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).

- float up [NP]

  Zonal wind perturbation [m/s].

- float vp [NP]

  Meridional wind perturbation [m/s].

- float wp [NP]

  Vertical velocity perturbation [hPa/s].

4.1.1 Detailed Description

Atmospheric data.

Definition at line 520 of file libtrac.h.

4.1.2 Field Documentation

4.1.2.1 int atm_t::np

Number of air pacels.

Definition at line 523 of file libtrac.h.

4.1.2.2 double atm_t::time[NP]

Time [s].

Definition at line 526 of file libtrac.h.

4.1.2.3 double atm_t::p[NP]

Pressure [hPa].

Definition at line 529 of file libtrac.h.

4.1.2.4 double atm_t::lon[NP]

Longitude [deg].

Definition at line 532 of file libtrac.h.

4.1.2.5 double atm_t::lat[NP]

Latitude [deg].

Definition at line 535 of file libtrac.h.

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

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

Definition at line 538 of file libtrac.h.

4.1.2.7 float atm_t::up[NP]

Zonal wind perturbation [m/s].

Definition at line 541 of file libtrac.h.

4.1.2.8 float atm_t::vp[NP]

Meridional wind perturbation [m/s].

Definition at line 544 of file libtrac.h.

4.1.2.9 float atm_t::wp[NP]

Vertical velocity perturbation [hPa/s].

Definition at line 547 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 qnt_gw_u750
  Quantity array index for low-level zonal wind.
- int qnt_gw_v750
  Quantity array index for low-level meridional wind.
- int qnt_gw_sso
  Quantity array index for subgrid-scale orography.
- int qnt_gw_var
  Quantity array index for gravity wave variances.
- 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].
- char met_stage [LEN]
  Command to stage meteo data.
- 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 gw_basename [LEN]

  Basename for gravity wave variance data.
- 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).
- int atm_bin

  Binary I/O of atmospheric data (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].

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 232 of file libtrac.h.

4.2.2 Field Documentation

4.2.2.1 int ctl_t::nq

Number of quantities.

Definition at line 235 of file libtrac.h.

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

Quantity names.

Definition at line 238 of file libtrac.h.

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

Quantity units.

Definition at line 241 of file libtrac.h.

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

Quantity output format.

Definition at line 244 of file libtrac.h.

4.2.2.5 int ctl_t::qnt_ens

Quantity array index for ensemble IDs.

Definition at line 247 of file libtrac.h.

4.2.2.6 int ctl_t::qnt_m

Quantity array index for mass.

Definition at line 250 of file libtrac.h.

4.2.2.7 int ctl_t::qnt_rho

Quantity array index for particle density.

Definition at line 253 of file libtrac.h.

4.2.2.8 int ctl_t::qnt_r

Quantity array index for particle radius.

Definition at line 256 of file libtrac.h.

4.2.2.9 int ctl_t::qnt_ps

Quantity array index for surface pressure.

Definition at line 259 of file libtrac.h.

4.2.2.10 int ctl_t::qnt_p

Quantity array index for pressure.

Definition at line 262 of file libtrac.h.

4.2.2.11 int ctl_t::qnt_t

Quantity array index for temperature.

Definition at line 265 of file libtrac.h.

4.2.2.12 int ctl_t::qnt_u

Quantity array index for zonal wind.

Definition at line 268 of file libtrac.h.

4.2.2.13 int ctl_t::qnt_v

Quantity array index for meridional wind.

Definition at line 271 of file libtrac.h.

4.2.2.14 int ctl_t::qnt_w

Quantity array index for vertical velocity.

Definition at line 274 of file libtrac.h.

4.2.2.15 int ctl_t::qnt_h2o

Quantity array index for water vapor vmr.

Definition at line 277 of file libtrac.h.

4.2.2.16 int ctl_t::qnt_o3

Quantity array index for ozone vmr.

Definition at line 280 of file libtrac.h.

4.2.2.17 int ctl_t::qnt_theta

Quantity array index for potential temperature.

Definition at line 283 of file libtrac.h.

4.2.2.18 int ctl_t::qnt_pv

Quantity array index for potential vorticity.

Definition at line 286 of file libtrac.h.

4.2.2.19 int ctl_t::qnt_tice

Quantity array index for T_ice.

Definition at line 289 of file libtrac.h.

4.2.2.20 int ctl_t::qnt_tsts

Quantity array index for T_STS.

Definition at line 292 of file libtrac.h.

4.2.2.21 int ctl_t::qnt_tnat

Quantity array index for T_NAT.

Definition at line 295 of file libtrac.h.

4.2.2.22 int ctl_t::qnt_stat

Quantity array index for station flag.

Definition at line 298 of file libtrac.h.

4.2.2.23 int ctl_t::qnt_gw_u750

Quantity array index for low-level zonal wind.

Definition at line 301 of file libtrac.h.

4.2.2.24 int ctl_t::qnt_gw_v750

Quantity array index for low-level meridional wind.

Definition at line 304 of file libtrac.h.

4.2.2.25 int ctl_t::qnt_gw_sso

Quantity array index for subgrid-scale orography.

Definition at line 307 of file libtrac.h.

4.2.2.26 int ctl_t::qnt_gw_var

Quantity array index for gravity wave variances.

Definition at line 310 of file libtrac.h.

4.2.2.27 int ctl_t::direction

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

Definition at line 313 of file libtrac.h.

4.2.2.28 double ctl_t::t_start

Start time of simulation [s].

Definition at line 316 of file libtrac.h.

4.2.2.29 double ctl_t::t_stop

Stop time of simulation [s].

Definition at line 319 of file libtrac.h.

4.2.2.30 double ctl_t::dt_mod

Time step of simulation [s].

Definition at line 322 of file libtrac.h.

4.2.2.31 double ctl_t::dt_met

Time step of meteorological data [s].

Definition at line 325 of file libtrac.h.

4.2.2.32 int ctl_t::met_np

Number of target pressure levels.

Definition at line 328 of file libtrac.h.

4.2.2.33 double ctl_t::met_p[EP]

Target pressure levels [hPa].

Definition at line 331 of file libtrac.h.

4.2.2.34 char ctl_t::met_stage[LEN]

Command to stage meteo data.

Definition at line 334 of file libtrac.h.

4.2.2.35 int ctl_t::isosurf

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

Definition at line 338 of file libtrac.h.

4.2.2.36 char ctl_t::balloon[LEN]

Balloon position filename.

Definition at line 341 of file libtrac.h.

4.2.2.37 double ctl_t::turb_dx_trop

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

Definition at line 344 of file libtrac.h.

4.2.2.38 double ctl_t::turb_dx_strat

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

Definition at line 347 of file libtrac.h.

4.2.2.39 double ctl_t::turb_dz_trop

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

Definition at line 350 of file libtrac.h.

4.2.2.40 double ctl_t::turb_dz_strat

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

Definition at line 353 of file libtrac.h.

4.2.2.41 double ctl_t::turb_meso

Scaling factor for mesoscale wind fluctuations.

Definition at line 356 of file libtrac.h.

4.2.2.42 double ctl_t::tdec_trop

Life time of particles (troposphere) [s].

Definition at line 359 of file libtrac.h.

4.2.2.43 double ctl_t::tdec_strat

Life time of particles (stratosphere) [s].

Definition at line 362 of file libtrac.h.

4.2.2.44 double ctl_t::psc_h2o

H2O volume mixing ratio for PSC analysis.

Definition at line 365 of file libtrac.h.

4.2.2.45 double ctl_t::psc_hno3

HNO3 volume mixing ratio for PSC analysis.

Definition at line 368 of file libtrac.h.

4.2.2.46 char ctl_t::gw_basename[LEN]

Basename for gravity wave variance data.

Definition at line 371 of file libtrac.h.

4.2.2.47 char ctl_t::atm_basename[LEN]

Basename of atmospheric data files.

Definition at line 374 of file libtrac.h.

4.2.2.48 char ctl_t::atm_gpfile[LEN]

Gnuplot file for atmospheric data.

Definition at line 377 of file libtrac.h.

4.2.2.49 double ctl_t::atm_dt_out

Time step for atmospheric data output [s].

Definition at line 380 of file libtrac.h.

4.2.2.50 int ctl_t::atm_filter

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

Definition at line 383 of file libtrac.h.

4.2.2.51 int ctl_t::atm_bin

Binary I/O of atmospheric data (0=no, 1=yes).

Definition at line 386 of file libtrac.h.

4.2.2.52 char ctl_t::csi_basename[LEN]

Basename of CSI data files.

Definition at line 389 of file libtrac.h.

4.2.2.53 double ctl_t::csi_dt_out

Time step for CSI data output [s].

Definition at line 392 of file libtrac.h.

4.2.2.54 char ctl_t::csi_obsfile[LEN]

Observation data file for CSI analysis.

Definition at line 395 of file libtrac.h.

4.2.2.55 double ctl_t::csi_obsmin

Minimum observation index to trigger detection.

Definition at line 398 of file libtrac.h.

4.2.2.56 double ctl_t::csi_modmin

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

Definition at line 401 of file libtrac.h.

4.2.2.57 int ctl_t::csi_nz

Number of altitudes of gridded CSI data.

Definition at line 404 of file libtrac.h.

4.2.2.58 double ctl_t::csi_z0

Lower altitude of gridded CSI data [km].

Definition at line 407 of file libtrac.h.

4.2.2.59 double ctl_t::csi_z1

Upper altitude of gridded CSI data [km].

Definition at line 410 of file libtrac.h.

4.2.2.60 int ctl_t::csi_nx

Number of longitudes of gridded CSI data.

Definition at line 413 of file libtrac.h.

4.2.2.61 double ctl_t::csi_lon0

Lower longitude of gridded CSI data [deg].

Definition at line 416 of file libtrac.h.

4.2.2.62 double ctl_t::csi_lon1

Upper longitude of gridded CSI data [deg].

Definition at line 419 of file libtrac.h.

4.2.2.63 int ctl_t::csi_ny

Number of latitudes of gridded CSI data.

Definition at line 422 of file libtrac.h.

4.2.2.64 double ctl_t::csi_lat0

Lower latitude of gridded CSI data [deg].

Definition at line 425 of file libtrac.h.

4.2.2.65 double ctl_t::csi_lat1

Upper latitude of gridded CSI data [deg].

Definition at line 428 of file libtrac.h.

4.2.2.66 char ctl_t::grid_basename[LEN]

Basename of grid data files.

Definition at line 431 of file libtrac.h.

4.2.2.67 char ctl_t::grid_gpfile[LEN]

Gnuplot file for gridded data.

Definition at line 434 of file libtrac.h.

4.2.2.68 double ctl_t::grid_dt_out

Time step for gridded data output [s].

Definition at line 437 of file libtrac.h.

4.2.2.69 int ctl_t::grid_sparse

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

Definition at line 440 of file libtrac.h.

4.2.2.70 int ctl_t::grid_nz

Number of altitudes of gridded data.

Definition at line 443 of file libtrac.h.

4.2.2.71 double ctl_t::grid_z0

Lower altitude of gridded data [km].

Definition at line 446 of file libtrac.h.

4.2.2.72 double ctl_t::grid_z1

Upper altitude of gridded data [km].

Definition at line 449 of file libtrac.h.

4.2.2.73 int ctl_t::grid_nx

Number of longitudes of gridded data.

Definition at line 452 of file libtrac.h.

4.2.2.74 double ctl_t::grid_lon0

Lower longitude of gridded data [deg].

Definition at line 455 of file libtrac.h.

4.2.2.75 double ctl_t::grid_lon1

Upper longitude of gridded data [deg].

Definition at line 458 of file libtrac.h.

4.2.2.76 int ctl_t::grid_ny

Number of latitudes of gridded data.

Definition at line 461 of file libtrac.h.

4.2.2.77 double ctl_t::grid_lat0

Lower latitude of gridded data [deg].

Definition at line 464 of file libtrac.h.

4.2.2.78 double ctl_t::grid_lat1

Upper latitude of gridded data [deg].

Definition at line 467 of file libtrac.h.

4.2.2.79 char ctl_t::prof_basename[LEN]

Basename for profile output file.

Definition at line 470 of file libtrac.h.

4.2.2.80 char ctl_t::prof_obsfile[LEN]

Observation data file for profile output.

Definition at line 473 of file libtrac.h.

4.2.2.81 int ctl_t::prof_nz

Number of altitudes of gridded profile data.

Definition at line 476 of file libtrac.h.

4.2.2.82 double ctl_t::prof_z0

Lower altitude of gridded profile data [km].

Definition at line 479 of file libtrac.h.

4.2.2.83 double ctl_t::prof_z1

Upper altitude of gridded profile data [km].

Definition at line 482 of file libtrac.h.

4.2.2.84 int ctl_t::prof_nx

Number of longitudes of gridded profile data.

Definition at line 485 of file libtrac.h.

4.2.2.85 double ctl_t::prof_lon0

Lower longitude of gridded profile data [deg].

Definition at line 488 of file libtrac.h.

4.2.2.86 double ctl_t::prof_lon1

Upper longitude of gridded profile data [deg].

Definition at line 491 of file libtrac.h.

4.2.2.87 int ctl_t::prof_ny

Number of latitudes of gridded profile data.

Definition at line 494 of file libtrac.h.

4.2.2.88 double ctl_t::prof_lat0

Lower latitude of gridded profile data [deg].

Definition at line 497 of file libtrac.h.

4.2.2.89 double ctl_t::prof_lat1

Upper latitude of gridded profile data [deg].

Definition at line 500 of file libtrac.h.

4.2.2.90 char ctl_t::ens_basename[LEN]

Basename of ensemble data file.

Definition at line 503 of file libtrac.h.

4.2.2.91 char ctl_t::stat_basename[LEN]

Basename of station data file.

Definition at line 506 of file libtrac.h.

4.2.2.92 double ctl_t::stat_lon

Longitude of station [deg].

Definition at line 509 of file libtrac.h.

4.2.2.93 double ctl_t::stat_lat

Latitude of station [deg].

Definition at line 512 of file libtrac.h.

4.2.2.94 double ctl_t::stat_r

Search radius around station [km].

Definition at line 515 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 552 of file libtrac.h.

4.3.2 Field Documentation

4.3.2.1 double met_t::time

Time [s].

Definition at line 555 of file libtrac.h.

4.3.2.2 int met_t::nx

Number of longitudes.

Definition at line 558 of file libtrac.h.

4.3.2.3 int met_t::ny

Number of latitudes.

Definition at line 561 of file libtrac.h.

4.3.2.4 int met_t::np

Number of pressure levels.

Definition at line 564 of file libtrac.h.

4.3.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 567 of file libtrac.h.

4.3.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 570 of file libtrac.h.

4.3.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 573 of file libtrac.h.

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

Surface pressure [hPa].

Definition at line 576 of file libtrac.h.

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

Pressure on model levels [hPa].

Definition at line 579 of file libtrac.h.

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

Temperature [K].

Definition at line 582 of file libtrac.h.

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

Zonal wind [m/s].

Definition at line 585 of file libtrac.h.

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

Meridional wind [m/s].

Definition at line 588 of file libtrac.h.

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

Vertical wind [hPa/s].

Definition at line 591 of file libtrac.h.

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

Water vapor volume mixing ratio [1].

Definition at line 594 of file libtrac.h.

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

Ozone volume mixing ratio [1].

