* Finalize MPI... */
00390   MPI_Finalize();
00391 #endif
00392
00393   return EXIT_SUCCESS;
00394 }
00395
00396 /*****************************************************************************/
00397
```

```
00398 void init_simtime(
00399   ctl_t * ctl,
00400   atm_t * atm) {
00401
00402   /* Set inital and final time... */
00403   if (ctl->direction == 1) {
00404     if (ctl->t_start < -1e99)
00405       ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00406     if (ctl->t_stop < -1e99)
00407       ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00408   } else if (ctl->direction == -1) {
00409     if (ctl->t_stop < -1e99)
00410       ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00411     if (ctl->t_start < -1e99)
00412       ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00413   }
00414
00415   /* Check time... */
00416   if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
00417     ERRMSG("Nothing to do!");
00418 }
00419
00420 /*****************************************************************************/
00421
00422 void module_advection(
00423   met_t * met0,
00424   met_t * met1,
00425   atm_t * atm,
00426   int ip,
00427   double dt) {
00428
00429   double v[3], xm[3];
00430
00431   /* Interpolate meteorological data... */
00432   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00433                   atm->lon[ip], atm->lat[ip], NULL, NULL,
00434                   &v[0], &v[1], &v[2], NULL, NULL);
00435
00436   /* Get position of the mid point... */
00437   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00438   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00439   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00440
00441   /* Interpolate meteorological data for mid point... */
00442   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00443                   xm[2], xm[0], xm[1], NULL, NULL,
00444                   &v[0], &v[1], &v[2], NULL, NULL);
00445
00446   /* Save new position... */
00447   atm->time[ip] += dt;
00448   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00449   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00450   atm->p[ip] += dt * v[2];
00451 }
00452
00453 /*****************************************************************************/
00454
00455 void module_decay(
00456   ctl_t * ctl,
00457   met_t * met0,
00458   met_t * met1,
00459   atm_t * atm,
00460   int ip,
00461   double dt) {
00462
00463   double ps, pt, tdec;
00464
00465   /* Check lifetime values... */
00466   if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
      qnt_m < 0)
00467     return;
00468
00469   /* Set constant lifetime... */
00470   if (ctl->tdec_trop == ctl->tdec_strat)
00471     tdec = ctl->tdec_trop;
00472
00473   /* Set altitude-dependent lifetime... */
00474   else {
00475
00476     /* Get surface pressure... */
00477     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00478                     atm->lon[ip], atm->lat[ip], &ps, NULL,
00479                     NULL, NULL, NULL, NULL, NULL);
00480
00481     /* Get tropopause pressure... */
00482     pt = tropopause(atm->time[ip], atm->lat[ip]);
00483
```

```
00484     /* Set lifetime... */
00485     if (atm->p[ip] <= pt)
00486       tdec = ctl->tdec_strat;
00487     else
00488       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00489   }
00490
00491   /* Calculate exponential decay... */
00492   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00493 }
00494
00495 /*****************************************************************************/
00496
00497 void module_diffusion_meso(
00498   ctl_t * ctl,
00499   met_t * met0,
00500   met_t * met1,
00501   atm_t * atm,
00502   int ip,
00503   double dt,
00504   gsl_rng * rng) {
00505
00506   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00507
00508   int ix, iy, iz;
00509
00510   /* Calculate mesoscale velocity fluctuations... */
00511   if (ctl->turb_meso > 0) {
00512
00513     /* Get indices... */
00514     ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00515     iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00516     iz = locate(met0->p, met0->np, atm->p[ip]);
00517
00518     /* Collect local wind data... */
00519     u[0] = met0->u[ix][iy][iz];
00520     u[1] = met0->u[ix + 1][iy][iz];
00521     u[2] = met0->u[ix][iy + 1][iz];
00522     u[3] = met0->u[ix + 1][iy + 1][iz];
00523     u[4] = met0->u[ix][iy][iz + 1];
00524     u[5] = met0->u[ix + 1][iy][iz + 1];
00525     u[6] = met0->u[ix][iy + 1][iz + 1];
00526     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00527
00528     v[0] = met0->v[ix][iy][iz];
00529     v[1] = met0->v[ix + 1][iy][iz];
00530     v[2] = met0->v[ix][iy + 1][iz];
00531     v[3] = met0->v[ix + 1][iy + 1][iz];
00532     v[4] = met0->v[ix][iy][iz + 1];
00533     v[5] = met0->v[ix + 1][iy][iz + 1];
00534     v[6] = met0->v[ix][iy + 1][iz + 1];
00535     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00536
00537     w[0] = met0->w[ix][iy][iz];
00538     w[1] = met0->w[ix + 1][iy][iz];
00539     w[2] = met0->w[ix][iy + 1][iz];
00540     w[3] = met0->w[ix + 1][iy + 1][iz];
00541     w[4] = met0->w[ix][iy][iz + 1];
00542     w[5] = met0->w[ix + 1][iy][iz + 1];
00543     w[6] = met0->w[ix][iy + 1][iz + 1];
00544     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00545
00546     /* Get indices... */
00547     ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00548     iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00549     iz = locate(met1->p, met1->np, atm->p[ip]);
00550
00551     /* Collect local wind data... */
00552     u[8] = met1->u[ix][iy][iz];
00553     u[9] = met1->u[ix + 1][iy][iz];
00554     u[10] = met1->u[ix][iy + 1][iz];
00555     u[11] = met1->u[ix + 1][iy + 1][iz];
00556     u[12] = met1->u[ix][iy][iz + 1];
00557     u[13] = met1->u[ix + 1][iy][iz + 1];
00558     u[14] = met1->u[ix][iy + 1][iz + 1];
00559     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00560
00561     v[8] = met1->v[ix][iy][iz];
00562     v[9] = met1->v[ix + 1][iy][iz];
00563     v[10] = met1->v[ix][iy + 1][iz];
00564     v[11] = met1->v[ix + 1][iy + 1][iz];
00565     v[12] = met1->v[ix][iy][iz + 1];
00566     v[13] = met1->v[ix + 1][iy][iz + 1];
00567     v[14] = met1->v[ix][iy + 1][iz + 1];
00568     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00569
```

```
00570     w[8] = met1->w[ix][iy][iz];
00571     w[9] = met1->w[ix + 1][iy][iz];
00572     w[10] = met1->w[ix][iy + 1][iz];
00573     w[11] = met1->w[ix + 1][iy + 1][iz];
00574     w[12] = met1->w[ix][iy][iz + 1];
00575     w[13] = met1->w[ix + 1][iy][iz + 1];
00576     w[14] = met1->w[ix][iy + 1][iz + 1];
00577     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00578
00579     /* Get standard deviations of local wind data... */
00580     usig = gsl_stats_sd(u, 1, 16);
00581     vsig = gsl_stats_sd(v, 1, 16);
00582     wsig = gsl_stats_sd(w, 1, 16);
00583
00584     /* Set temporal correlations for mesoscale fluctuations... */
00585     r = 1 - 2 * fabs(dt) / ctl->dt_met;
00586     rs = sqrt(1 - r * r);
00587
00588     /* Calculate mesoscale wind fluctuations... */
00589     atm->up[ip] =
00590       r * atm->up[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00591                                                        ctl->turb_meso * usig);
00592     atm->vp[ip] =
00593       r * atm->vp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00594                                                        ctl->turb_meso * vsig);
00595     atm->wp[ip] =
00596       r * atm->wp[ip] + rs * gsl_ran_gaussian_ziggurat(rng,
00597                                                        ctl->turb_meso * wsig);
00598
00599     /* Calculate air parcel displacement... */
00600     atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00601     atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00602     atm->p[ip] += atm->wp[ip] * dt;
00603   }
00604 }
00605
00606 /*****************************************************************************/
00607
00608 void module_diffusion_turb(
00609   ctl_t * ctl,
00610   atm_t * atm,
00611   int ip,
00612   double dt,
00613   gsl_rng * rng) {
00614
00615   double dx, dz, pt, p0, p1, w;
00616
00617   /* Get tropopause pressure... */
00618   pt = tropopause(atm->time[ip], atm->lat[ip]);
00619
00620   /* Get weighting factor... */
00621   p1 = pt * 0.866877899;
00622   p0 = pt / 0.866877899;
00623   if (atm->p[ip] > p0)
00624     w = 1;
00625   else if (atm->p[ip] < p1)
00626     w = 0;
00627   else
00628     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00629
00630   /* Set diffusivitiy... */
00631   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00632   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00633
00634   /* Horizontal turbulent diffusion... */
00635   if (dx > 0) {
00636     atm->lon[ip]
00637       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00638                 / 1000., atm->lat[ip]);
00639     atm->lat[ip]
00640       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00641                 / 1000.);
00642   }
00643
00644   /* Vertical turbulent diffusion... */
00645   if (dz > 0)
00646     atm->p[ip]
00647       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00648                / 1000., atm->p[ip]);
00649 }
00650
00651 /*****************************************************************************/
00652
00653 void module_isosurf(
00654   ctl_t * ctl,
00655   met_t * met0,
00656   met_t * met1,
```