Definition at line 597 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[ ]*)

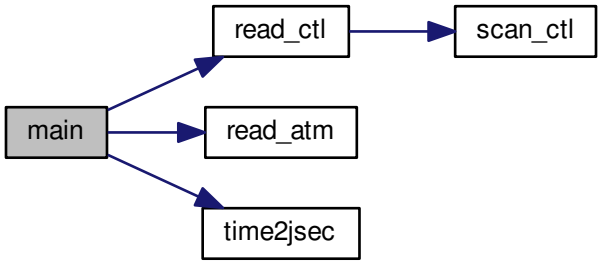
Definition at line 27 of file center.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double latm, lats, lonm, lons, t, zm, zs;
00040
00041   int f, ip, year, mon, day, hour, min;
00042
00043   /* Allocate... */
00044   ALLOC(atm, atm_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052
00053   /* Write info... */
00054   printf("Write center of mass data: %s\n", argv[2]);
00055
00056   /* Create output file... */
00057   if (!(out = fopen(argv[2], "w")))
00058     ERRMSG("Cannot create file!");
00059
00060   /* Write header... */
00061   fprintf(out,
00062           "# $1  = time [s]\n"
00063           "# $2  = altitude (mean) [km]\n"
00064           "# $3  = altitude (sigma) [km]\n"
00065           "# $4  = altitude (minimum) [km]\n"
00066           "# $5  = altitude (10%% percentile) [km]\n"
00067           "# $6  = altitude (1st quarter) [km]\n"
00068           "# $7  = altitude (median) [km]\n"
00069           "# $8  = altitude (3rd quarter) [km]\n"
00070           "# $9  = altitude (90%% percentile) [km]\n"
00071           "# $10 = altitude (maximum) [km]\n");
00072   fprintf(out,
```

```
00073           "# $11 = longitude (mean) [deg]\n"
00074           "# $12 = longitude (sigma) [deg]\n"
00075           "# $13 = longitude (minimum) [deg]\n"
00076           "# $14 = longitude (10%% percentile) [deg]\n"
00077           "# $15 = longitude (1st quarter) [deg]\n"
00078           "# $16 = longitude (median) [deg]\n"
00079           "# $17 = longitude (3rd quarter) [deg]\n"
00080           "# $18 = longitude (90%% percentile) [deg]\n"
00081           "# $19 = longitude (maximum) [deg]\n");
00082   fprintf(out,
00083           "# $20 = latitude (mean) [deg]\n"
00084           "# $21 = latitude (sigma) [deg]\n"
00085           "# $22 = latitude (minimum) [deg]\n"
00086           "# $23 = latitude (10%% percentile) [deg]\n"
00087           "# $24 = latitude (1st quarter) [deg]\n"
00088           "# $25 = latitude (median) [deg]\n"
00089           "# $26 = latitude (3rd quarter) [deg]\n"
00090           "# $27 = latitude (90%% percentile) [deg]\n"
00091           "# $28 = latitude (maximum) [deg]\n\n");
00092
00093   /* Loop over files... */
00094   for (f = 3; f < argc; f++) {
00095
00096     /* Read atmopheric data... */
00097     read_atm(argv[f], &ctl, atm);
00098
00099     /* Initialize... */
00100     zm = zs = 0;
00101     lonm = lons = 0;
00102     latm = lats = 0;
00103
00104     /* Calculate mean and standard deviation... */
00105     for (ip = 0; ip < atm->np; ip++) {
00106       zm += Z(atm->p[ip]) / atm->np;
00107       lonm += atm->lon[ip] / atm->np;
00108       latm += atm->lat[ip] / atm->np;
00109       zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
00110       lons += gsl_pow_2(atm->lon[ip]) / atm->np;
00111       lats += gsl_pow_2(atm->lat[ip]) / atm->np;
00112     }
00113
00114     /* Normalize... */
00115     zs = sqrt(zs - gsl_pow_2(zm));
00116     lons = sqrt(lons - gsl_pow_2(lonm));
00117     lats = sqrt(lats - gsl_pow_2(latm));
00118
00119     /* Sort arrays... */
00120     gsl_sort(atm->p, 1, (size_t) atm->np);
00121     gsl_sort(atm->lon, 1, (size_t) atm->np);
00122     gsl_sort(atm->lat, 1, (size_t) atm->np);
00123
00124     /* Get time from filename... */
00125     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00126     year = atoi(tstr);
00127     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00128     mon = atoi(tstr);
00129     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00130     day = atoi(tstr);
00131     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00132     hour = atoi(tstr);
00133     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00134     min = atoi(tstr);
00135     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00136
00137     /* Write data... */
00138     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g "
00139             "%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00140             t, zm, zs, Z(atm->p[atm->np - 1]),
00141             Z(atm->p[atm->np - atm->np / 10]),
00142             Z(atm->p[atm->np - atm->np / 4]),
00143             Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
00144             Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00145             lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00146             atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00147             atm->lon[atm->np - atm->np / 4],
00148             atm->lon[atm->np - atm->np / 10],
00149             atm->lon[atm->np - 1],
00150             latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00151             atm->lat[atm->np / 4], atm->lat[atm->np / 2],
00152             atm->lat[atm->np - atm->np / 4],
00153             atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00154   }
00155
00156   /* Close file... */
00157   fclose(out);
00158
00159   /* Free... */
```

```
00160   free(atm);
00161
00162   return EXIT_SUCCESS;
00163 }
```

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   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   atm_t *atm;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double latm, lats, lonm, lons, t, zm, zs;
00040
00041   int f, ip, year, mon, day, hour, min;
00042
00043   /* Allocate... */
00044   ALLOC(atm, atm_t, 1);
00045
00046   /* Check arguments... */
00047   if (argc < 4)
00048     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00049
00050   /* Read control parameters... */
00051   read_ctl(argv[1], argc, argv, &ctl);
00052
```

```
  /* Write info... */
  printf("Write center of mass data: %s\n", argv[2]);

  /* Create output file... */
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = altitude (mean) [km]\n"
          "# $3  = altitude (sigma) [km]\n"
          "# $4  = altitude (minimum) [km]\n"
          "# $5  = altitude (10%% percentile) [km]\n"
          "# $6  = altitude (1st quarter) [km]\n"
          "# $7  = altitude (median) [km]\n"
          "# $8  = altitude (3rd quarter) [km]\n"
          "# $9  = altitude (90%% percentile) [km]\n"
          "# $10 = altitude (maximum) [km]\n");
  fprintf(out,
          "# $11 = longitude (mean) [deg]\n"
          "# $12 = longitude (sigma) [deg]\n"
          "# $13 = longitude (minimum) [deg]\n"
          "# $14 = longitude (10%% percentile) [deg]\n"
          "# $15 = longitude (1st quarter) [deg]\n"
          "# $16 = longitude (median) [deg]\n"
          "# $17 = longitude (3rd quarter) [deg]\n"
          "# $18 = longitude (90%% percentile) [deg]\n"
          "# $19 = longitude (maximum) [deg]\n");
  fprintf(out,
          "# $20 = latitude (mean) [deg]\n"
          "# $21 = latitude (sigma) [deg]\n"
          "# $22 = latitude (minimum) [deg]\n"
          "# $23 = latitude (10%% percentile) [deg]\n"
          "# $24 = latitude (1st quarter) [deg]\n"
          "# $25 = latitude (median) [deg]\n"
          "# $26 = latitude (3rd quarter) [deg]\n"
          "# $27 = latitude (90%% percentile) [deg]\n"
          "# $28 = latitude (maximum) [deg]\n\n");

  /* Loop over files... */
  for (f = 3; f < argc; f++) {

    /* Read atmopheric data... */
    read_atm(argv[f], &ctl, atm);

    /* Initialize... */
    zm = zs = 0;
    lonm = lons = 0;
    latm = lats = 0;

    /* Calculate mean and standard deviation... */
    for (ip = 0; ip < atm->np; ip++) {
      zm += Z(atm->p[ip]) / atm->np;
      lonm += atm->lon[ip] / atm->np;
      latm += atm->lat[ip] / atm->np;
      zs += gsl_pow_2(Z(atm->p[ip])) / atm->np;
      lons += gsl_pow_2(atm->lon[ip]) / atm->np;
      lats += gsl_pow_2(atm->lat[ip]) / atm->np;
    }

    /* Normalize... */
    zs = sqrt(zs - gsl_pow_2(zm));
    lons = sqrt(lons - gsl_pow_2(lonm));
    lats = sqrt(lats - gsl_pow_2(latm));

    /* Sort arrays... */
    gsl_sort(atm->p, 1, (size_t) atm->np);
    gsl_sort(atm->lon, 1, (size_t) atm->np);
    gsl_sort(atm->lat, 1, (size_t) atm->np);

    /* Get time from filename... */
    sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
    year = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
    mon = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
    day = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
    hour = atoi(tstr);
    sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
    min = atoi(tstr);
    time2jsec(year, mon, day, hour, min, 0, 0, &t);

    /* Write data... */
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g "
            "%g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
```

```
00140             t, zm, zs, Z(atm->p[atm->np - 1]),
00141             Z(atm->p[atm->np - atm->np / 10]),
00142             Z(atm->p[atm->np - atm->np / 4]),
00143             Z(atm->p[atm->np / 2]), Z(atm->p[atm->np / 4]),
00144             Z(atm->p[atm->np / 10]), Z(atm->p[0]),
00145             lonm, lons, atm->lon[0], atm->lon[atm->np / 10],
00146             atm->lon[atm->np / 4], atm->lon[atm->np / 2],
00147             atm->lon[atm->np - atm->np / 4],
00148             atm->lon[atm->np - atm->np / 10],
00149             atm->lon[atm->np - 1],
00150             latm, lats, atm->lat[0], atm->lat[atm->np / 10],
00151             atm->lat[atm->np / 4], atm->lat[atm->np / 2],
00152             atm->lat[atm->np - atm->np / 4],
00153             atm->lat[atm->np - atm->np / 10], atm->lat[atm->np - 1]);
00154   }
00155
00156   /* Close file... */
00157   fclose(out);
00158
00159   /* Free... */
00160   free(atm);
00161
00162   return EXIT_SUCCESS;
00163 }
```

5.3 cluster.c File Reference

Clustering of trajectories.

Functions

- int main (int argc, char ∗argv[ ])

5.3.1 Detailed Description

Clustering of trajectories.

Definition in file cluster.c.

5.3.2 Function Documentation

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

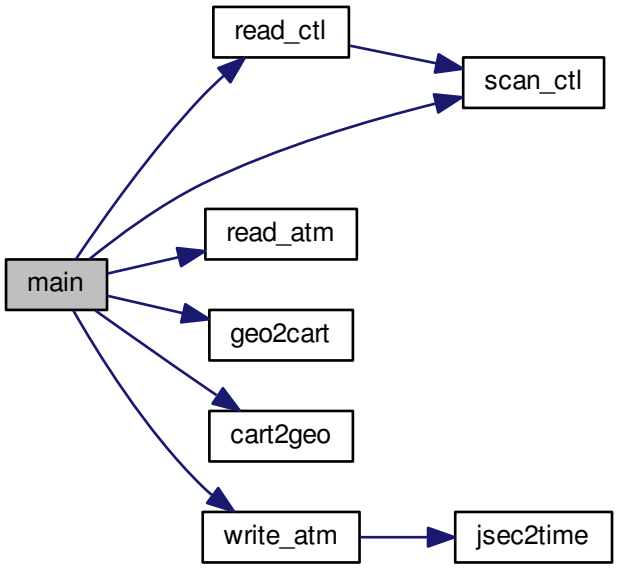
Definition at line 41 of file cluster.c.

```
00043                 {
00044
00045   ctl_t ctl;
00046
00047   atm_t *atm;
00048
00049   gsl_rng *rng;
00050
00051   FILE *out;
00052
00053   static double d2, *dist, lat, lon, rmsd[NS],
00054     x[3], xs[NT][NS][3], z, zs[NT][NS];
00055
00056   static int *cluster, f, idx[NS], ip, is, it, itmax, np[NS], ns;
00057
00058   /* Check arguments... */
00059   if (argc < 4)
00060     ERRMSG("Give parameters: <ctl> <cluster.log> <atm1> [<atm2> <atm3> ...]");
00061
00062   /* Read control parameters... */
```

```
00063   read_ctl(argv[1], argc, argv, &ctl);
00064   ns = (int) scan_ctl(argv[1], argc, argv, "CLUSTER_NS", -1, "7", NULL);
00065   if (ns > NS)
00066     ERRMSG("Too many seeds!");
00067   itmax =
00068     (int) scan_ctl(argv[1], argc, argv, "CLUSTER_ITMAX", -1, "10", NULL);
00069
00070   /* Initialize random number generator... */
00071   gsl_rng_env_setup();
00072   rng = gsl_rng_alloc(gsl_rng_default);
00073
00074   /* Allocate... */
00075   ALLOC(atm, atm_t, 1);
00076   ALLOC(cluster, int,
00077         NP);
00078   ALLOC(dist, double,
00079         NP * NS);
00080
00081   /* Create output file... */
00082   printf("Write cluster data: %s\n", argv[2]);
00083   if (!(out = fopen(argv[2], "w")))
00084     ERRMSG("Cannot create file!");
00085
00086   /* Write header... */
00087   fprintf(out,
00088           "# $1 = iteration index\n"
00089           "# $2 = seed index\n"
00090           "# $3 = time step index\n"
00091           "# $4 = mean altitude [km]\n"
00092           "# $5 = mean longitude [deg]\n"
00093           "# $6 = mean latitude [deg]\n"
00094           "# $7 = number of points\n" "# $8 = RMSD [km^2]\n");
00095
00096   /* Get seeds (random selection of trajectories)... */
00097   for (f = 3; f < argc; f++) {
00098
00099     /* Check number of timesteps... */
00100     if (f - 3 > NT)
00101       ERRMSG("Too many timesteps!");
00102
00103     /* Read atmopheric data... */
00104     read_atm(argv[f], &ctl, atm);
00105
00106     /* Pick seeds (random selection)... */
00107     if (f == 3)
00108       for (is = 0; is < ns; is++)
00109         idx[is] = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00110
00111     /* Save seeds... */
00112     for (is = 0; is < ns; is++) {
00113       geo2cart(0, atm->lon[idx[is]], atm->lat[idx[is]], xs[f - 3][is]);
00114       zs[f - 3][is] = Z(atm->p[idx[is]]);
00115     }
00116   }
00117
00118   /* Iterations... */
00119   for (it = 0; it < itmax; it++) {
00120
00121     /* Write output... */
00122     for (is = 0; is < ns; is++) {
00123       fprintf(out, "\n");
00124       for (f = 3; f < argc; f++) {
00125         cart2geo(xs[f - 3][is], &z, &lon, &lat);
00126         fprintf(out, "%d %d %d %g %g %g %d %g\n",
00127                 it, is, f - 3, zs[f - 3][is], lon, lat, np[is], rmsd[is]);
00128       }
00129     }
00130
00131     /* Init... */
00132     for (ip = 0; ip < atm->np; ip++)
00133       for (is = 0; is < ns; is++) {
00134         dist[ip * NS + is] = 0;
00135         rmsd[is] = 0;
00136       }
00137
00138     /* Get distances between seeds and trajectories... */
00139     for (f = 3; f < argc; f++) {
00140
00141       /* Read atmopheric data... */
00142       read_atm(argv[f], &ctl, atm);
00143
00144       /* Get distances... */
00145       for (ip = 0; ip < atm->np; ip++) {
00146         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
00147         z = Z(atm->p[ip]);
00148         for (is = 0; is < ns; is++) {
00149           d2 =
```

```
00150             DIST2(x, xs[f - 3][is]) + gsl_pow_2((z - zs[f - 3][is]) * 200.);
00151           dist[ip * NS + is] += d2;
00152           rmsd[is] += d2;
00153         }
00154       }
00155     }
00156
00157     /* Assign clusters... */
00158     for (ip = 0; ip < atm->np; ip++)
00159       cluster[ip] = (int) gsl_stats_min_index(&dist[ip * NS], 1, (size_t) ns);
00160
00161     /* Recalculate seeds (mean trajectories)... */
00162     for (f = 3; f < argc; f++) {
00163
00164       /* Read atmopheric data... */
00165       read_atm(argv[f], &ctl, atm);
00166
00167       /* Calculate new seeds... */
00168       for (is = 0; is < ns; is++) {
00169         xs[f - 3][is][0] = 0;
00170         xs[f - 3][is][1] = 0;
00171         xs[f - 3][is][2] = 0;
00172         zs[f - 3][is] = 0;
00173         np[is] = 0;
00174       }
00175       for (ip = 0; ip < atm->np; ip++) {
00176         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
00177         xs[f - 3][cluster[ip]][0] += x[0];
00178         xs[f - 3][cluster[ip]][1] += x[1];
00179         xs[f - 3][cluster[ip]][2] += x[2];
00180         zs[f - 3][cluster[ip]] += Z(atm->p[ip]);
00181         np[cluster[ip]]++;
00182       }
00183       for (is = 0; is < ns; is++) {
00184         xs[f - 3][is][0] /= np[is];
00185         xs[f - 3][is][1] /= np[is];
00186         xs[f - 3][is][2] /= np[is];
00187         zs[f - 3][is] /= np[is];
00188       }
00189     }
00190   }
00191
00192   /* Write output... */
00193   for (is = 0; is < ns; is++) {
00194     fprintf(out, "\n");
00195     for (f = 3; f < argc; f++) {
00196       cart2geo(xs[f - 3][is], &z, &lon, &lat);
00197       fprintf(out, "%d %d %d %g %g %g %d %g\n",
00198               it, is, f - 3, zs[f - 3][is], lon, lat, np[is], rmsd[is]);
00199     }
00200   }
00201
00202   /* Close output file... */
00203   fclose(out);
00204
00205   /* Write clustering results... */
00206   if (ctl.qnt_ens >= 0)
00207
00208     /* Recalculate seeds (mean trajectories)... */
00209     for (f = 3; f < argc; f++) {
00210
00211       /* Read atmopheric data... */
00212       read_atm(argv[f], &ctl, atm);
00213
00214       /* Set ensemble ID... */
00215       for (ip = 0; ip < atm->np; ip++)
00216         atm->q[ctl.qnt_ens][ip] = cluster[ip];
00217
00218       /* Write atmospheric data... */
00219       write_atm(argv[f], &ctl, atm, 0);
00220     }
00221
00222   /* Free... */
00223   gsl_rng_free(rng);
00224   free(atm);
00225   free(cluster);
00226   free(dist);
00227
00228   return EXIT_SUCCESS;
00229 }
```

Here is the call graph for this function:



5.4 cluster.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    Defines...
00029    ------------------------------------------------------------ */
00030
00032 #define NS 100
00033
00035 #define NT 1000
00036
00037 /* ------------------------------------------------------------
00038    Main...
00039    ------------------------------------------------------------ */
00040
00041 int main(
00042   int argc,
00043   char *argv[]) {
00044
00045   ctl_t ctl;
00046
```

```
00047   atm_t *atm;
00048
00049   gsl_rng *rng;
00050
00051   FILE *out;
00052
00053   static double d2, *dist, lat, lon, rmsd[NS],
00054     x[3], xs[NT][NS][3], z, zs[NT][NS];
00055
00056   static int *cluster, f, idx[NS], ip, is, it, itmax, np[NS], ns;
00057
00058   /* Check arguments... */
00059   if (argc < 4)
00060     ERRMSG("Give parameters: <ctl> <cluster.log> <atm1> [<atm2> <atm3> ...]");
00061
00062   /* Read control parameters... */
00063   read_ctl(argv[1], argc, argv, &ctl);
00064   ns = (int) scan_ctl(argv[1], argc, argv, "CLUSTER_NS", -1, "7", NULL);
00065   if (ns > NS)
00066     ERRMSG("Too many seeds!");
00067   itmax =
00068     (int) scan_ctl(argv[1], argc, argv, "CLUSTER_ITMAX", -1, "10", NULL);
00069
00070   /* Initialize random number generator... */
00071   gsl_rng_env_setup();
00072   rng = gsl_rng_alloc(gsl_rng_default);
00073
00074   /* Allocate... */
00075   ALLOC(atm, atm_t, 1);
00076   ALLOC(cluster, int,
00077         NP);
00078   ALLOC(dist, double,
00079         NP * NS);
00080
00081   /* Create output file... */
00082   printf("Write cluster data: %s\n", argv[2]);
00083   if (!(out = fopen(argv[2], "w")))
00084     ERRMSG("Cannot create file!");
00085
00086   /* Write header... */
00087   fprintf(out,
00088           "# $1 = iteration index\n"
00089           "# $2 = seed index\n"
00090           "# $3 = time step index\n"
00091           "# $4 = mean altitude [km]\n"
00092           "# $5 = mean longitude [deg]\n"
00093           "# $6 = mean latitude [deg]\n"
00094           "# $7 = number of points\n" "# $8 = RMSD [km^2]\n");
00095
00096   /* Get seeds (random selection of trajectories)... */
00097   for (f = 3; f < argc; f++) {
00098
00099     /* Check number of timesteps... */
00100     if (f - 3 > NT)
00101       ERRMSG("Too many timesteps!");
00102
00103     /* Read atmopheric data... */
00104     read_atm(argv[f], &ctl, atm);
00105
00106     /* Pick seeds (random selection)... */
00107     if (f == 3)
00108       for (is = 0; is < ns; is++)
00109         idx[is] = (int) gsl_rng_uniform_int(rng, (long unsigned int) atm->np);
00110
00111     /* Save seeds... */
00112     for (is = 0; is < ns; is++) {
00113       geo2cart(0, atm->lon[idx[is]], atm->lat[idx[is]], xs[f - 3][is]);
00114       zs[f - 3][is] = Z(atm->p[idx[is]]);
00115     }
00116   }
00117
00118   /* Iterations... */
00119   for (it = 0; it < itmax; it++) {
00120
00121     /* Write output... */
00122     for (is = 0; is < ns; is++) {
00123       fprintf(out, "\n");
00124       for (f = 3; f < argc; f++) {
00125         cart2geo(xs[f - 3][is], &z, &lon, &lat);
00126         fprintf(out, "%d %d %d %g %g %g %d %g\n",
00127                 it, is, f - 3, zs[f - 3][is], lon, lat, np[is], rmsd[is]);
00128       }
00129     }
00130
00131     /* Init... */
00132     for (ip = 0; ip < atm->np; ip++)
00133       for (is = 0; is < ns; is++) {
```

```
00134         dist[ip * NS + is] = 0;
00135         rmsd[is] = 0;
00136       }
00137
00138     /* Get distances between seeds and trajectories... */
00139     for (f = 3; f < argc; f++) {
00140
00141       /* Read atmopheric data... */
00142       read_atm(argv[f], &ctl, atm);
00143
00144       /* Get distances... */
00145       for (ip = 0; ip < atm->np; ip++) {
00146         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
00147         z = Z(atm->p[ip]);
00148         for (is = 0; is < ns; is++) {
00149           d2 =
00150             DIST2(x, xs[f - 3][is]) + gsl_pow_2((z - zs[f - 3][is]) * 200.);
00151           dist[ip * NS + is] += d2;
00152           rmsd[is] += d2;
00153         }
00154       }
00155     }
00156
00157     /* Assign clusters... */
00158     for (ip = 0; ip < atm->np; ip++)
00159       cluster[ip] = (int) gsl_stats_min_index(&dist[ip * NS], 1, (size_t) ns);
00160
00161     /* Recalculate seeds (mean trajectories)... */
00162     for (f = 3; f < argc; f++) {
00163
00164       /* Read atmopheric data... */
00165       read_atm(argv[f], &ctl, atm);
00166
00167       /* Calculate new seeds... */
00168       for (is = 0; is < ns; is++) {
00169         xs[f - 3][is][0] = 0;
00170         xs[f - 3][is][1] = 0;
00171         xs[f - 3][is][2] = 0;
00172         zs[f - 3][is] = 0;
00173         np[is] = 0;
00174       }
00175       for (ip = 0; ip < atm->np; ip++) {
00176         geo2cart(0, atm->lon[ip], atm->lat[ip], x);
00177         xs[f - 3][cluster[ip]][0] += x[0];
00178         xs[f - 3][cluster[ip]][1] += x[1];
00179         xs[f - 3][cluster[ip]][2] += x[2];
00180         zs[f - 3][cluster[ip]] += Z(atm->p[ip]);
00181         np[cluster[ip]]++;
00182       }
00183       for (is = 0; is < ns; is++) {
00184         xs[f - 3][is][0] /= np[is];
00185         xs[f - 3][is][1] /= np[is];
00186         xs[f - 3][is][2] /= np[is];
00187         zs[f - 3][is] /= np[is];
00188       }
00189     }
00190   }
00191
00192   /* Write output... */
00193   for (is = 0; is < ns; is++) {
00194     fprintf(out, "\n");
00195     for (f = 3; f < argc; f++) {
00196       cart2geo(xs[f - 3][is], &z, &lon, &lat);
00197       fprintf(out, "%d %d %d %g %g %g %d %g\n",
00198               it, is, f - 3, zs[f - 3][is], lon, lat, np[is], rmsd[is]);
00199     }
00200   }
00201
00202   /* Close output file... */
00203   fclose(out);
00204
00205   /* Write clustering results... */
00206   if (ctl.qnt_ens >= 0)
00207
00208     /* Recalculate seeds (mean trajectories)... */
00209     for (f = 3; f < argc; f++) {
00210
00211       /* Read atmopheric data... */
00212       read_atm(argv[f], &ctl, atm);
00213
00214       /* Set ensemble ID... */
00215       for (ip = 0; ip < atm->np; ip++)
00216         atm->q[ctl.qnt_ens][ip] = cluster[ip];
00217
00218       /* Write atmospheric data... */
00219       write_atm(argv[f], &ctl, atm, 0);
00220     }
```

```
00221
00222   /* Free... */
00223   gsl_rng_free(rng);
00224   free(atm);
00225   free(cluster);
00226   free(dist);
00227
00228   return EXIT_SUCCESS;
00229 }
```

5.5 dist.c File Reference

Calculate transport deviations of trajectories.

Functions

- int main (int argc, char ∗argv[ ])

5.5.1 Detailed Description

Calculate transport deviations of trajectories.

Definition in file dist.c.

5.5.2 Function Documentation

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

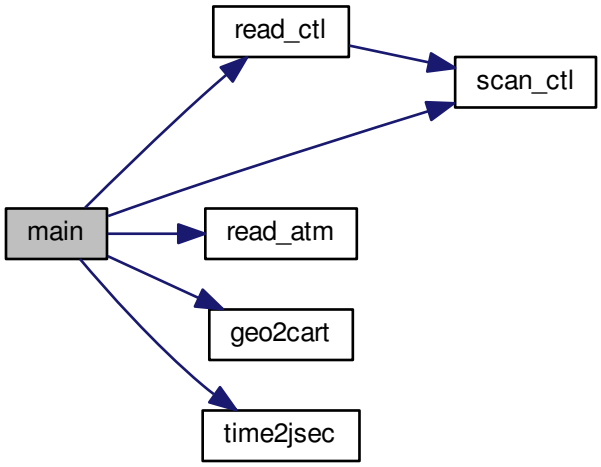
Definition at line 27 of file dist.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm1, *atm2;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double x0[3], x1[3], x2[3], *lon1, *lat1, *p1, *lh1, *lv1,
00040     *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, aqtd[NQ], rhtd, rvtd, t;
00041
00042   int ens, f, ip, iq, np, year, mon, day, hour, min;
00043
00044   /* Allocate... */
00045   ALLOC(atm1, atm_t, 1);
00046   ALLOC(atm2, atm_t, 1);
00047   ALLOC(lon1, double,
00048         NP);
00049   ALLOC(lat1, double,
00050         NP);
00051   ALLOC(p1, double,
00052         NP);
00053   ALLOC(lh1, double,
00054         NP);
00055   ALLOC(lv1, double,
00056         NP);
00057   ALLOC(lon2, double,
00058         NP);
00059   ALLOC(lat2, double,
00060         NP);
00061   ALLOC(p2, double,
00062         NP);
00063   ALLOC(lh2, double,
```

```
00064         NP);
00065   ALLOC(lv2, double,
00066         NP);
00067
00068   /* Check arguments... */
00069   if (argc < 5)
00070     ERRMSG("Give parameters: <ctl> <outfile> <atm1a> <atm1b>"
00071            " [<atm2a> <atm2b> ...]");
00072
00073   /* Read control parameters... */
00074   read_ctl(argv[1], argc, argv, &ctl);
00075   ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
00076
00077   /* Write info... */
00078   printf("Write transport deviations: %s\n", argv[2]);
00079
00080   /* Create output file... */
00081   if (!(out = fopen(argv[2], "w")))
00082     ERRMSG("Cannot create file!");
00083
00084   /* Write header... */
00085   fprintf(out,
00086           "# $1 = time [s]\n"
00087           "# $2 = AHTD [km]\n"
00088           "# $3 = RHTD [km]\n" "# $4 = AVTD [km]\n" "# $5 = RVTD [km]\n");
00089   for (iq = 0; iq < ctl.nq; iq++)
00090     fprintf(out,
00091             "# $%d = AQTD (%s) [%s]\n",
00092             6 + iq, ctl.qnt_name[iq], ctl.qnt_unit[iq]);
00093   fprintf(out, "\n");
00094
00095   /* Loop over file pairs... */
00096   for (f = 3; f < argc; f += 2) {
00097
00098     /* Read atmopheric data... */
00099     read_atm(argv[f], &ctl, atm1);
00100     read_atm(argv[f + 1], &ctl, atm2);
00101
00102     /* Check if structs match... */
00103     if (atm1->np != atm2->np)
00104       ERRMSG("Different numbers of parcels!");
00105     for (ip = 0; ip < atm1->np; ip++)
00106       if (atm1->time[ip] != atm2->time[ip])
00107         ERRMSG("Times do not match!");
00108
00109     /* Init... */
00110     np = 0;
00111     ahtd = avtd = rhtd = rvtd = 0;
00112     for (iq = 0; iq < ctl.nq; iq++)
00113       aqtd[iq] = 0;
00114
00115     /* Loop over air parcels... */
00116     for (ip = 0; ip < atm1->np; ip++)
00117       if (ens < 0 || (ctl.qnt_ens >= 0 && atm1->q[ctl.qnt_ens][ip] == ens)) {
00118
00119         /* Get Cartesian coordinates... */
00120         geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00121         geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00122
00123         /* Calculate absolute transport deviations... */
00124         ahtd += DIST(x1, x2);
00125         avtd += fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00126         for (iq = 0; iq < ctl.nq; iq++)
00127           aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00128
00129         /* Calculate relative transport deviations... */
00130         if (f > 3) {
00131
00132           /* Get trajectory lengths... */
00133           geo2cart(0, lon1[ip], lat1[ip], x0);
00134           lh1[ip] += DIST(x0, x1);
00135           lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00136
00137           geo2cart(0, lon2[ip], lat2[ip], x0);
00138           lh2[ip] += DIST(x0, x2);
00139           lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00140
00141           /* Get relative transport deviations... */
00142           if (lh1[ip] + lh2[ip] > 0)
00143             rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00144           if (lv1[ip] + lv2[ip] > 0)
00145             rvtd += 200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]))
00146               / (lv1[ip] + lv2[ip]);
00147         }
00148
00149         /* Save positions of air parcels... */
00150         lon1[ip] = atm1->lon[ip];
```

```
00151         lat1[ip] = atm1->lat[ip];
00152         p1[ip] = atm1->p[ip];
00153
00154         lon2[ip] = atm2->lon[ip];
00155         lat2[ip] = atm2->lat[ip];
00156         p2[ip] = atm2->p[ip];
00157
00158         /* Increment air parcel counter... */
00159         np++;
00160       }
00161
00162     /* Get time from filename... */
00163     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00164     year = atoi(tstr);
00165     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00166     mon = atoi(tstr);
00167     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00168     day = atoi(tstr);
00169     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00170     hour = atoi(tstr);
00171     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00172     min = atoi(tstr);
00173     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00174
00175     /* Write output... */
00176     fprintf(out, "%.2f %g %g %g %g", t,
00177             ahtd / np, rhtd / np, avtd / np, rvtd / np);
00178     for (iq = 0; iq < ctl.nq; iq++) {
00179       fprintf(out, " ");
00180       fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
00181     }
00182     fprintf(out, "\n");
00183   }
00184
00185   /* Close file... */
00186   fclose(out);
00187
00188   /* Free... */
00189   free(atm1);
00190   free(atm2);
00191   free(lon1);
00192   free(lat1);
00193   free(p1);
00194   free(lh1);
00195   free(lv1);
00196   free(lon2);
00197   free(lat2);
00198   free(p2);
00199   free(lh2);
00200   free(lv2);
00201
00202   return EXIT_SUCCESS;
00203 }
```

Here is the call graph for this function:



5.6 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   Copyright (C) 2013-2018 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   atm_t *atm1, *atm2;
00034
00035   FILE *out;
00036
00037   char tstr[LEN];
00038
00039   double x0[3], x1[3], x2[3], *lon1, *lat1, *p1, *lh1, *lv1,
00040     *lon2, *lat2, *p2, *lh2, *lv2, ahtd, avtd, aqtd[NQ], rhtd, rvtd, t;
00041
00042   int ens, f, ip, iq, np, year, mon, day, hour, min;
00043
00044   /* Allocate... */
00045   ALLOC(atm1, atm_t, 1);
00046   ALLOC(atm2, atm_t, 1);
00047   ALLOC(lon1, double,
00048         NP);
00049   ALLOC(lat1, double,
```

```
00050         NP);
00051   ALLOC(p1, double,
00052         NP);
00053   ALLOC(lh1, double,
00054         NP);
00055   ALLOC(lv1, double,
00056         NP);
00057   ALLOC(lon2, double,
00058         NP);
00059   ALLOC(lat2, double,
00060         NP);
00061   ALLOC(p2, double,
00062         NP);
00063   ALLOC(lh2, double,
00064         NP);
00065   ALLOC(lv2, double,
00066         NP);
00067
00068   /* Check arguments... */
00069   if (argc < 5)
00070     ERRMSG("Give parameters: <ctl> <outfile> <atm1a> <atm1b>"
00071            " [<atm2a> <atm2b> ...]");
00072
00073   /* Read control parameters... */
00074   read_ctl(argv[1], argc, argv, &ctl);
00075   ens = (int) scan_ctl(argv[1], argc, argv, "DIST_ENS", -1, "-1", NULL);
00076
00077   /* Write info... */
00078   printf("Write transport deviations: %s\n", argv[2]);
00079
00080   /* Create output file... */
00081   if (!(out = fopen(argv[2], "w")))
00082     ERRMSG("Cannot create file!");
00083
00084   /* Write header... */
00085   fprintf(out,
00086           "# $1 = time [s]\n"
00087           "# $2 = AHTD [km]\n"
00088           "# $3 = RHTD [km]\n" "# $4 = AVTD [km]\n" "# $5 = RVTD [km]\n");
00089   for (iq = 0; iq < ctl.nq; iq++)
00090     fprintf(out,
00091             "# $%d = AQTD (%s) [%s]\n",
00092             6 + iq, ctl.qnt_name[iq], ctl.qnt_unit[iq]);
00093   fprintf(out, "\n");
00094
00095   /* Loop over file pairs... */
00096   for (f = 3; f < argc; f += 2) {
00097
00098     /* Read atmopheric data... */
00099     read_atm(argv[f], &ctl, atm1);
00100     read_atm(argv[f + 1], &ctl, atm2);
00101
00102     /* Check if structs match... */
00103     if (atm1->np != atm2->np)
00104       ERRMSG("Different numbers of parcels!");
00105     for (ip = 0; ip < atm1->np; ip++)
00106       if (atm1->time[ip] != atm2->time[ip])
00107         ERRMSG("Times do not match!");
00108
00109     /* Init... */
00110     np = 0;
00111     ahtd = avtd = rhtd = rvtd = 0;
00112     for (iq = 0; iq < ctl.nq; iq++)
00113       aqtd[iq] = 0;
00114
00115     /* Loop over air parcels... */
00116     for (ip = 0; ip < atm1->np; ip++)
00117       if (ens < 0 || (ctl.qnt_ens >= 0 && atm1->q[ctl.qnt_ens][ip] == ens)) {
00118
00119         /* Get Cartesian coordinates... */
00120         geo2cart(0, atm1->lon[ip], atm1->lat[ip], x1);
00121         geo2cart(0, atm2->lon[ip], atm2->lat[ip], x2);
00122
00123         /* Calculate absolute transport deviations... */
00124         ahtd += DIST(x1, x2);
00125         avtd += fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]));
00126         for (iq = 0; iq < ctl.nq; iq++)
00127           aqtd[iq] += fabs(atm1->q[iq][ip] - atm2->q[iq][ip]);
00128
00129         /* Calculate relative transport deviations... */
00130         if (f > 3) {
00131
00132           /* Get trajectory lengths... */
00133           geo2cart(0, lon1[ip], lat1[ip], x0);
00134           lh1[ip] += DIST(x0, x1);
00135           lv1[ip] += fabs(Z(p1[ip]) - Z(atm1->p[ip]));
00136
```