```
00657   atm_t * atm,
00658   int ip) {
00659
00660   static double *iso, *ps, t, *ts;
00661
00662   static int idx, ip2, n, nb = 100000;
00663
00664   FILE *in;
00665
00666   char line[LEN];
00667
00668   /* Check control parameter... */
00669   if (ctl->isosurf < 1 || ctl->isosurf > 4)
00670     return;
00671
00672   /* Initialize... */
00673   if (ip < 0) {
00674
00675     /* Allocate... */
00676     ALLOC(iso, double,
00677           NP);
00678     ALLOC(ps, double,
00679           nb);
00680     ALLOC(ts, double,
00681           nb);
00682
00683     /* Save pressure... */
00684     if (ctl->isosurf == 1)
00685       for (ip2 = 0; ip2 < atm->np; ip2++)
00686         iso[ip2] = atm->p[ip2];
00687
00688     /* Save density... */
00689     else if (ctl->isosurf == 2)
00690       for (ip2 = 0; ip2 < atm->np; ip2++) {
00691         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00692                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00693                         NULL, NULL, NULL);
00694         iso[ip2] = atm->p[ip2] / t;
00695       }
00696
00697     /* Save potential temperature... */
00698     else if (ctl->isosurf == 3)
00699       for (ip2 = 0; ip2 < atm->np; ip2++) {
00700         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00701                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00702                         NULL, NULL, NULL);
00703         iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00704       }
00705
00706     /* Read balloon pressure data... */
00707     else if (ctl->isosurf == 4) {
00708
00709       /* Write info... */
00710       printf("Read balloon pressure data: %s\n", ctl->balloon);
00711
00712       /* Open file... */
00713       if (!(in = fopen(ctl->balloon, "r")))
00714         ERRMSG("Cannot open file!");
00715
00716       /* Read pressure time series... */
00717       while (fgets(line, LEN, in))
00718         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00719           if ((++n) > 100000)
00720             ERRMSG("Too many data points!");
00721
00722       /* Check number of points... */
00723       if (n < 1)
00724         ERRMSG("Could not read any data!");
00725
00726       /* Close file... */
00727       fclose(in);
00728     }
00729
00730     /* Leave initialization... */
00731     return;
00732   }
00733
00734   /* Restore pressure... */
00735   if (ctl->isosurf == 1)
00736     atm->p[ip] = iso[ip];
00737
00738   /* Restore density... */
00739   else if (ctl->isosurf == 2) {
00740     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00741                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00742     atm->p[ip] = iso[ip] * t;
```