```
00137           geo2cart(0, lon2[ip], lat2[ip], x0);
00138           lh2[ip] += DIST(x0, x2);
00139           lv2[ip] += fabs(Z(p2[ip]) - Z(atm2->p[ip]));
00140
00141           /* Get relative transport deviations... */
00142           if (lh1[ip] + lh2[ip] > 0)
00143             rhtd += 200. * DIST(x1, x2) / (lh1[ip] + lh2[ip]);
00144           if (lv1[ip] + lv2[ip] > 0)
00145             rvtd += 200. * fabs(Z(atm1->p[ip]) - Z(atm2->p[ip]))
00146               / (lv1[ip] + lv2[ip]);
00147         }
00148
00149         /* Save positions of air parcels... */
00150         lon1[ip] = atm1->lon[ip];
00151         lat1[ip] = atm1->lat[ip];
00152         p1[ip] = atm1->p[ip];
00153
00154         lon2[ip] = atm2->lon[ip];
00155         lat2[ip] = atm2->lat[ip];
00156         p2[ip] = atm2->p[ip];
00157
00158         /* Increment air parcel counter... */
00159         np++;
00160       }
00161
00162     /* Get time from filename... */
00163     sprintf(tstr, "%.4s", &argv[f][strlen(argv[f]) - 20]);
00164     year = atoi(tstr);
00165     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 15]);
00166     mon = atoi(tstr);
00167     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 12]);
00168     day = atoi(tstr);
00169     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 9]);
00170     hour = atoi(tstr);
00171     sprintf(tstr, "%.2s", &argv[f][strlen(argv[f]) - 6]);
00172     min = atoi(tstr);
00173     time2jsec(year, mon, day, hour, min, 0, 0, &t);
00174
00175     /* Write output... */
00176     fprintf(out, "%.2f %g %g %g %g", t,
00177             ahtd / np, rhtd / np, avtd / np, rvtd / np);
00178     for (iq = 0; iq < ctl.nq; iq++) {
00179       fprintf(out, " ");
00180       fprintf(out, ctl.qnt_format[iq], aqtd[iq] / np);
00181     }
00182     fprintf(out, "\n");
00183   }
00184
00185   /* Close file... */
00186   fclose(out);
00187
00188   /* Free... */
00189   free(atm1);
00190   free(atm2);
00191   free(lon1);
00192   free(lat1);
00193   free(p1);
00194   free(lh1);
00195   free(lv1);
00196   free(lon2);
00197   free(lat2);
00198   free(p2);
00199   free(lh2);
00200   free(lv2);
00201
00202   return EXIT_SUCCESS;
00203 }
```

5.7 extract.c File Reference

Extract single trajectory from atmospheric data files.

Functions

- int main (int argc, char ∗argv[ ])

5.7.1 Detailed Description

Extract single trajectory from atmospheric data files.

Definition in file extract.c.

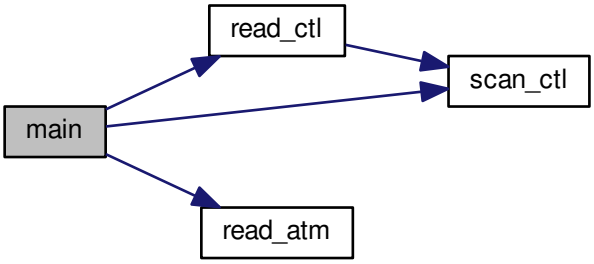
5.7.2 Function Documentation

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

Definition at line 27 of file extract.c.

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

Here is the call graph for this function:



5.8 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
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm;
00034
00035   FILE *in, *out;
00036
00037   int f, ip, iq;
00038
00039   /* Allocate... */
00040   ALLOC(atm, atm_t, 1);
00041
00042   /* Check arguments... */
00043   if (argc < 4)
00044     ERRMSG("Give parameters: <ctl> <outfile> <atm1> [<atm2> ...]");
00045
00046   /* Read control parameters... */
00047   read_ctl(argv[1], argc, argv, &ctl);
00048   ip = (int) scan_ctl(argv[1], argc, argv, "EXTRACT_IP", -1, "0", NULL);
00049
00050   /* Write info... */
00051   printf("Write trajectory data: %s\n", argv[2]);
00052
00053   /* Create output file... */
00054   if (!(out = fopen(argv[2], "w")))
00055     ERRMSG("Cannot create file!");
00056
00057   /* Write header... */
00058   fprintf(out,
00059           "# $1 = time [s]\n"
```

```
00060           "# $2 = altitude [km]\n"
00061           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00062   for (iq = 0; iq < ctl.nq; iq++)
00063     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00064             ctl.qnt_unit[iq]);
00065   fprintf(out, "\n");
00066
00067   /* Loop over files... */
00068   for (f = 3; f < argc; f++) {
00069
00070     /* Read atmopheric data... */
00071     if (!(in = fopen(argv[f], "r")))
00072       continue;
00073     else
00074       fclose(in);
00075     read_atm(argv[f], &ctl, atm);
00076
00077     /* Write data... */
00078     fprintf(out, "%.2f %g %g %g", atm->time[ip],
00079             Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
00080     for (iq = 0; iq < ctl.nq; iq++) {
00081       fprintf(out, " ");
00082       fprintf(out, ctl.qnt_format[iq], atm->q[iq][ip]);
00083     }
00084     fprintf(out, "\n");
00085   }
00086
00087   /* Close file... */
00088   fclose(out);
00089
00090   /* Free... */
00091   free(atm);
00092
00093   return EXIT_SUCCESS;
00094 }
```

5.9 init.c File Reference

Create atmospheric data file with initial air parcel positions.

Functions

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

5.9.1 Detailed Description

Create atmospheric data file with initial air parcel positions.

Definition in file init.c.

5.9.2 Function Documentation

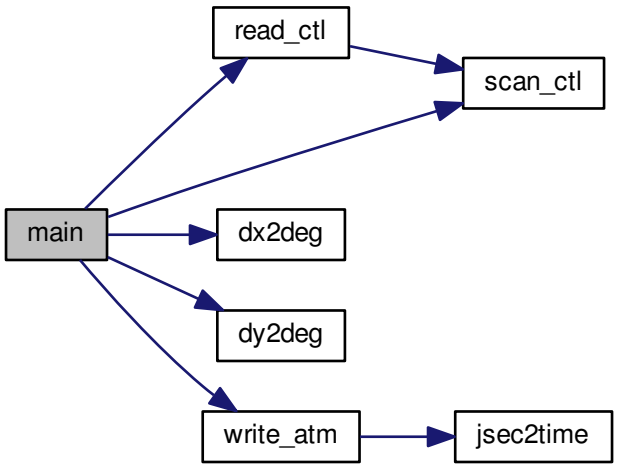
5.9.2.1 int main (int *argc*, char * *argv[]*)

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 even, 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   even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
00073   rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074   m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075
00076   /* Initialize random number generator... */
00077   gsl_rng_env_setup();
00078   rng = gsl_rng_alloc(gsl_rng_default);
00079
00080   /* Create grid... */
00081   for (t = t0; t <= t1; t += dt)
00082     for (z = z0; z <= z1; z += dz)
00083       for (lon = lon0; lon <= lon1; lon += dlon)
00084         for (lat = lat0; lat <= lat1; lat += dlat)
00085           for (irep = 0; irep < rep; irep++) {
00086
00087             /* Set position... */
00088             atm->time[atm->np]
00089               = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00090                  + ut * (gsl_rng_uniform(rng) - 0.5));
00091             atm->p[atm->np]
00092               = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093                   + uz * (gsl_rng_uniform(rng) - 0.5));
00094             atm->lon[atm->np]
00095               = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00096                  + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
00097                  + ulon * (gsl_rng_uniform(rng) - 0.5));
00098             do {
00099               atm->lat[atm->np]
00100                 = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00101                    + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
00102                    + ulat * (gsl_rng_uniform(rng) - 0.5));
00103             } while (even && gsl_rng_uniform(rng) >
00104                      fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106             /* Set particle counter... */
00107             if ((++atm->np) >= NP)
00108               ERRMSG("Too many particles!");
00109           }
00110
00111   /* Check number of air parcels... */
00112   if (atm->np <= 0)
00113     ERRMSG("Did not create any air parcels!");
00114
00115   /* Initialize mass... */
```

```
00116   if (ctl.qnt_m >= 0)
00117     for (ip = 0; ip < atm->np; ip++)
00118       atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120   /* Save data... */
00121   write_atm(argv[2], &ctl, atm, t0);
00122
00123   /* Free... */
00124   gsl_rng_free(rng);
00125   free(atm);
00126
00127   return EXIT_SUCCESS;
00128 }
```

Here is the call graph for this function:



5.10 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) 2013-2018 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 even, 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   even = (int) scan_ctl(argv[1], argc, argv, "INIT_EVENLY", -1, "1", NULL);
00073   rep = (int) scan_ctl(argv[1], argc, argv, "INIT_REP", -1, "1", NULL);
00074   m = scan_ctl(argv[1], argc, argv, "INIT_MASS", -1, "0", NULL);
00075
00076   /* Initialize random number generator... */
00077   gsl_rng_env_setup();
00078   rng = gsl_rng_alloc(gsl_rng_default);
00079
00080   /* Create grid... */
00081   for (t = t0; t <= t1; t += dt)
00082     for (z = z0; z <= z1; z += dz)
00083       for (lon = lon0; lon <= lon1; lon += dlon)
00084         for (lat = lat0; lat <= lat1; lat += dlat)
00085           for (irep = 0; irep < rep; irep++) {
00086
00087             /* Set position... */
00088             atm->time[atm->np]
00089               = (t + gsl_ran_gaussian_ziggurat(rng, st / 2.3548)
00090                  + ut * (gsl_rng_uniform(rng) - 0.5));
00091             atm->p[atm->np]
00092               = P(z + gsl_ran_gaussian_ziggurat(rng, sz / 2.3548)
00093                   + uz * (gsl_rng_uniform(rng) - 0.5));
00094             atm->lon[atm->np]
00095               = (lon + gsl_ran_gaussian_ziggurat(rng, slon / 2.3548)
00096                  + gsl_ran_gaussian_ziggurat(rng, dx2deg(sx, lat) / 2.3548)
00097                  + ulon * (gsl_rng_uniform(rng) - 0.5));
00098             do {
00099               atm->lat[atm->np]
00100                 = (lat + gsl_ran_gaussian_ziggurat(rng, slat / 2.3548)
00101                    + gsl_ran_gaussian_ziggurat(rng, dy2deg(sx) / 2.3548)
00102                    + ulat * (gsl_rng_uniform(rng) - 0.5));
00103             } while (even && gsl_rng_uniform(rng) >
00104                      fabs(cos(atm->lat[atm->np] * M_PI / 180.)));
00105
00106             /* Set particle counter... */
00107             if ((++atm->np) >= NP)
00108               ERRMSG("Too many particles!");
00109           }
00110
00111   /* Check number of air parcels... */
00112   if (atm->np <= 0)
00113     ERRMSG("Did not create any air parcels!");
00114
00115   /* Initialize mass... */
00116   if (ctl.qnt_m >= 0)
00117     for (ip = 0; ip < atm->np; ip++)
00118       atm->q[ctl.qnt_m][ip] = m / atm->np;
00119
00120   /* Save data... */
```

```
00121   write_atm(argv[2], &ctl, atm, t0);
00122
00123   /* Free... */
00124   gsl_rng_free(rng);
00125   free(atm);
00126
00127   return EXIT_SUCCESS;
00128 }
```

5.11 jsec2time.c File Reference

Convert Julian seconds to date.

Functions

- int main (int argc, char ∗argv[ ])

5.11.1 Detailed Description

Convert Julian seconds to date.

Definition in file jsec2time.c.

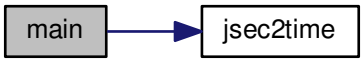
5.11.2 Function Documentation

5.11.2.1 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:



5.12 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   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 < 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 }
```

5.13 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)

  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 definitions.

Definition in file libtrac.c.

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 }
```

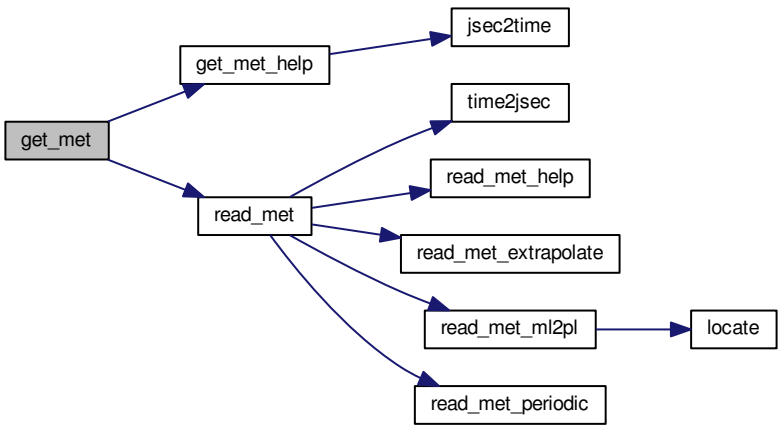
5.13.2.9 void get_met (ctl_t * *ctl,* char * *metbase,* double *t,* met_t * *met0,* met_t * *met1*)

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:



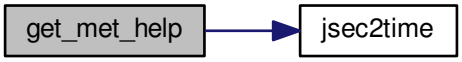
5.13.2.10 void get_met_help (double *t,* int *direct,* char * *metbase,* double *dt_met,* char * *filename*)

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 }
```

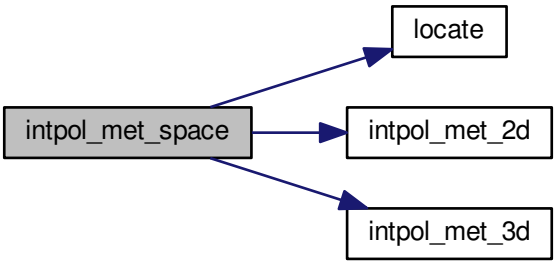
5.13.2.13 void intpol_met_space (met_t * *met*, double *p*, double *lon*, double *lat*, double * *ps*, double * *t*, double * *u*, double * *v*, double * *w*, double * *h2o*, double * *o3*)

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:



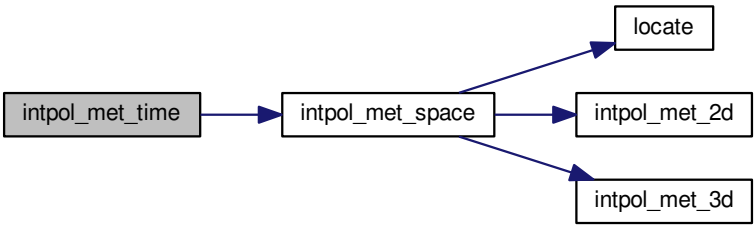
5.13.2.14 void intpol_met_time (met_t * *met0*, met_t * *met1*, double *ts*, double *p*, double *lon*, double *lat*, double * *ps*, double * *t*, double * *u*, double * *v*, double * *w*, double * *h2o*, double * *o3*)

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 binary data... */
00430     if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
00431       FREAD(&atm->np, int,
00432             1,
00433             in);
00434       FREAD(atm->time, double,
00435               (size_t) atm->np,
00436             in);
00437       FREAD(atm->p, double,
00438               (size_t) atm->np,
00439             in);
00440       FREAD(atm->lon, double,
00441               (size_t) atm->np,
00442             in);
00443       FREAD(atm->lat, double,
00444               (size_t) atm->np,
00445             in);
00446       for (iq = 0; iq < ctl->nq; iq++)
00447         FREAD(atm->q[iq], double,
00448                 (size_t) atm->np,
00449               in);
00450     }
00451
00452     /* Read ASCII data... */
00453     else {
00454
00455       /* Read line... */
00456       while (fgets(line, LEN, in)) {
00457
00458         /* Read data... */
00459         TOK(line, tok, "%lg", atm->time[atm->np]);
00460         TOK(NULL, tok, "%lg", atm->p[atm->np]);
00461         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00462         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00463         for (iq = 0; iq < ctl->nq; iq++)
00464           TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00465
00466         /* Convert altitude to pressure... */
00467         atm->p[atm->np] = P(atm->p[atm->np]);
00468
00469         /* Increment data point counter... */
```

```
00470       if ((++atm->np) > NP)
00471         ERRMSG("Too many data points!");
00472     }
00473   }
00474
00475   /* Close file... */
00476   fclose(in);
00477
00478   /* Check number of points... */
00479   if (atm->np < 1)
00480     ERRMSG("Can not read any data!");
00481 }
```

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

Read control parameters.

Definition at line 485 of file libtrac.c.

```
00489               {
00490
00491   int ip, iq;
00492
00493   /* Write info... */
00494   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00495          "(executable: %s | compiled: %s, %s)\n\n",
00496          argv[0], __DATE__, __TIME__);
00497
00498   /* Initialize quantity indices... */
00499   ctl->qnt_ens = -1;
00500   ctl->qnt_m = -1;
00501   ctl->qnt_r = -1;
00502   ctl->qnt_rho = -1;
00503   ctl->qnt_ps = -1;
00504   ctl->qnt_p = -1;
00505   ctl->qnt_t = -1;
00506   ctl->qnt_u = -1;
00507   ctl->qnt_v = -1;
00508   ctl->qnt_w = -1;
00509   ctl->qnt_h2o = -1;
00510   ctl->qnt_o3 = -1;
00511   ctl->qnt_theta = -1;
00512   ctl->qnt_pv = -1;
00513   ctl->qnt_tice = -1;
00514   ctl->qnt_tsts = -1;
00515   ctl->qnt_tnat = -1;
00516   ctl->qnt_gw_var = -1;
00517   ctl->qnt_stat = -1;
00518
00519   /* Read quantities... */
00520   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00521   if (ctl->nq > NQ)
00522     ERRMSG("Too many quantities!");
00523   for (iq = 0; iq < ctl->nq; iq++) {
00524
00525     /* Read quantity name and format... */
00526     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
00527     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
00528              ctl->qnt_format[iq]);
00529
00530     /* Try to identify quantity... */
00531     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00532       ctl->qnt_ens = iq;
00533       sprintf(ctl->qnt_unit[iq], "-");
00534     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00535       ctl->qnt_m = iq;
00536       sprintf(ctl->qnt_unit[iq], "kg");
00537     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
00538       ctl->qnt_r = iq;
00539       sprintf(ctl->qnt_unit[iq], "m");
00540     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00541       ctl->qnt_rho = iq;
00542       sprintf(ctl->qnt_unit[iq], "kg/m^3");
00543     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00544       ctl->qnt_ps = iq;
00545       sprintf(ctl->qnt_unit[iq], "hPa");
00546     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00547       ctl->qnt_p = iq;
00548       sprintf(ctl->qnt_unit[iq], "hPa");
00549     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
```

```
00550       ctl->qnt_t = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00553       ctl->qnt_u = iq;
00554       sprintf(ctl->qnt_unit[iq], "m/s");
00555     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00556       ctl->qnt_v = iq;
00557       sprintf(ctl->qnt_unit[iq], "m/s");
00558     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00559       ctl->qnt_w = iq;
00560       sprintf(ctl->qnt_unit[iq], "hPa/s");
00561     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00562       ctl->qnt_h2o = iq;
00563       sprintf(ctl->qnt_unit[iq], "1");
00564     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00565       ctl->qnt_o3 = iq;
00566       sprintf(ctl->qnt_unit[iq], "1");
00567     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00568       ctl->qnt_theta = iq;
00569       sprintf(ctl->qnt_unit[iq], "K");
00570     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00571       ctl->qnt_pv = iq;
00572       sprintf(ctl->qnt_unit[iq], "PVU");
00573     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00574       ctl->qnt_tice = iq;
00575       sprintf(ctl->qnt_unit[iq], "K");
00576     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00577       ctl->qnt_tsts = iq;
00578       sprintf(ctl->qnt_unit[iq], "K");
00579     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00580       ctl->qnt_tnat = iq;
00581       sprintf(ctl->qnt_unit[iq], "K");
00582     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
00583       ctl->qnt_gw_var = iq;
00584       sprintf(ctl->qnt_unit[iq], "K^2");
00585     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00586       ctl->qnt_stat = iq;
00587       sprintf(ctl->qnt_unit[iq], "-");
00588     } else
00589       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00590   }
00591
00592   /* Time steps of simulation... */
00593   ctl->direction =
00594     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00595   if (ctl->direction != -1 && ctl->direction != 1)
00596     ERRMSG("Set DIRECTION to -1 or 1!");
00597   ctl->t_start =
00598     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00599   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00600   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00601
00602   /* Meteorological data... */
00603   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00604   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00605   if (ctl->met_np > EP)
00606     ERRMSG("Too many levels!");
00607   for (ip = 0; ip < ctl->met_np; ip++)
00608     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00609   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
00610
00611   /* Isosurface parameters... */
00612   ctl->isosurf
00613     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00614   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00615
00616   /* Diffusion parameters... */
00617   ctl->turb_dx_trop
00618     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00619   ctl->turb_dx_strat
00620     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00621   ctl->turb_dz_trop
00622     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00623   ctl->turb_dz_strat
00624     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00625   ctl->turb_meso =
00626     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00627
00628   /* Life time of particles... */
00629   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00630   ctl->tdec_strat =
00631     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00632
00633   /* PSC analysis... */
00634   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00635   ctl->psc_hno3 =
00636     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
```

```
00637
00638   /* Gravity wave analysis... */
00639   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
00640
00641   /* Output of atmospheric data... */
00642   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00643   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00644   ctl->atm_dt_out =
00645     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00646   ctl->atm_filter =
00647     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00648   ctl->atm_bin =
00649     (int) scan_ctl(filename, argc, argv, "ATM_BIN", -1, "0", NULL);
00650
00651   /* Output of CSI data... */
00652   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00653   ctl->csi_dt_out =
00654     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
00655   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00656            ctl->csi_obsfile);
00657   ctl->csi_obsmin =
00658     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00659   ctl->csi_modmin =
00660     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00661   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
00662   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
00663   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00664   ctl->csi_lon0 =
00665     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
00666   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00667   ctl->csi_nx =
00668     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
00669   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
00670   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
00671   ctl->csi_ny =
00672     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00673
00674   /* Output of ensemble data... */
00675   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
00676
00677   /* Output of grid data... */
00678   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00679            ctl->grid_basename);
00680   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00681   ctl->grid_dt_out =
00682     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00683   ctl->grid_sparse =
00684     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00685   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
00686   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00687   ctl->grid_nz =
00688     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00689   ctl->grid_lon0 =
00690     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
00691   ctl->grid_lon1 =
00692     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00693   ctl->grid_nx =
00694     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00695   ctl->grid_lat0 =
00696     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
00697   ctl->grid_lat1 =
00698     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00699   ctl->grid_ny =
00700     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00701
00702   /* Output of profile data... */
00703   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00704            ctl->prof_basename);
00705   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
00706   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00707   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00708   ctl->prof_nz =
00709     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00710   ctl->prof_lon0 =
00711     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
00712   ctl->prof_lon1 =
00713     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00714   ctl->prof_nx =
00715     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00716   ctl->prof_lat0 =
00717     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
```

```
00718   ctl->prof_lat1 =
00719     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00720   ctl->prof_ny =
00721     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00722
00723   /* Output of station data... */
00724   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00725            ctl->stat_basename);
00726   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
00727   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
00728   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00729 }
```

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.

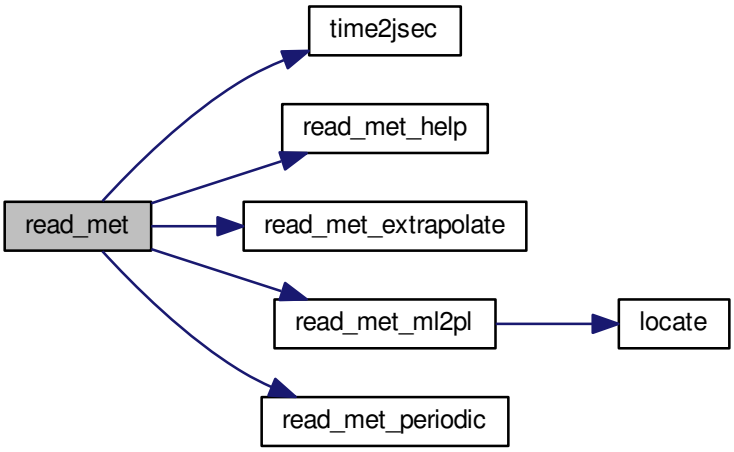
Definition at line 733 of file libtrac.c.

```
00736               {
00737
00738   char cmd[LEN], levname[LEN], tstr[10];
00739
00740   static float help[EX * EY];
00741
00742   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00743
00744   size_t np, nx, ny;
00745
00746   /* Write info... */
00747   printf("Read meteorological data: %s\n", filename);
00748
00749   /* Get time from filename... */
00750   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00751   year = atoi(tstr);
00752   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00753   mon = atoi(tstr);
00754   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00755   day = atoi(tstr);
00756   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00757   hour = atoi(tstr);
00758   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00759
00760   /* Open netCDF file... */
00761   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00762
00763     /* Try to stage meteo file... */
00764     STOP_TIMER(TIMER_INPUT);
00765     START_TIMER(TIMER_STAGE);
00766     if (ctl->met_stage[0] != '-') {
00767       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
00768               year, mon, day, hour, filename);
00769       if (system(cmd) != 0)
00770         ERRMSG("Error while staging meteo data!");
00771     }
00772     STOP_TIMER(TIMER_STAGE);
00773     START_TIMER(TIMER_INPUT);
00774
00775     /* Try to open again... */
00776     NC(nc_open(filename, NC_NOWRITE, &ncid));
```

```
00777   }
00778
00779   /* Get dimensions... */
00780   NC(nc_inq_dimid(ncid, "lon", &dimid));
00781   NC(nc_inq_dimlen(ncid, dimid, &nx));
00782   if (nx < 2 || nx > EX)
00783     ERRMSG("Number of longitudes out of range!");
00784
00785   NC(nc_inq_dimid(ncid, "lat", &dimid));
00786   NC(nc_inq_dimlen(ncid, dimid, &ny));
00787   if (ny < 2 || ny > EY)
00788     ERRMSG("Number of latitudes out of range!");
00789
00790   sprintf(levname, "lev");
00791   NC(nc_inq_dimid(ncid, levname, &dimid));
00792   NC(nc_inq_dimlen(ncid, dimid, &np));
00793   if (np == 1) {
00794     sprintf(levname, "lev_2");
00795     NC(nc_inq_dimid(ncid, levname, &dimid));
00796     NC(nc_inq_dimlen(ncid, dimid, &np));
00797   }
00798   if (np < 2 || np > EP)
00799     ERRMSG("Number of levels out of range!");
00800
00801   /* Store dimensions... */
00802   met->np = (int) np;
00803   met->nx = (int) nx;
00804   met->ny = (int) ny;
00805
00806   /* Get horizontal grid... */
00807   NC(nc_inq_varid(ncid, "lon", &varid));
00808   NC(nc_get_var_double(ncid, varid, met->lon));
00809   NC(nc_inq_varid(ncid, "lat", &varid));
00810   NC(nc_get_var_double(ncid, varid, met->lat));
00811
00812   /* Read meteorological data... */
00813   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00814   read_met_help(ncid, "u", "U", met, met->u, 1.0);
00815   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00816   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00817   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00818   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00819
00820   /* Meteo data on pressure levels... */
00821   if (ctl->met_np <= 0) {
00822
00823     /* Read pressure levels from file... */
00824     NC(nc_inq_varid(ncid, levname, &varid));
00825     NC(nc_get_var_double(ncid, varid, met->p));
00826     for (ip = 0; ip < met->np; ip++)
00827       met->p[ip] /= 100.;
00828
00829     /* Extrapolate data for lower boundary... */
00830     read_met_extrapolate(met);
00831   }
00832
00833   /* Meteo data on model levels... */
00834   else {
00835
00836     /* Read pressure data from file... */
00837     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00838
00839     /* Interpolate from model levels to pressure levels... */
00840     read_met_ml2pl(ctl, met, met->t);
00841     read_met_ml2pl(ctl, met, met->u);
00842     read_met_ml2pl(ctl, met, met->v);
00843     read_met_ml2pl(ctl, met, met->w);
00844     read_met_ml2pl(ctl, met, met->h2o);
00845     read_met_ml2pl(ctl, met, met->o3);
00846
00847     /* Set pressure levels... */
00848     met->np = ctl->met_np;
00849     for (ip = 0; ip < met->np; ip++)
00850       met->p[ip] = ctl->met_p[ip];
00851   }
00852
00853   /* Check ordering of pressure levels... */
00854   for (ip = 1; ip < met->np; ip++)
00855     if (met->p[ip - 1] < met->p[ip])
00856       ERRMSG("Pressure levels must be descending!");
00857
00858   /* Read surface pressure... */
00859   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
00860       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00861     NC(nc_get_var_float(ncid, varid, help));
00862     for (iy = 0; iy < met->ny; iy++)
00863       for (ix = 0; ix < met->nx; ix++)
```

```
00864         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00865     } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
00866                || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00867       NC(nc_get_var_float(ncid, varid, help));
00868       for (iy = 0; iy < met->ny; iy++)
00869         for (ix = 0; ix < met->nx; ix++)
00870           met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00871     } else
00872       for (ix = 0; ix < met->nx; ix++)
00873         for (iy = 0; iy < met->ny; iy++)
00874           met->ps[ix][iy] = met->p[0];
00875
00876     /* Create periodic boundary conditions... */
00877     read_met_periodic(met);
00878
00879     /* Close file... */
00880     NC(nc_close(ncid));
00881   }
```

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 885 of file libtrac.c.

```
00886                 {
00887
00888     int ip, ip0, ix, iy;
00889
00890     /* Loop over columns... */
00891     for (ix = 0; ix < met->nx; ix++)
00892       for (iy = 0; iy < met->ny; iy++) {
00893
00894         /* Find lowest valid data point... */
00895         for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00896           if (!gsl_finite(met->t[ix][iy][ip0])
00897               || !gsl_finite(met->u[ix][iy][ip0])
00898               || !gsl_finite(met->v[ix][iy][ip0])
00899               || !gsl_finite(met->w[ix][iy][ip0]))
00900             break;
00901
00902         /* Extrapolate... */
```

```
00903       for (ip = ip0; ip >= 0; ip--) {
00904         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00905         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00906         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00907         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00908         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00909         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00910       }
00911     }
00912 }
```

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 916 of file libtrac.c.

```
00922             {
00923
00924   static float help[EX * EY * EP];
00925
00926   int ip, ix, iy, n = 0, varid;
00927
00928   /* Check if variable exists... */
00929   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00930     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00931       return;
00932
00933   /* Read data... */
00934   NC(nc_get_var_float(ncid, varid, help));
00935
00936   /* Copy and check data... */
00937   for (ip = 0; ip < met->np; ip++)
00938     for (iy = 0; iy < met->ny; iy++)
00939       for (ix = 0; ix < met->nx; ix++) {
00940         dest[ix][iy][ip] = scl * help[n++];
00941         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00942           dest[ix][iy][ip] = GSL_NAN;
00943       }
00944 }
```

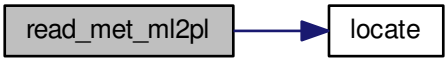
5.13.2.22 void read_met_ml2pl (ctl_t * *ctl*, met_t * *met*, float *var[EX][EY][EP]*)

Convert meteorological data from model levels to pressure levels.

Definition at line 948 of file libtrac.c.

```
00951                         {
00952
00953   double aux[EP], p[EP], pt;
00954
00955   int ip, ip2, ix, iy;
00956
00957   /* Loop over columns... */
00958   for (ix = 0; ix < met->nx; ix++)
00959     for (iy = 0; iy < met->ny; iy++) {
00960
00961       /* Copy pressure profile... */
00962       for (ip = 0; ip < met->np; ip++)
00963         p[ip] = met->pl[ix][iy][ip];
00964
00965       /* Interpolate... */
00966       for (ip = 0; ip < ctl->met_np; ip++) {
00967         pt = ctl->met_p[ip];
00968         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00969           pt = p[0];
00970         else if ((pt > p[met->np - 1] && p[1] > p[0])
00971                  || (pt < p[met->np - 1] && p[1] < p[0]))
00972           pt = p[met->np - 1];
00973         ip2 = locate(p, met->np, pt);
00974         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00975                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00976       }
00977
00978       /* Copy data... */
00979       for (ip = 0; ip < ctl->met_np; ip++)
00980         var[ix][iy][ip] = (float) aux[ip];
00981     }
00982 }
```

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 986 of file libtrac.c.

```
00987                {
00988
00989   int ip, iy;
00990
00991   /* Check longitudes... */
00992   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00993              + met->lon[1] - met->lon[0] - 360) < 0.01))
00994     return;
00995
00996   /* Increase longitude counter... */
00997   if ((++met->nx) > EX)
00998     ERRMSG("Cannot create periodic boundary conditions!");
00999
01000   /* Set longitude... */
01001   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01002
01003   /* Loop over latitudes and pressure levels... */
01004   for (iy = 0; iy < met->ny; iy++)
01005     for (ip = 0; ip < met->np; ip++) {
01006       met->ps[met->nx - 1][iy] = met->ps[0][iy];
01007       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01008       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01009       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01010       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01011       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01012       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01013     }
01014 }
```

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 1018 of file libtrac.c.

```
01025                {
01026
01027   FILE *in = NULL;
01028
01029   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01030     msg[LEN], rvarname[LEN], rval[LEN];
01031
01032   int contain = 0, i;
01033
01034   /* Open file... */
01035   if (filename[strlen(filename) - 1] != '-')
01036     if (!(in = fopen(filename, "r")))
```

```
01037       ERRMSG("Cannot open file!");
01038
01039   /* Set full variable name... */
01040   if (arridx >= 0) {
01041     sprintf(fullname1, "%s[%d]", varname, arridx);
01042     sprintf(fullname2, "%s[*]", varname);
01043   } else {
01044     sprintf(fullname1, "%s", varname);
01045     sprintf(fullname2, "%s", varname);
01046   }
01047
01048   /* Read data... */
01049   if (in != NULL)
01050     while (fgets(line, LEN, in))
01051       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
01052         if (strcasecmp(rvarname, fullname1) == 0 ||
01053             strcasecmp(rvarname, fullname2) == 0) {
01054           contain = 1;
01055           break;
01056         }
01057   for (i = 1; i < argc - 1; i++)
01058     if (strcasecmp(argv[i], fullname1) == 0 ||
01059         strcasecmp(argv[i], fullname2) == 0) {
01060       sprintf(rval, "%s", argv[i + 1]);
01061       contain = 1;
01062       break;
01063     }
01064
01065   /* Close file... */
01066   if (in != NULL)
01067     fclose(in);
01068
01069   /* Check for missing variables... */
01070   if (!contain) {
01071     if (strlen(defvalue) > 0)
01072       sprintf(rval, "%s", defvalue);
01073     else {
01074       sprintf(msg, "Missing variable %s!\n", fullname1);
01075       ERRMSG(msg);
01076     }
01077   }
01078
01079   /* Write info... */
01080   printf("%s = %s\n", fullname1, rval);
01081
01082   /* Return values... */
01083   if (value != NULL)
01084     sprintf(value, "%s", rval);
01085   return atof(rval);
01086 }
```

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 1090 of file libtrac.c.

```
01098                 {
01099
01100   struct tm t0, t1;
01101
01102   t0.tm_year = 100;
01103   t0.tm_mon = 0;
01104   t0.tm_mday = 1;
01105   t0.tm_hour = 0;
01106   t0.tm_min = 0;
01107   t0.tm_sec = 0;
01108
01109   t1.tm_year = year - 1900;
01110   t1.tm_mon = mon - 1;
01111   t1.tm_mday = day;
01112   t1.tm_hour = hour;
01113   t1.tm_min = min;
01114   t1.tm_sec = sec;
01115
01116   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01117 }
```

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

Measure wall-clock time.