```
00743   }
00744
00745   /* Restore potential temperature... */
00746   else if (ctl->isosurf == 3) {
00747     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00748                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00749     atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00750   }
00751
00752   /* Interpolate pressure... */
00753   else if (ctl->isosurf == 4) {
00754     if (atm->time[ip] <= ts[0])
00755       atm->p[ip] = ps[0];
00756     else if (atm->time[ip] >= ts[n - 1])
00757       atm->p[ip] = ps[n - 1];
00758     else {
00759       idx = locate(ts, n, atm->time[ip]);
00760       atm->p[ip] = LIN(ts[idx], ps[idx],
00761                        ts[idx + 1], ps[idx + 1], atm->time[ip]);
00762     }
00763   }
00764 }
00765
00766 /*****************************************************************************/
00767
00768 void module_meteo(
00769   ctl_t * ctl,
00770   met_t * met0,
00771   met_t * met1,
00772   atm_t * atm,
00773   int ip) {
00774
00775   double b, c, ps, p1, p_hno3, p_h2o, t, t1, term1, term2,
00776     u, u1, v, v1, w, x1, x2, h2o, o3, grad, vort;
00777
00778   /* Interpolate meteorological data... */
00779   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00780                   atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00781
00782   /* Set surface pressure... */
00783   if (ctl->qnt_ps >= 0)
00784     atm->q[ctl->qnt_ps][ip] = ps;
00785
00786   /* Set pressure... */
00787   if (ctl->qnt_p >= 0)
00788     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00789
00790   /* Set temperature... */
00791   if (ctl->qnt_t >= 0)
00792     atm->q[ctl->qnt_t][ip] = t;
00793
00794   /* Set zonal wind... */
00795   if (ctl->qnt_u >= 0)
00796     atm->q[ctl->qnt_u][ip] = u;
00797
00798   /* Set meridional wind... */
00799   if (ctl->qnt_v >= 0)
00800     atm->q[ctl->qnt_v][ip] = v;
00801
00802   /* Set vertical velocity... */
00803   if (ctl->qnt_w >= 0)
00804     atm->q[ctl->qnt_w][ip] = w;
00805
00806   /* Set water vapor vmr... */
00807   if (ctl->qnt_h2o >= 0)
00808     atm->q[ctl->qnt_h2o][ip] = h2o;
00809
00810   /* Set ozone vmr... */
00811   if (ctl->qnt_o3 >= 0)
00812     atm->q[ctl->qnt_o3][ip] = o3;
00813
00814   /* Calculate potential temperature... */
00815   if (ctl->qnt_theta >= 0)
00816     atm->q[ctl->qnt_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00817
00818   /* Calculate potential vorticity... */
00819   if (ctl->qnt_pv >= 0) {
00820
00821     /* Absolute vorticity... */
00822     vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00823     if (fabs(atm->lat[ip]) < 89.) {
00824       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00825                       (atm->lon[ip] >=
00826                        0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00827                       atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
```

```
00828       vort += (v1 - v) / 1000.
00829         / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00830     }
00831     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
00832                     (atm->lat[ip] >=
00833                      0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
00834                     &u1, NULL, NULL, NULL, NULL);
00835     vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00836
00837     /* Potential temperature gradient... */
00838     p1 = 0.85 * atm->p[ip];
00839     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
00840                     atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL, NULL);
00841     grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00842       / (100. * (p1 - atm->p[ip]));
00843
00844     /* Calculate PV... */
00845     atm->q[ctl->qnt_pv][ip] = -1e6 * G0 * vort * grad;
00846   }
00847
00848   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00849   if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00850     atm->q[ctl->qnt_tice][ip] = -2663.5
00851       / (log10(ctl->psc_h2o * atm->p[ip] * 100.) - 12.537);
00852
00853   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00854   if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
00855     p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00856     p_h2o = ctl->psc_h2o * atm->p[ip] / 1.333224;
00857     term1 = 38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o);
00858     term2 = 0.009179 - 0.00088 * log10(p_h2o);
00859     b = term1 / term2;
00860     c = -11397.0 / term2;
00861     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00862     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00863     if (x1 > 0)
00864       atm->q[ctl->qnt_tnat][ip] = x1;
00865     if (x2 > 0)
00866       atm->q[ctl->qnt_tnat][ip] = x2;
00867   }
00868
00869   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00870   if (ctl->qnt_tsts >= 0) {
00871     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00872       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00873     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00874                                        + atm->q[ctl->qnt_tnat][ip]);
00875   }
00876 }
00877
00878 /*****************************************************************************/
00879
00880 void module_position(
00881   met_t * met0,
00882   met_t * met1,
00883   atm_t * atm,
00884   int ip) {
00885
00886   double ps;
00887
00888   /* Calculate modulo... */
00889   atm->lon[ip] = fmod(atm->lon[ip], 360);
00890   atm->lat[ip] = fmod(atm->lat[ip], 360);
00891
00892   /* Check latitude... */
00893   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
00894     if (atm->lat[ip] > 90) {
00895       atm->lat[ip] = 180 - atm->lat[ip];
00896       atm->lon[ip] += 180;
00897     }
00898     if (atm->lat[ip] < -90) {
00899       atm->lat[ip] = -180 - atm->lat[ip];
00900       atm->lon[ip] += 180;
00901     }
00902   }
00903
00904   /* Check longitude... */
00905   while (atm->lon[ip] < -180)
00906     atm->lon[ip] += 360;
00907   while (atm->lon[ip] >= 180)
00908     atm->lon[ip] -= 360;
00909
00910   /* Get surface pressure... */
00911   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00912                   atm->lon[ip], atm->lat[ip], &ps, NULL,
00913                   NULL, NULL, NULL, NULL, NULL);
```

```
00914
00915   /* Check pressure... */
00916   if (atm->p[ip] > ps)
00917     atm->p[ip] = ps;
00918   else if (atm->p[ip] < met0->p[met0->np - 1])
00919     atm->p[ip] = met0->p[met0->np - 1];
00920 }
00921
00922 /*****************************************************************************/
00923
00924 void module_sedi(
00925   ctl_t * ctl,
00926   met_t * met0,
00927   met_t * met1,
00928   atm_t * atm,
00929   int ip,
00930   double dt) {
00931
00932   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
00933   const double A = 1.249, B = 0.42, C = 0.87;
00934
00935   /* Specific gas constant for dry air [J/(kg K)]: */
00936   const double R = 287.058;
00937
00938   /* Average mass of an air molecule [kg/molec]: */
00939   const double m = 4.8096e-26;
00940
00941   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
00942
00943   /* Check if parameters are available... */
00944   if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
00945     return;
00946
00947   /* Convert units... */
00948   p = 100 * atm->p[ip];
00949   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
00950   rho_p = atm->q[ctl->qnt_rho][ip];
00951
00952   /* Get temperature... */
00953   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00954                   atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
00955
00956   /* Density of dry air... */
00957   rho = p / (R * T);
00958
00959   /* Dynamic viscosity of air... */
00960   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
00961
00962   /* Thermal velocity of an air molecule... */
00963   v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
00964
00965   /* Mean free path of an air molecule... */
00966   lambda = 2 * eta / (rho * v);
00967
00968   /* Knudsen number for air... */
00969   K = lambda / r_p;
00970
00971   /* Cunningham slip-flow correction... */
00972   G = 1 + K * (A + B * exp(-C / K));
00973
00974   /* Sedimentation (fall) velocity... */
00975   v_p =
00976     2. * gsl_pow_2(r_p) * (rho_p -
00977                            rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
00978
00979   /* Calculate pressure change... */
00980   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
00981 }
00982
00983 /*****************************************************************************/
00984
00985 void write_output(
00986   const char *dirname,
00987   ctl_t * ctl,
00988   met_t * met0,
00989   met_t * met1,
00990   atm_t * atm,
00991   double t) {
00992
00993   char filename[LEN];
00994
00995   double r;
00996
00997   int year, mon, day, hour, min, sec;
00998
00999   /* Get time... */
```

```
01000   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01001
01002   /* Write atmospheric data... */
01003   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01004     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01005             dirname, ctl->atm_basename, year, mon, day, hour, min);
01006     write_atm(filename, ctl, atm, t);
01007   }
01008
01009   /* Write CSI data... */
01010   if (ctl->csi_basename[0] != '-') {
01011     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01012     write_csi(filename, ctl, atm, t);
01013   }
01014
01015   /* Write ensemble data... */
01016   if (ctl->ens_basename[0] != '-') {
01017     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01018     write_ens(filename, ctl, atm, t);
01019   }
01020
01021   /* Write gridded data... */
01022   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01023     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01024             dirname, ctl->grid_basename, year, mon, day, hour, min);
01025     write_grid(filename, ctl, met0, met1, atm, t);
01026   }
01027
01028   /* Write profile data... */
01029   if (ctl->prof_basename[0] != '-') {
01030     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01031     write_prof(filename, ctl, met0, met1, atm, t);
01032   }
01033
01034   /* Write station data... */
01035   if (ctl->stat_basename[0] != '-') {
01036     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01037     write_station(filename, ctl, atm, t);
01038   }
01039 }
```

5.33 wind.c File Reference

Create meteorological data files with synthetic wind fields.

Functions

- void add_text_attribute (int ncid, char *varname, char *attrname, char *text)
- int main (int argc, char *argv[ ])

5.33.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

5.33.2 Function Documentation

5.33.2.1 void add_text_attribute (int *ncid*, char * *varname*, char * *attrname*, char * *text*)

Definition at line 173 of file wind.c.