Definition at line 1121 of file libtrac.c.

```
01124            {
01125
01126   static double starttime[NTIMER], runtime[NTIMER];
01127
01128   /* Check id... */
01129   if (id < 0 || id >= NTIMER)
01130     ERRMSG("Too many timers!");
01131
01132   /* Start timer... */
01133   if (mode == 1) {
01134     if (starttime[id] <= 0)
01135       starttime[id] = omp_get_wtime();
01136     else
01137       ERRMSG("Timer already started!");
01138   }
01139
01140   /* Stop timer... */
01141   else if (mode == 2) {
01142     if (starttime[id] > 0) {
01143       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01144       starttime[id] = -1;
01145     }
01146   }
01147
01148   /* Print timer... */
01149   else if (mode == 3)
01150     printf("%s = %.3f s\n", name, runtime[id]);
01151 }
```

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

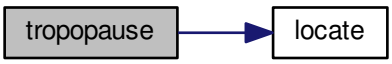
Definition at line 1155 of file libtrac.c.

```
01157              {
01158
01159   static double doys[12]
01160   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01161
01162   static double lats[73]
01163     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01164     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01165     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01166     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01167     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01168     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01169     75, 77.5, 80, 82.5, 85, 87.5, 90
01170   };
01171
01172   static double tps[12][73]
01173     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01174          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01175          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01176          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01177          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01178          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01179          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01180          275.3, 275.6, 275.4, 274.1, 273.5},
01181   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01182    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01183    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01184    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01185    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01186    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01187    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01188    287.5, 286.2, 285.8},
01189   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01190    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01191    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01192    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01193    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01194    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
```

```
01195    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01196    304.3, 304.9, 306, 306.6, 306.2, 306},
01197   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01198    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01199    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01200    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01201    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01202    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01203    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01204    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01205   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01206    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01207    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01208    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01209    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01210    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01211    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01212    325.3, 325.8, 325.8},
01213   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01214    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01215    228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01216    105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01217    106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01218    127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01219    251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01220    308.5, 312.2, 313.1, 313.3},
01221   {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01222    187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01223    235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01224    110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01225    111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01226    117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01227    224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01228    275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01229   {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01230    185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01231    233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01232    110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01233    112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01234    120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01235    230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01236    278.2, 282.6, 287.4, 290.9, 292.5, 293},
01237   {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01238    183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01239    243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01240    114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01241    110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01242    114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01243    203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01244    276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01245   {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01246    215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01247    237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01248    111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01249    106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01250    112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01251    206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01252    279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01253    305.1},
01254   {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01255    253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01256    223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01257    108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01258    102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01259    109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01260    241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01261    286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01262   {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01263    284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01264    175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01265    100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01266    100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01267    186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01268    280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01269    281.7, 281.1, 281.2}
01270   };
01271
01272   double doy, p0, p1, pt;
01273
01274   int imon, ilat;
01275
01276   /* Get day of year... */
01277   doy = fmod(t / 86400., 365.25);
01278   while (doy < 0)
01279     doy += 365.25;
01280
01281   /* Get indices... */
```

```
01282   imon = locate(doys, 12, doy);
01283   ilat = locate(lats, 73, lat);
01284
01285   /* Get tropopause pressure... */
01286   p0 = LIN(lats[ilat], tps[imon][ilat],
01287            lats[ilat + 1], tps[imon][ilat + 1], lat);
01288   p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01289            lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01290   pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01291
01292   /* Return tropopause pressure... */
01293   return pt;
01294 }
```

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 1298 of file libtrac.c.

```
01302             {
01303
01304   FILE *in, *out;
01305
01306   char line[LEN];
01307
01308   double r, t0, t1;
01309
01310   int ip, iq, year, mon, day, hour, min, sec;
01311
01312   /* Set time interval for output... */
01313   t0 = t - 0.5 * ctl->dt_mod;
01314   t1 = t + 0.5 * ctl->dt_mod;
01315
01316   /* Check if gnuplot output is requested... */
01317   if (ctl->atm_gpfile[0] != '-') {
01318
01319     /* Write info... */
01320     printf("Plot atmospheric data: %s.png\n", filename);
01321
01322     /* Create gnuplot pipe... */
01323     if (!(out = popen("gnuplot", "w")))
01324       ERRMSG("Cannot create pipe to gnuplot!");
01325
01326     /* Set plot filename... */
01327     fprintf(out, "set out \"%s.png\"\n", filename);
01328
01329     /* Set time string... */
01330     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01331     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01332             year, mon, day, hour, min);
01333
01334     /* Dump gnuplot file to pipe... */
01335     if (!(in = fopen(ctl->atm_gpfile, "r")))
01336       ERRMSG("Cannot open file!");
01337     while (fgets(line, LEN, in))
01338       fprintf(out, "%s", line);
01339     fclose(in);
01340   }
01341
```

```
01342   else {
01343
01344     /* Write info... */
01345     printf("Write atmospheric data: %s\n", filename);
01346
01347     /* Create file... */
01348     if (!(out = fopen(filename, "w")))
01349       ERRMSG("Cannot create file!");
01350   }
01351
01352   /* Write binary data... */
01353   if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
01354     FWRITE(&atm->np, int,
01355            1,
01356            out);
01357     FWRITE(atm->time, double,
01358              (size_t) atm->np,
01359            out);
01360     FWRITE(atm->p, double,
01361              (size_t) atm->np,
01362            out);
01363     FWRITE(atm->lon, double,
01364              (size_t) atm->np,
01365            out);
01366     FWRITE(atm->lat, double,
01367              (size_t) atm->np,
01368            out);
01369     for (iq = 0; iq < ctl->nq; iq++)
01370       FWRITE(atm->q[iq], double,
01371                (size_t) atm->np,
01372              out);
01373   }
01374
01375   /* Write ASCII data... */
01376   else {
01377
01378     /* Write header... */
01379     fprintf(out,
01380             "# $1 = time [s]\n"
01381             "# $2 = altitude [km]\n"
01382             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01383     for (iq = 0; iq < ctl->nq; iq++)
01384       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01385               ctl->qnt_unit[iq]);
01386     fprintf(out, "\n");
01387
01388     /* Write data... */
01389     for (ip = 0; ip < atm->np; ip++) {
01390
01391       /* Check time... */
01392       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01393         continue;
01394
01395       /* Write output... */
01396       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01397               atm->lon[ip], atm->lat[ip]);
01398       for (iq = 0; iq < ctl->nq; iq++) {
01399         fprintf(out, " ");
01400         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01401       }
01402       fprintf(out, "\n");
01403     }
01404   }
01405
01406   /* Close file... */
01407   fclose(out);
01408 }
```

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 1412 of file libtrac.c.

```
01416            {
01417
01418   static FILE *in, *out;
01419
01420   static char line[LEN];
01421
01422   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01423     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01424
01425   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01426
01427   /* Init... */
01428   if (!init) {
01429     init = 1;
01430
01431     /* Check quantity index for mass... */
01432     if (ctl->qnt_m < 0)
01433       ERRMSG("Need quantity mass to analyze CSI!");
01434
01435     /* Open observation data file... */
01436     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01437     if (!(in = fopen(ctl->csi_obsfile, "r")))
01438       ERRMSG("Cannot open file!");
01439
01440     /* Create new file... */
01441     printf("Write CSI data: %s\n", filename);
01442     if (!(out = fopen(filename, "w")))
01443       ERRMSG("Cannot create file!");
01444
01445     /* Write header... */
01446     fprintf(out,
01447             "# $1 = time [s]\n"
01448             "# $2 = number of hits (cx)\n"
01449             "# $3 = number of misses (cy)\n"
01450             "# $4 = number of false alarms (cz)\n"
01451             "# $5 = number of observations (cx + cy)\n"
01452             "# $6 = number of forecasts (cx + cz)\n"
01453             "# $7 = bias (forecasts/observations) [%%]\n"
01454             "# $8 = probability of detection (POD) [%%]\n"
01455             "# $9 = false alarm rate (FAR) [%%]\n"
01456             "# $10 = critical success index (CSI) [%%]\n\n");
01457   }
01458
01459   /* Set time interval... */
01460   t0 = t - 0.5 * ctl->dt_mod;
01461   t1 = t + 0.5 * ctl->dt_mod;
01462
01463   /* Initialize grid cells... */
01464   for (ix = 0; ix < ctl->csi_nx; ix++)
01465     for (iy = 0; iy < ctl->csi_ny; iy++)
01466       for (iz = 0; iz < ctl->csi_nz; iz++)
01467         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01468
01469   /* Read data... */
01470   while (fgets(line, LEN, in)) {
01471
01472     /* Read data... */
01473     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01474         5)
01475       continue;
01476
01477     /* Check time... */
01478     if (rt < t0)
01479       continue;
01480     if (rt > t1)
01481       break;
01482
01483     /* Calculate indices... */
01484     ix = (int) ((rlon - ctl->csi_lon0)
01485                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01486     iy = (int) ((rlat - ctl->csi_lat0)
01487                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01488     iz = (int) ((rz - ctl->csi_z0)
01489                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01490
01491     /* Check indices... */
01492     if (ix < 0 || ix >= ctl->csi_nx ||
```

```
01493         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01494       continue;
01495
01496     /* Get mean observation index... */
01497     obsmean[ix][iy][iz] += robs;
01498     obscount[ix][iy][iz]++;
01499   }
01500
01501   /* Analyze model data... */
01502   for (ip = 0; ip < atm->np; ip++) {
01503
01504     /* Check time... */
01505     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01506       continue;
01507
01508     /* Get indices... */
01509     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01510                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01511     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01512                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01513     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01514                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01515
01516     /* Check indices... */
01517     if (ix < 0 || ix >= ctl->csi_nx ||
01518         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01519       continue;
01520
01521     /* Get total mass in grid cell... */
01522     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01523   }
01524
01525   /* Analyze all grid cells... */
01526   for (ix = 0; ix < ctl->csi_nx; ix++)
01527     for (iy = 0; iy < ctl->csi_ny; iy++)
01528       for (iz = 0; iz < ctl->csi_nz; iz++) {
01529
01530         /* Calculate mean observation index... */
01531         if (obscount[ix][iy][iz] > 0)
01532           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01533
01534         /* Calculate column density... */
01535         if (modmean[ix][iy][iz] > 0) {
01536           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01537           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01538           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01539           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01540             * cos(lat * M_PI / 180.);
01541           modmean[ix][iy][iz] /= (1e6 * area);
01542         }
01543
01544         /* Calculate CSI... */
01545         if (obscount[ix][iy][iz] > 0) {
01546           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01547               modmean[ix][iy][iz] >= ctl->csi_modmin)
01548             cx++;
01549           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01550                    modmean[ix][iy][iz] < ctl->csi_modmin)
01551             cy++;
01552           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01553                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01554             cz++;
01555         }
01556       }
01557
01558   /* Write output... */
01559   if (fmod(t, ctl->csi_dt_out) == 0) {
01560
01561     /* Write... */
01562     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01563             t, cx, cy, cz, cx + cy, cx + cz,
01564             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01565             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01566             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01567             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01568
01569     /* Set counters to zero... */
01570     cx = cy = cz = 0;
01571   }
01572
01573   /* Close file... */
01574   if (t == ctl->t_stop)
01575     fclose(out);
01576 }
```

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

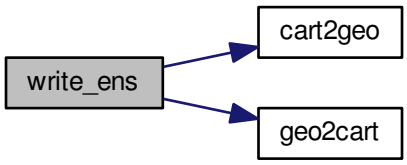
Write ensemble data.

Definition at line 1580 of file libtrac.c.

```
01584           {
01585
01586   static FILE *out;
01587
01588   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01589     t0, t1, x[NENS][3], xm[3];
01590
01591   static int init, ip, iq;
01592
01593   static size_t i, n;
01594
01595   /* Init... */
01596   if (!init) {
01597     init = 1;
01598
01599     /* Check quantities... */
01600     if (ctl->qnt_ens < 0)
01601       ERRMSG("Missing ensemble IDs!");
01602
01603     /* Create new file... */
01604     printf("Write ensemble data: %s\n", filename);
01605     if (!(out = fopen(filename, "w")))
01606       ERRMSG("Cannot create file!");
01607
01608     /* Write header... */
01609     fprintf(out,
01610             "# $1 = time [s]\n"
01611             "# $2 = altitude [km]\n"
01612             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01613     for (iq = 0; iq < ctl->nq; iq++)
01614       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01615               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01616     for (iq = 0; iq < ctl->nq; iq++)
01617       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01618               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01619     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01620   }
01621
01622   /* Set time interval... */
01623   t0 = t - 0.5 * ctl->dt_mod;
01624   t1 = t + 0.5 * ctl->dt_mod;
01625
01626   /* Init... */
01627   ens = GSL_NAN;
01628   n = 0;
01629
01630   /* Loop over air parcels... */
01631   for (ip = 0; ip < atm->np; ip++) {
01632
01633     /* Check time... */
01634     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01635       continue;
01636
01637     /* Check ensemble id... */
01638     if (atm->q[ctl->qnt_ens][ip] != ens) {
01639
01640       /* Write results... */
01641       if (n > 0) {
01642
01643         /* Get mean position... */
01644         xm[0] = xm[1] = xm[2] = 0;
01645         for (i = 0; i < n; i++) {
01646           xm[0] += x[i][0] / (double) n;
01647           xm[1] += x[i][1] / (double) n;
01648           xm[2] += x[i][2] / (double) n;
01649         }
01650         cart2geo(xm, &dummy, &lon, &lat);
01651         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01652                 lat);
01653
01654         /* Get quantity statistics... */
01655         for (iq = 0; iq < ctl->nq; iq++) {
01656           fprintf(out, " ");
01657           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01658         }
01659         for (iq = 0; iq < ctl->nq; iq++) {
01660           fprintf(out, " ");
```

```
01661           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01662         }
01663         fprintf(out, " %lu\n", n);
01664       }
01665
01666       /* Init new ensemble... */
01667       ens = atm->q[ctl->qnt_ens][ip];
01668       n = 0;
01669     }
01670
01671     /* Save data... */
01672     p[n] = atm->p[ip];
01673     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01674     for (iq = 0; iq < ctl->nq; iq++)
01675       q[iq][n] = atm->q[iq][ip];
01676     if ((++n) >= NENS)
01677       ERRMSG("Too many data points!");
01678   }
01679
01680   /* Write results... */
01681   if (n > 0) {
01682
01683     /* Get mean position... */
01684     xm[0] = xm[1] = xm[2] = 0;
01685     for (i = 0; i < n; i++) {
01686       xm[0] += x[i][0] / (double) n;
01687       xm[1] += x[i][1] / (double) n;
01688       xm[2] += x[i][2] / (double) n;
01689     }
01690     cart2geo(xm, &dummy, &lon, &lat);
01691     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01692
01693     /* Get quantity statistics... */
01694     for (iq = 0; iq < ctl->nq; iq++) {
01695       fprintf(out, " ");
01696       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01697     }
01698     for (iq = 0; iq < ctl->nq; iq++) {
01699       fprintf(out, " ");
01700       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01701     }
01702     fprintf(out, " %lu\n", n);
01703   }
01704
01705   /* Close file... */
01706   if (t == ctl->t_stop)
01707     fclose(out);
01708 }
```

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*)

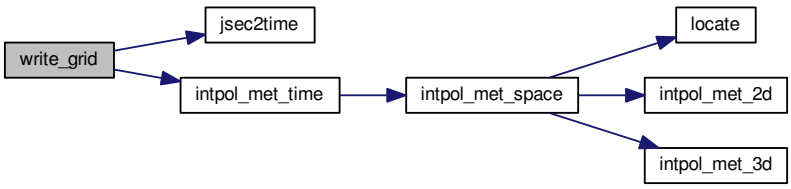
Write gridded data.

Definition at line 1712 of file libtrac.c.

```
01718           {
01719
01720   FILE *in, *out;
01721
01722   char line[LEN];
01723
01724   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01725     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01726
01727   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01728
01729   /* Check dimensions... */
01730   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01731     ERRMSG("Grid dimensions too large!");
01732
01733   /* Check quantity index for mass... */
01734   if (ctl->qnt_m < 0)
01735     ERRMSG("Need quantity mass to write grid data!");
01736
01737   /* Set time interval for output... */
01738   t0 = t - 0.5 * ctl->dt_mod;
01739   t1 = t + 0.5 * ctl->dt_mod;
01740
01741   /* Set grid box size... */
01742   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01743   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01744   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01745
01746   /* Initialize grid... */
01747   for (ix = 0; ix < ctl->grid_nx; ix++)
01748     for (iy = 0; iy < ctl->grid_ny; iy++)
01749       for (iz = 0; iz < ctl->grid_nz; iz++)
01750         grid_m[ix][iy][iz] = 0;
01751
01752   /* Average data... */
01753   for (ip = 0; ip < atm->np; ip++)
01754     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01755
01756       /* Get index... */
01757       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01758       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01759       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01760
01761       /* Check indices... */
01762       if (ix < 0 || ix >= ctl->grid_nx ||
01763           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01764         continue;
01765
01766       /* Add mass... */
01767       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01768     }
01769
01770   /* Check if gnuplot output is requested... */
01771   if (ctl->grid_gpfile[0] != '-') {
01772
01773     /* Write info... */
01774     printf("Plot grid data: %s.png\n", filename);
01775
01776     /* Create gnuplot pipe... */
01777     if (!(out = popen("gnuplot", "w")))
01778       ERRMSG("Cannot create pipe to gnuplot!");
01779
01780     /* Set plot filename... */
01781     fprintf(out, "set out \"%s.png\"\n", filename);
01782
01783     /* Set time string... */
01784     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01785     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01786             year, mon, day, hour, min);
01787
01788     /* Dump gnuplot file to pipe... */
01789     if (!(in = fopen(ctl->grid_gpfile, "r")))
01790       ERRMSG("Cannot open file!");
01791     while (fgets(line, LEN, in))
01792       fprintf(out, "%s", line);
01793     fclose(in);
01794   }
01795
01796   else {
01797
01798     /* Write info... */
01799     printf("Write grid data: %s\n", filename);
01800
01801     /* Create file... */
01802     if (!(out = fopen(filename, "w")))
01803       ERRMSG("Cannot create file!");
01804   }
```

```
01805
01806   /* Write header... */
01807   fprintf(out,
01808           "# $1 = time [s]\n"
01809           "# $2 = altitude [km]\n"
01810           "# $3 = longitude [deg]\n"
01811           "# $4 = latitude [deg]\n"
01812           "# $5 = surface area [km^2]\n"
01813           "# $6 = layer width [km]\n"
01814           "# $7 = temperature [K]\n"
01815           "# $8 = column density [kg/m^2]\n"
01816           "# $9 = mass mixing ratio [1]\n\n");
01817
01818   /* Write data... */
01819   for (ix = 0; ix < ctl->grid_nx; ix++) {
01820     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01821       fprintf(out, "\n");
01822     for (iy = 0; iy < ctl->grid_ny; iy++) {
01823       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01824         fprintf(out, "\n");
01825       for (iz = 0; iz < ctl->grid_nz; iz++)
01826         if (!ctl->grid_sparse
01827             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01828
01829           /* Set coordinates... */
01830           z = ctl->grid_z0 + dz * (iz + 0.5);
01831           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01832           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01833
01834           /* Get pressure and temperature... */
01835           press = P(z);
01836           intpol_met_time(met0, met1, t, press, lon, lat,
01837                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01838
01839           /* Calculate surface area... */
01840           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01841             * cos(lat * M_PI / 180.);
01842
01843           /* Calculate column density... */
01844           cd = grid_m[ix][iy][iz] / (1e6 * area);
01845
01846           /* Calculate mass mixing ratio... */
01847           rho_air = 100. * press / (287.058 * temp);
01848           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01849
01850           /* Write output... */
01851           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01852                   t, z, lon, lat, area, dz, temp, cd, mmr);
01853         }
01854     }
01855   }
01856
01857   /* Close file... */
01858   fclose(out);
01859 }
```

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*)

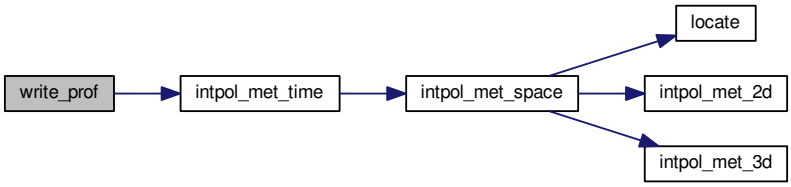
Write profile data.

Definition at line 1863 of file libtrac.c.

```
01869           {
01870
01871   static FILE *in, *out;
01872
01873   static char line[LEN];
01874
01875   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01876     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01877     press, temp, rho_air, mmr, h2o, o3;
01878
01879   static int init, obscount[GX][GY], ip, ix, iy, iz;
01880
01881   /* Init... */
01882   if (!init) {
01883     init = 1;
01884
01885     /* Check quantity index for mass... */
01886     if (ctl->qnt_m < 0)
01887       ERRMSG("Need quantity mass!");
01888
01889     /* Check dimensions... */
01890     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01891       ERRMSG("Grid dimensions too large!");
01892
01893     /* Open observation data file... */
01894     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01895     if (!(in = fopen(ctl->prof_obsfile, "r")))
01896       ERRMSG("Cannot open file!");
01897
01898     /* Create new file... */
01899     printf("Write profile data: %s\n", filename);
01900     if (!(out = fopen(filename, "w")))
01901       ERRMSG("Cannot create file!");
01902
01903     /* Write header... */
01904     fprintf(out,
01905             "# $1  = time [s]\n"
01906             "# $2  = altitude [km]\n"
01907             "# $3  = longitude [deg]\n"
01908             "# $4  = latitude [deg]\n"
01909             "# $5  = pressure [hPa]\n"
01910             "# $6  = temperature [K]\n"
01911             "# $7  = mass mixing ratio [1]\n"
01912             "# $8  = H2O volume mixing ratio [1]\n"
01913             "# $9  = O3 volume mixing ratio [1]\n"
01914             "# $10 = mean BT index [K]\n");
01915
01916     /* Set grid box size... */
01917     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01918     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01919     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01920   }
01921
01922   /* Set time interval... */
01923   t0 = t - 0.5 * ctl->dt_mod;
01924   t1 = t + 0.5 * ctl->dt_mod;
01925
01926   /* Initialize... */
01927   for (ix = 0; ix < ctl->prof_nx; ix++)
01928     for (iy = 0; iy < ctl->prof_ny; iy++) {
01929       obsmean[ix][iy] = 0;
01930       obscount[ix][iy] = 0;
01931       tmean[ix][iy] = 0;
01932       for (iz = 0; iz < ctl->prof_nz; iz++)
01933         mass[ix][iy][iz] = 0;
01934     }
01935
01936   /* Read data... */
01937   while (fgets(line, LEN, in)) {
01938
01939     /* Read data... */
01940     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01941       continue;
01942
01943     /* Check time... */
01944     if (rt < t0)
01945       continue;
01946     if (rt > t1)
01947       break;
01948
01949     /* Calculate indices... */
01950     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01951     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01952
01953     /* Check indices... */
01954     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01955       continue;
```

```
01956
01957     /* Get mean observation index... */
01958     obsmean[ix][iy] += robs;
01959     tmean[ix][iy] += rt;
01960     obscount[ix][iy]++;
01961   }
01962
01963   /* Analyze model data... */
01964   for (ip = 0; ip < atm->np; ip++) {
01965
01966     /* Check time... */
01967     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01968       continue;
01969
01970     /* Get indices... */
01971     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01972     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01973     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01974
01975     /* Check indices... */
01976     if (ix < 0 || ix >= ctl->prof_nx ||
01977         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01978       continue;
01979
01980     /* Get total mass in grid cell... */
01981     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01982   }
01983
01984   /* Extract profiles... */
01985   for (ix = 0; ix < ctl->prof_nx; ix++)
01986     for (iy = 0; iy < ctl->prof_ny; iy++)
01987       if (obscount[ix][iy] > 0) {
01988
01989         /* Write output... */
01990         fprintf(out, "\n");
01991
01992         /* Loop over altitudes... */
01993         for (iz = 0; iz < ctl->prof_nz; iz++) {
01994
01995           /* Set coordinates... */
01996           z = ctl->prof_z0 + dz * (iz + 0.5);
01997           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01998           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01999
02000           /* Get meteorological data... */
02001           press = P(z);
02002           intpol_met_time(met0, met1, t, press, lon, lat,
02003                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02004
02005           /* Calculate mass mixing ratio... */
02006           rho_air = 100. * press / (287.058 * temp);
02007           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
02008             * cos(lat * M_PI / 180.);
02009           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02010
02011           /* Write output... */
02012           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
02013                   tmean[ix][iy] / obscount[ix][iy],
02014                   z, lon, lat, press, temp, mmr, h2o, o3,
02015                   obsmean[ix][iy] / obscount[ix][iy]);
02016         }
02017       }
02018
02019   /* Close file... */
02020   if (t == ctl->t_stop)
02021     fclose(out);
02022 }
```

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*)

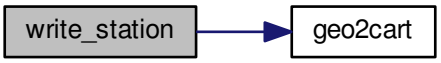
Write station data.

Definition at line 2026 of file libtrac.c.

```
02030           {
02031
02032   static FILE *out;
02033
02034   static double rmax2, t0, t1, x0[3], x1[3];
02035
02036   static int init, ip, iq;
02037
02038   /* Init... */
02039   if (!init) {
02040     init = 1;
02041
02042     /* Write info... */
02043     printf("Write station data: %s\n", filename);
02044
02045     /* Create new file... */
02046     if (!(out = fopen(filename, "w")))
02047       ERRMSG("Cannot create file!");
02048
02049     /* Write header... */
02050     fprintf(out,
02051             "# $1 = time [s]\n"
02052             "# $2 = altitude [km]\n"
02053             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02054     for (iq = 0; iq < ctl->nq; iq++)
02055       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
02056               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02057     fprintf(out, "\n");
02058
02059     /* Set geolocation and search radius... */
02060     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02061     rmax2 = gsl_pow_2(ctl->stat_r);
02062   }
02063
02064   /* Set time interval for output... */
02065   t0 = t - 0.5 * ctl->dt_mod;
02066   t1 = t + 0.5 * ctl->dt_mod;
02067
02068   /* Loop over air parcels... */
02069   for (ip = 0; ip < atm->np; ip++) {
02070
02071     /* Check time... */
02072     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02073       continue;
02074
02075     /* Check station flag... */
02076     if (ctl->qnt_stat >= 0)
02077       if (atm->q[ctl->qnt_stat][ip])
02078         continue;
02079
02080     /* Get Cartesian coordinates... */
02081     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02082
02083     /* Check horizontal distance... */
```

```
02084     if (DIST2(x0, x1) > rmax2)
02085       continue;
02086
02087     /* Set station flag... */
02088     if (ctl->qnt_stat >= 0)
02089       atm->q[ctl->qnt_stat][ip] = 1;
02090
02091     /* Write data... */
02092     fprintf(out, "%.2f %g %g %g",
02093             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02094     for (iq = 0; iq < ctl->nq; iq++) {
02095       fprintf(out, " ");
02096       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02097     }
02098     fprintf(out, "\n");
02099   }
02100
02101   /* Close file... */
02102   if (t == ctl->t_stop)
02103     fclose(out);
02104 }
```

Here is the call graph for this function:



5.14 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) 2013-2018 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 binary data... */
00430     if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
00431       FREAD(&atm->np, int,
00432             1,
00433             in);
00434       FREAD(atm->time, double,
00435             (size_t) atm->np,
00436             in);
00437       FREAD(atm->p, double,
00438             (size_t) atm->np,
00439             in);
00440       FREAD(atm->lon, double,
00441             (size_t) atm->np,
00442             in);
00443       FREAD(atm->lat, double,
00444             (size_t) atm->np,
00445             in);
00446       for (iq = 0; iq < ctl->nq; iq++)
00447         FREAD(atm->q[iq], double,
00448               (size_t) atm->np,
00449               in);
00450     }
00451
00452     /* Read ASCII data... */
00453     else {
00454
00455       /* Read line... */
00456       while (fgets(line, LEN, in)) {
00457
00458         /* Read data... */
00459         TOK(line, tok, "%lg", atm->time[atm->np]);
00460         TOK(NULL, tok, "%lg", atm->p[atm->np]);
00461         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00462         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00463         for (iq = 0; iq < ctl->nq; iq++)
00464           TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00465
00466         /* Convert altitude to pressure... */
00467         atm->p[atm->np] = P(atm->p[atm->np]);
00468
00469         /* Increment data point counter... */
00470         if ((++atm->np) > NP)
00471           ERRMSG("Too many data points!");
00472       }
00473     }
00474
00475     /* Close file... */
00476     fclose(in);
00477
00478     /* Check number of points... */
```

```
00479   if (atm->np < 1)
00480     ERRMSG("Can not read any data!");
00481 }
00482
00483 /*****************************************************************************/
00484
00485 void read_ctl(
00486   const char *filename,
00487   int argc,
00488   char *argv[],
00489   ctl_t * ctl) {
00490
00491   int ip, iq;
00492
00493   /* Write info... */
00494   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00495          "(executable: %s | compiled: %s, %s)\n\n",
00496          argv[0], __DATE__, __TIME__);
00497
00498   /* Initialize quantity indices... */
00499   ctl->qnt_ens = -1;
00500   ctl->qnt_m = -1;
00501   ctl->qnt_r = -1;
00502   ctl->qnt_rho = -1;
00503   ctl->qnt_ps = -1;
00504   ctl->qnt_p = -1;
00505   ctl->qnt_t = -1;
00506   ctl->qnt_u = -1;
00507   ctl->qnt_v = -1;
00508   ctl->qnt_w = -1;
00509   ctl->qnt_h2o = -1;
00510   ctl->qnt_o3 = -1;
00511   ctl->qnt_theta = -1;
00512   ctl->qnt_pv = -1;
00513   ctl->qnt_tice = -1;
00514   ctl->qnt_tsts = -1;
00515   ctl->qnt_tnat = -1;
00516   ctl->qnt_gw_var = -1;
00517   ctl->qnt_stat = -1;
00518
00519   /* Read quantities... */
00520   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00521   if (ctl->nq > NQ)
00522     ERRMSG("Too many quantities!");
00523   for (iq = 0; iq < ctl->nq; iq++) {
00524
00525     /* Read quantity name and format... */
00526     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
00527     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
00528              ctl->qnt_format[iq]);
00529
00530     /* Try to identify quantity... */
00531     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00532       ctl->qnt_ens = iq;
00533       sprintf(ctl->qnt_unit[iq], "-");
00534     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00535       ctl->qnt_m = iq;
00536       sprintf(ctl->qnt_unit[iq], "kg");
00537     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
00538       ctl->qnt_r = iq;
00539       sprintf(ctl->qnt_unit[iq], "m");
00540     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00541       ctl->qnt_rho = iq;
00542       sprintf(ctl->qnt_unit[iq], "kg/m^3");
00543     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00544       ctl->qnt_ps = iq;
00545       sprintf(ctl->qnt_unit[iq], "hPa");
00546     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00547       ctl->qnt_p = iq;
00548       sprintf(ctl->qnt_unit[iq], "hPa");
00549     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00550       ctl->qnt_t = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00553       ctl->qnt_u = iq;
00554       sprintf(ctl->qnt_unit[iq], "m/s");
00555     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00556       ctl->qnt_v = iq;
00557       sprintf(ctl->qnt_unit[iq], "m/s");
00558     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00559       ctl->qnt_w = iq;
00560       sprintf(ctl->qnt_unit[iq], "hPa/s");
00561     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00562       ctl->qnt_h2o = iq;
00563       sprintf(ctl->qnt_unit[iq], "1");
00564     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00565       ctl->qnt_o3 = iq;
```

```
00566       sprintf(ctl->qnt_unit[iq], "1");
00567     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00568       ctl->qnt_theta = iq;
00569       sprintf(ctl->qnt_unit[iq], "K");
00570     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00571       ctl->qnt_pv = iq;
00572       sprintf(ctl->qnt_unit[iq], "PVU");
00573     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00574       ctl->qnt_tice = iq;
00575       sprintf(ctl->qnt_unit[iq], "K");
00576     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00577       ctl->qnt_tsts = iq;
00578       sprintf(ctl->qnt_unit[iq], "K");
00579     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00580       ctl->qnt_tnat = iq;
00581       sprintf(ctl->qnt_unit[iq], "K");
00582     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
00583       ctl->qnt_gw_var = iq;
00584       sprintf(ctl->qnt_unit[iq], "K^2");
00585     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00586       ctl->qnt_stat = iq;
00587       sprintf(ctl->qnt_unit[iq], "-");
00588     } else
00589       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00590   }
00591
00592   /* Time steps of simulation... */
00593   ctl->direction =
00594     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00595   if (ctl->direction != -1 && ctl->direction != 1)
00596     ERRMSG("Set DIRECTION to -1 or 1!");
00597   ctl->t_start =
00598     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00599   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00600   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00601
00602   /* Meteorological data... */
00603   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00604   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00605   if (ctl->met_np > EP)
00606     ERRMSG("Too many levels!");
00607   for (ip = 0; ip < ctl->met_np; ip++)
00608     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00609   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
00610
00611   /* Isosurface parameters... */
00612   ctl->isosurf
00613     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00614   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
00615
00616   /* Diffusion parameters... */
00617   ctl->turb_dx_trop
00618     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00619   ctl->turb_dx_strat
00620     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00621   ctl->turb_dz_trop
00622     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00623   ctl->turb_dz_strat
00624     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00625   ctl->turb_meso =
00626     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00627
00628   /* Life time of particles... */
00629   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00630   ctl->tdec_strat =
00631     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00632
00633   /* PSC analysis... */
00634   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00635   ctl->psc_hno3 =
00636     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00637
00638   /* Gravity wave analysis... */
00639   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
00640
00641   /* Output of atmospheric data... */
00642   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00643   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00644   ctl->atm_dt_out =
00645     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00646   ctl->atm_filter =
00647     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00648   ctl->atm_bin =
00649     (int) scan_ctl(filename, argc, argv, "ATM_BIN", -1, "0", NULL);
00650
```