```
00177               {
00178
00179   int varid;
00180
00181   NC(nc_inq_varid(ncid, varname, &varid));
00182   NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00183 }
```

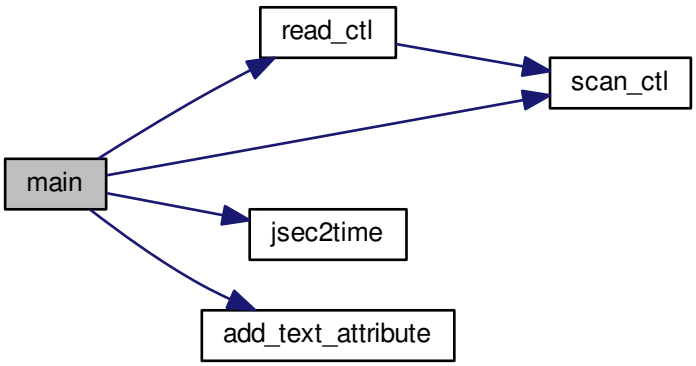
5.33.2.2 int main (int *argc*, char * *argv[]*)

Definition at line 41 of file wind.c.

```
00043                {
00044
00045   ctl_t ctl;
00046
00047   static char filename[LEN];
00048
00049   static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050     u0, u1, alpha;
00051
00052   static float dataT[EP * EY * EX], dataU[EP * EY * EX], dataV[EP * EY * EX],
00053     dataW[EP * EY * EX];
00054
00055   static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00056     idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00057
00058   /* Check arguments... */
00059   if (argc < 3)
00060     ERRMSG("Give parameters: <ctl> <metbase>");
00061
00062   /* Read control parameters... */
00063   read_ctl(argv[1], argc, argv, &ctl);
00064   t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00065   nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00066   ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00067   nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00068   z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00069   z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00070   u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00071   u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00072   alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00073
00074   /* Check dimensions... */
00075   if (nx < 1 || nx > EX)
00076     ERRMSG("Set 1 <= NX <= MAX!");
00077   if (ny < 1 || ny > EY)
00078     ERRMSG("Set 1 <= NY <= MAX!");
00079   if (nz < 1 || nz > EP)
00080     ERRMSG("Set 1 <= NZ <= MAX!");
00081
00082   /* Get time... */
00083   jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00084   t0 = year * 10000. + mon * 100. + day + hour / 24.;
00085
00086   /* Set filename... */
00087   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00088
00089   /* Create netCDF file... */
00090   NC(nc_create(filename, NC_CLOBBER, &ncid));
00091
00092   /* Create dimensions... */
00093   NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00094   NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00095   NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00096   NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00097
00098   /* Create variables... */
00099   NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00100   NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
00101   NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00102   NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00103   NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
00104   NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
00105   NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00106   NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00107
00108   /* Set attributes... */
00109   add_text_attribute(ncid, "time", "long_name", "time");
00110   add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
00111   add_text_attribute(ncid, "lon", "long_name", "longitude");
00112   add_text_attribute(ncid, "lon", "units", "degrees_east");
00113   add_text_attribute(ncid, "lat", "long_name", "latitude");
00114   add_text_attribute(ncid, "lat", "units", "degrees_north");
00115   add_text_attribute(ncid, "lev", "long_name", "air_pressure");
00116   add_text_attribute(ncid, "lev", "units", "Pa");
00117   add_text_attribute(ncid, "T", "long_name", "Temperature");
00118   add_text_attribute(ncid, "T", "units", "K");
00119   add_text_attribute(ncid, "U", "long_name", "U velocity");
00120   add_text_attribute(ncid, "U", "units", "m s**-1");
00121   add_text_attribute(ncid, "V", "long_name", "V velocity");
00122   add_text_attribute(ncid, "V", "units", "m s**-1");
```

```
00123   add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00124   add_text_attribute(ncid, "W", "units", "Pa s**-1");
00125
00126   /* End definition... */
00127   NC(nc_enddef(ncid));
00128
00129   /* Set coordinates... */
00130   for (ix = 0; ix < nx; ix++)
00131     dataLon[ix] = 360.0 / nx * (double) ix;
00132   for (iy = 0; iy < ny; iy++)
00133     dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00134   for (iz = 0; iz < nz; iz++)
00135     dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00136
00137   /* Write coordinates... */
00138   NC(nc_put_var_double(ncid, timid, &t0));
00139   NC(nc_put_var_double(ncid, levid, dataZ));
00140   NC(nc_put_var_double(ncid, lonid, dataLon));
00141   NC(nc_put_var_double(ncid, latid, dataLat));
00142
00143   /* Create wind fields (Williamson et al., 1992)... */
00144   for (ix = 0; ix < nx; ix++)
00145     for (iy = 0; iy < ny; iy++)
00146       for (iz = 0; iz < nz; iz++) {
00147         idx = (iz * ny + iy) * nx + ix;
00148         dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00149                               * (cos(dataLat[iy] * M_PI / 180.0)
00150                                  * cos(alpha * M_PI / 180.0)
00151                                  + sin(dataLat[iy] * M_PI / 180.0)
00152                                  * cos(dataLon[ix] * M_PI / 180.0)
00153                                  * sin(alpha * M_PI / 180.0)));
00154         dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00155                               * sin(dataLon[ix] * M_PI / 180.0)
00156                               * sin(alpha * M_PI / 180.0));
00157       }
00158
00159   /* Write wind data... */
00160   NC(nc_put_var_float(ncid, tid, dataT));
00161   NC(nc_put_var_float(ncid, uid, dataU));
00162   NC(nc_put_var_float(ncid, vid, dataV));
00163   NC(nc_put_var_float(ncid, wid, dataW));
00164
00165   /* Close file... */
00166   NC(nc_close(ncid));
00167
00168   return EXIT_SUCCESS;
00169 }
```

Here is the call graph for this function:



5.34 wind.c

```
00001 /*
```

```
00002   This file is part of MPTRAC.
00003
00004   MPTRAC is free software: you can redistribute it and/or modify
00005   it under the terms of the GNU General Public License as published by
00006   the Free Software Foundation, either version 3 of the License, or
00007   (at your option) any later version.
00008
00009   MPTRAC is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00012   GNU General Public License for more details.
00013
00014   You should have received a copy of the GNU General Public License
00015   along with MPTRAC. If not, see <http://www.gnu.org/licenses/>.
00016
00017   Copright (C) 2013-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "libtrac.h"
00026
00027 /* ------------------------------------------------------------
00028    Functions...
00029    ------------------------------------------------------------ */
00030
00031 void add_text_attribute(
00032   int ncid,
00033   char *varname,
00034   char *attrname,
00035   char *text);
00036
00037 /* ------------------------------------------------------------
00038    Main...
00039    ------------------------------------------------------------ */
00040
00041 int main(
00042   int argc,
00043   char *argv[]) {
00044
00045   ctl_t ctl;
00046
00047   static char filename[LEN];
00048
00049   static double r, t0, z0, z1, dataLon[EX], dataLat[EY], dataZ[EP],
00050     u0, u1, alpha;
00051
00052   static float dataT[EP * EY * EX], dataU[EP * EY * EX], dataV[EP * EY * EX],
00053     dataW[EP * EY * EX];
00054
00055   static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00056     idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00057
00058   /* Check arguments... */
00059   if (argc < 3)
00060     ERRMSG("Give parameters: <ctl> <metbase>");
00061
00062   /* Read control parameters... */
00063   read_ctl(argv[1], argc, argv, &ctl);
00064   t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00065   nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00066   ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00067   nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00068   z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00069   z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00070   u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00071   u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00072   alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00073
00074   /* Check dimensions... */
00075   if (nx < 1 || nx > EX)
00076     ERRMSG("Set 1 <= NX <= MAX!");
00077   if (ny < 1 || ny > EY)
00078     ERRMSG("Set 1 <= NY <= MAX!");
00079   if (nz < 1 || nz > EP)
00080     ERRMSG("Set 1 <= NZ <= MAX!");
00081
00082   /* Get time... */
00083   jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00084   t0 = year * 10000. + mon * 100. + day + hour / 24.;
00085
00086   /* Set filename... */
00087   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00088
00089   /* Create netCDF file... */
00090   NC(nc_create(filename, NC_CLOBBER, &ncid));
00091
00092   /* Create dimensions... */
00093   NC(nc_def_dim(ncid, "time", 1, &dims[0]));
```

```
00094   NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00095   NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00096   NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00097
00098   /* Create variables... */
00099   NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00100   NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
00101   NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00102   NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00103   NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
00104   NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
00105   NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00106   NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00107
00108   /* Set attributes... */
00109   add_text_attribute(ncid, "time", "long_name", "time");
00110   add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
00111   add_text_attribute(ncid, "lon", "long_name", "longitude");
00112   add_text_attribute(ncid, "lon", "units", "degrees_east");
00113   add_text_attribute(ncid, "lat", "long_name", "latitude");
00114   add_text_attribute(ncid, "lat", "units", "degrees_north");
00115   add_text_attribute(ncid, "lev", "long_name", "air_pressure");
00116   add_text_attribute(ncid, "lev", "units", "Pa");
00117   add_text_attribute(ncid, "T", "long_name", "Temperature");
00118   add_text_attribute(ncid, "T", "units", "K");
00119   add_text_attribute(ncid, "U", "long_name", "U velocity");
00120   add_text_attribute(ncid, "U", "units", "m s**-1");
00121   add_text_attribute(ncid, "V", "long_name", "V velocity");
00122   add_text_attribute(ncid, "V", "units", "m s**-1");
00123   add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00124   add_text_attribute(ncid, "W", "units", "Pa s**-1");
00125
00126   /* End definition... */
00127   NC(nc_enddef(ncid));
00128
00129   /* Set coordinates... */
00130   for (ix = 0; ix < nx; ix++)
00131     dataLon[ix] = 360.0 / nx * (double) ix;
00132   for (iy = 0; iy < ny; iy++)
00133     dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00134   for (iz = 0; iz < nz; iz++)
00135     dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00136
00137   /* Write coordinates... */
00138   NC(nc_put_var_double(ncid, timid, &t0));
00139   NC(nc_put_var_double(ncid, levid, dataZ));
00140   NC(nc_put_var_double(ncid, lonid, dataLon));
00141   NC(nc_put_var_double(ncid, latid, dataLat));
00142
00143   /* Create wind fields (Williamson et al., 1992)... */
00144   for (ix = 0; ix < nx; ix++)
00145     for (iy = 0; iy < ny; iy++)
00146       for (iz = 0; iz < nz; iz++) {
00147         idx = (iz * ny + iy) * nx + ix;
00148         dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00149                               * (cos(dataLat[iy] * M_PI / 180.0)
00150                                  * cos(alpha * M_PI / 180.0)
00151                                  + sin(dataLat[iy] * M_PI / 180.0)
00152                                  * cos(dataLon[ix] * M_PI / 180.0)
00153                                  * sin(alpha * M_PI / 180.0)));
00154         dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00155                               * sin(dataLon[ix] * M_PI / 180.0)
00156                               * sin(alpha * M_PI / 180.0));
00157       }
00158
00159   /* Write wind data... */
00160   NC(nc_put_var_float(ncid, tid, dataT));
00161   NC(nc_put_var_float(ncid, uid, dataU));
00162   NC(nc_put_var_float(ncid, vid, dataV));
00163   NC(nc_put_var_float(ncid, wid, dataW));
00164
00165   /* Close file... */
00166   NC(nc_close(ncid));
00167
00168   return EXIT_SUCCESS;
00169 }
00170
00171 /*****************************************************************************/
00172
00173 void add_text_attribute(
00174   int ncid,
00175   char *varname,
00176   char *attrname,
00177   char *text) {
00178
00179   int varid;
00180
```

```
00181   NC(nc_inq_varid(ncid, varname, &varid));
00182   NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00183 }
```

Index