```
00651   /* Output of CSI data... */
00652   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00653   ctl->csi_dt_out =
00654     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
00655   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00656            ctl->csi_obsfile);
00657   ctl->csi_obsmin =
00658     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00659   ctl->csi_modmin =
00660     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00661   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
00662   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
00663   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00664   ctl->csi_lon0 =
00665     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
00666   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00667   ctl->csi_nx =
00668     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
00669   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
00670   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
00671   ctl->csi_ny =
00672     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00673
00674   /* Output of ensemble data... */
00675   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
00676
00677   /* Output of grid data... */
00678   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00679            ctl->grid_basename);
00680   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00681   ctl->grid_dt_out =
00682     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00683   ctl->grid_sparse =
00684     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00685   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
00686   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00687   ctl->grid_nz =
00688     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00689   ctl->grid_lon0 =
00690     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
00691   ctl->grid_lon1 =
00692     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00693   ctl->grid_nx =
00694     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00695   ctl->grid_lat0 =
00696     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
00697   ctl->grid_lat1 =
00698     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00699   ctl->grid_ny =
00700     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00701
00702   /* Output of profile data... */
00703   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00704            ctl->prof_basename);
00705   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
00706   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00707   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00708   ctl->prof_nz =
00709     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00710   ctl->prof_lon0 =
00711     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
00712   ctl->prof_lon1 =
00713     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00714   ctl->prof_nx =
00715     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00716   ctl->prof_lat0 =
00717     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
00718   ctl->prof_lat1 =
00719     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00720   ctl->prof_ny =
00721     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00722
00723   /* Output of station data... */
00724   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00725            ctl->stat_basename);
00726   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
00727   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
00728   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00729 }
00730
00731 /*****************************************************************************/
00732
00733 void read_met(
```

```
00734   ctl_t * ctl,
00735   char *filename,
00736   met_t * met) {
00737
00738   char cmd[LEN], levname[LEN], tstr[10];
00739
00740   static float help[EX * EY];
00741
00742   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00743
00744   size_t np, nx, ny;
00745
00746   /* Write info... */
00747   printf("Read meteorological data: %s\n", filename);
00748
00749   /* Get time from filename... */
00750   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00751   year = atoi(tstr);
00752   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00753   mon = atoi(tstr);
00754   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
00755   day = atoi(tstr);
00756   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00757   hour = atoi(tstr);
00758   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00759
00760   /* Open netCDF file... */
00761   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00762
00763     /* Try to stage meteo file... */
00764     STOP_TIMER(TIMER_INPUT);
00765     START_TIMER(TIMER_STAGE);
00766     if (ctl->met_stage[0] != '-') {
00767       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
00768               year, mon, day, hour, filename);
00769       if (system(cmd) != 0)
00770         ERRMSG("Error while staging meteo data!");
00771     }
00772     STOP_TIMER(TIMER_STAGE);
00773     START_TIMER(TIMER_INPUT);
00774
00775     /* Try to open again... */
00776     NC(nc_open(filename, NC_NOWRITE, &ncid));
00777   }
00778
00779   /* Get dimensions... */
00780   NC(nc_inq_dimid(ncid, "lon", &dimid));
00781   NC(nc_inq_dimlen(ncid, dimid, &nx));
00782   if (nx < 2 || nx > EX)
00783     ERRMSG("Number of longitudes out of range!");
00784
00785   NC(nc_inq_dimid(ncid, "lat", &dimid));
00786   NC(nc_inq_dimlen(ncid, dimid, &ny));
00787   if (ny < 2 || ny > EY)
00788     ERRMSG("Number of latitudes out of range!");
00789
00790   sprintf(levname, "lev");
00791   NC(nc_inq_dimid(ncid, levname, &dimid));
00792   NC(nc_inq_dimlen(ncid, dimid, &np));
00793   if (np == 1) {
00794     sprintf(levname, "lev_2");
00795     NC(nc_inq_dimid(ncid, levname, &dimid));
00796     NC(nc_inq_dimlen(ncid, dimid, &np));
00797   }
00798   if (np < 2 || np > EP)
00799     ERRMSG("Number of levels out of range!");
00800
00801   /* Store dimensions... */
00802   met->np = (int) np;
00803   met->nx = (int) nx;
00804   met->ny = (int) ny;
00805
00806   /* Get horizontal grid... */
00807   NC(nc_inq_varid(ncid, "lon", &varid));
00808   NC(nc_get_var_double(ncid, varid, met->lon));
00809   NC(nc_inq_varid(ncid, "lat", &varid));
00810   NC(nc_get_var_double(ncid, varid, met->lat));
00811
00812   /* Read meteorological data... */
00813   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00814   read_met_help(ncid, "u", "U", met, met->u, 1.0);
00815   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00816   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00817   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00818   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00819
00820   /* Meteo data on pressure levels... */
```

```
00821   if (ctl->met_np <= 0) {
00822
00823     /* Read pressure levels from file... */
00824     NC(nc_inq_varid(ncid, levname, &varid));
00825     NC(nc_get_var_double(ncid, varid, met->p));
00826     for (ip = 0; ip < met->np; ip++)
00827       met->p[ip] /= 100.;
00828
00829     /* Extrapolate data for lower boundary... */
00830     read_met_extrapolate(met);
00831   }
00832
00833   /* Meteo data on model levels... */
00834   else {
00835
00836     /* Read pressure data from file... */
00837     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00838
00839     /* Interpolate from model levels to pressure levels... */
00840     read_met_ml2pl(ctl, met, met->t);
00841     read_met_ml2pl(ctl, met, met->u);
00842     read_met_ml2pl(ctl, met, met->v);
00843     read_met_ml2pl(ctl, met, met->w);
00844     read_met_ml2pl(ctl, met, met->h2o);
00845     read_met_ml2pl(ctl, met, met->o3);
00846
00847     /* Set pressure levels... */
00848     met->np = ctl->met_np;
00849     for (ip = 0; ip < met->np; ip++)
00850       met->p[ip] = ctl->met_p[ip];
00851   }
00852
00853   /* Check ordering of pressure levels... */
00854   for (ip = 1; ip < met->np; ip++)
00855     if (met->p[ip - 1] < met->p[ip])
00856       ERRMSG("Pressure levels must be descending!");
00857
00858   /* Read surface pressure... */
00859   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
00860       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00861     NC(nc_get_var_float(ncid, varid, help));
00862     for (iy = 0; iy < met->ny; iy++)
00863       for (ix = 0; ix < met->nx; ix++)
00864         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00865   } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
00866              || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00867     NC(nc_get_var_float(ncid, varid, help));
00868     for (iy = 0; iy < met->ny; iy++)
00869       for (ix = 0; ix < met->nx; ix++)
00870         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00871   } else
00872     for (ix = 0; ix < met->nx; ix++)
00873       for (iy = 0; iy < met->ny; iy++)
00874         met->ps[ix][iy] = met->p[0];
00875
00876   /* Create periodic boundary conditions... */
00877   read_met_periodic(met);
00878
00879   /* Close file... */
00880   NC(nc_close(ncid));
00881 }
00882
00883 /*****************************************************************************/
00884
00885 void read_met_extrapolate(
00886   met_t * met) {
00887
00888   int ip, ip0, ix, iy;
00889
00890   /* Loop over columns... */
00891   for (ix = 0; ix < met->nx; ix++)
00892     for (iy = 0; iy < met->ny; iy++) {
00893
00894       /* Find lowest valid data point... */
00895       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00896         if (!gsl_finite(met->t[ix][iy][ip0])
00897             || !gsl_finite(met->u[ix][iy][ip0])
00898             || !gsl_finite(met->v[ix][iy][ip0])
00899             || !gsl_finite(met->w[ix][iy][ip0]))
00900           break;
00901
00902       /* Extrapolate... */
00903       for (ip = ip0; ip >= 0; ip--) {
00904         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00905         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00906         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00907         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
```

```
00908         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00909         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00910       }
00911     }
00912 }
00913
00914 /*****************************************************************************/
00915
00916 void read_met_help(
00917   int ncid,
00918   char *varname,
00919   char *varname2,
00920   met_t * met,
00921   float dest[EX][EY][EP],
00922   float scl) {
00923
00924   static float help[EX * EY * EP];
00925
00926   int ip, ix, iy, n = 0, varid;
00927
00928   /* Check if variable exists... */
00929   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00930     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00931       return;
00932
00933   /* Read data... */
00934   NC(nc_get_var_float(ncid, varid, help));
00935
00936   /* Copy and check data... */
00937   for (ip = 0; ip < met->np; ip++)
00938     for (iy = 0; iy < met->ny; iy++)
00939       for (ix = 0; ix < met->nx; ix++) {
00940         dest[ix][iy][ip] = scl * help[n++];
00941         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00942           dest[ix][iy][ip] = GSL_NAN;
00943       }
00944 }
00945
00946 /*****************************************************************************/
00947
00948 void read_met_ml2pl(
00949   ctl_t * ctl,
00950   met_t * met,
00951   float var[EX][EY][EP]) {
00952
00953   double aux[EP], p[EP], pt;
00954
00955   int ip, ip2, ix, iy;
00956
00957   /* Loop over columns... */
00958   for (ix = 0; ix < met->nx; ix++)
00959     for (iy = 0; iy < met->ny; iy++) {
00960
00961       /* Copy pressure profile... */
00962       for (ip = 0; ip < met->np; ip++)
00963         p[ip] = met->pl[ix][iy][ip];
00964
00965       /* Interpolate... */
00966       for (ip = 0; ip < ctl->met_np; ip++) {
00967         pt = ctl->met_p[ip];
00968         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00969           pt = p[0];
00970         else if ((pt > p[met->np - 1] && p[1] > p[0])
00971                  || (pt < p[met->np - 1] && p[1] < p[0]))
00972           pt = p[met->np - 1];
00973         ip2 = locate(p, met->np, pt);
00974         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00975                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00976       }
00977
00978       /* Copy data... */
00979       for (ip = 0; ip < ctl->met_np; ip++)
00980         var[ix][iy][ip] = (float) aux[ip];
00981     }
00982 }
00983
00984 /*****************************************************************************/
00985
00986 void read_met_periodic(
00987   met_t * met) {
00988
00989   int ip, iy;
00990
00991   /* Check longitudes... */
00992   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00993              + met->lon[1] - met->lon[0] - 360) < 0.01))
00994     return;
```

```
00995
00996   /* Increase longitude counter... */
00997   if ((++met->nx) > EX)
00998     ERRMSG("Cannot create periodic boundary conditions!");
00999
01000   /* Set longitude... */
01001   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01002
01003   /* Loop over latitudes and pressure levels... */
01004   for (iy = 0; iy < met->ny; iy++)
01005     for (ip = 0; ip < met->np; ip++) {
01006       met->ps[met->nx - 1][iy] = met->ps[0][iy];
01007       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01008       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01009       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01010       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01011       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01012       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01013     }
01014 }
01015
01016 /*****************************************************************************/
01017
01018 double scan_ctl(
01019   const char *filename,
01020   int argc,
01021   char *argv[],
01022   const char *varname,
01023   int arridx,
01024   const char *defvalue,
01025   char *value) {
01026
01027   FILE *in = NULL;
01028
01029   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01030     msg[LEN], rvarname[LEN], rval[LEN];
01031
01032   int contain = 0, i;
01033
01034   /* Open file... */
01035   if (filename[strlen(filename) - 1] != '-')
01036     if (!(in = fopen(filename, "r")))
01037       ERRMSG("Cannot open file!");
01038
01039   /* Set full variable name... */
01040   if (arridx >= 0) {
01041     sprintf(fullname1, "%s[%d]", varname, arridx);
01042     sprintf(fullname2, "%s[*]", varname);
01043   } else {
01044     sprintf(fullname1, "%s", varname);
01045     sprintf(fullname2, "%s", varname);
01046   }
01047
01048   /* Read data... */
01049   if (in != NULL)
01050     while (fgets(line, LEN, in))
01051       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
01052         if (strcasecmp(rvarname, fullname1) == 0 ||
01053             strcasecmp(rvarname, fullname2) == 0) {
01054           contain = 1;
01055           break;
01056         }
01057   for (i = 1; i < argc - 1; i++)
01058     if (strcasecmp(argv[i], fullname1) == 0 ||
01059         strcasecmp(argv[i], fullname2) == 0) {
01060       sprintf(rval, "%s", argv[i + 1]);
01061       contain = 1;
01062       break;
01063     }
01064
01065   /* Close file... */
01066   if (in != NULL)
01067     fclose(in);
01068
01069   /* Check for missing variables... */
01070   if (!contain) {
01071     if (strlen(defvalue) > 0)
01072       sprintf(rval, "%s", defvalue);
01073     else {
01074       sprintf(msg, "Missing variable %s!\n", fullname1);
01075       ERRMSG(msg);
01076     }
01077   }
01078
01079   /* Write info... */
01080   printf("%s = %s\n", fullname1, rval);
```

```
01081
01082   /* Return values... */
01083   if (value != NULL)
01084     sprintf(value, "%s", rval);
01085   return atof(rval);
01086 }
01087
01088 /*****************************************************************************/
01089
01090 void time2jsec(
01091   int year,
01092   int mon,
01093   int day,
01094   int hour,
01095   int min,
01096   int sec,
01097   double remain,
01098   double *jsec) {
01099
01100   struct tm t0, t1;
01101
01102   t0.tm_year = 100;
01103   t0.tm_mon = 0;
01104   t0.tm_mday = 1;
01105   t0.tm_hour = 0;
01106   t0.tm_min = 0;
01107   t0.tm_sec = 0;
01108
01109   t1.tm_year = year - 1900;
01110   t1.tm_mon = mon - 1;
01111   t1.tm_mday = day;
01112   t1.tm_hour = hour;
01113   t1.tm_min = min;
01114   t1.tm_sec = sec;
01115
01116   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01117 }
01118
01119 /*****************************************************************************/
01120
01121 void timer(
01122   const char *name,
01123   int id,
01124   int mode) {
01125
01126   static double starttime[NTIMER], runtime[NTIMER];
01127
01128   /* Check id... */
01129   if (id < 0 || id >= NTIMER)
01130     ERRMSG("Too many timers!");
01131
01132   /* Start timer... */
01133   if (mode == 1) {
01134     if (starttime[id] <= 0)
01135       starttime[id] = omp_get_wtime();
01136     else
01137       ERRMSG("Timer already started!");
01138   }
01139
01140   /* Stop timer... */
01141   else if (mode == 2) {
01142     if (starttime[id] > 0) {
01143       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01144       starttime[id] = -1;
01145     }
01146   }
01147
01148   /* Print timer... */
01149   else if (mode == 3)
01150     printf("%s = %.3f s\n", name, runtime[id]);
01151 }
01152
01153 /*****************************************************************************/
01154
01155 double tropopause(
01156   double t,
01157   double lat) {
01158
01159   static double doys[12]
01160   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01161
01162   static double lats[73]
01163     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01164     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01165     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01166     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01167     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
```

```
01168     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01169     75, 77.5, 80, 82.5, 85, 87.5, 90
01170   };
01171
01172   static double tps[12][73]
01173     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01174          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01175          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01176          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01177          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01178          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01179          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01180          275.3, 275.6, 275.4, 274.1, 273.5},
01181   {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01182    300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01183    150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01184    98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01185    98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01186    220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01187    284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01188    287.5, 286.2, 285.8},
01189   {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01190    297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01191    161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01192    100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01193    99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01194    186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01195    279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01196    304.3, 304.9, 306, 306.6, 306.2, 306},
01197   {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01198    290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01199    195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01200    102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01201    99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01202    148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01203    263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01204    315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01205   {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01206    260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01207    205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01208    101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01209    102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01210    165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01211    273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01212    325.3, 325.8, 325.8},
01213   {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01214    222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01215    228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01216    105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01217    106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01218    127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01219    251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01220    308.5, 312.2, 313.1, 313.3},
01221   {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01222    187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01223    235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01224    110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01225    111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01226    117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01227    224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01228    275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01229   {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01230    185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01231    233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01232    110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01233    112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01234    120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01235    230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01236    278.2, 282.6, 287.4, 290.9, 292.5, 293},
01237   {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01238    183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01239    243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01240    114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01241    110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01242    114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01243    203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01244    276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01245   {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01246    215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01247    237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01248    111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01249    106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01250    112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01251    206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01252    279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01253    305.1},
01254   {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
```

```
01255     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01256     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01257     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01258     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
01259     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01260     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01261     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01262    {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01263     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01264     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01265     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01266     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01267     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01268     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01269     281.7, 281.1, 281.2}
01270    };
01271
01272    double doy, p0, p1, pt;
01273
01274    int imon, ilat;
01275
01276    /* Get day of year... */
01277    doy = fmod(t / 86400., 365.25);
01278    while (doy < 0)
01279      doy += 365.25;
01280
01281    /* Get indices... */
01282    imon = locate(doys, 12, doy);
01283    ilat = locate(lats, 73, lat);
01284
01285    /* Get tropopause pressure... */
01286    p0 = LIN(lats[ilat], tps[imon][ilat],
01287             lats[ilat + 1], tps[imon][ilat + 1], lat);
01288    p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01289             lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01290    pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01291
01292    /* Return tropopause pressure... */
01293    return pt;
01294  }
01295
01296  /*****************************************************************************/
01297
01298  void write_atm(
01299    const char *filename,
01300    ctl_t * ctl,
01301    atm_t * atm,
01302    double t) {
01303
01304    FILE *in, *out;
01305
01306    char line[LEN];
01307
01308    double r, t0, t1;
01309
01310    int ip, iq, year, mon, day, hour, min, sec;
01311
01312    /* Set time interval for output... */
01313    t0 = t - 0.5 * ctl->dt_mod;
01314    t1 = t + 0.5 * ctl->dt_mod;
01315
01316    /* Check if gnuplot output is requested... */
01317    if (ctl->atm_gpfile[0] != '-') {
01318
01319      /* Write info... */
01320      printf("Plot atmospheric data: %s.png\n", filename);
01321
01322      /* Create gnuplot pipe... */
01323      if (!(out = popen("gnuplot", "w")))
01324        ERRMSG("Cannot create pipe to gnuplot!");
01325
01326      /* Set plot filename... */
01327      fprintf(out, "set out \"%s.png\"\n", filename);
01328
01329      /* Set time string... */
01330      jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01331      fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01332              year, mon, day, hour, min);
01333
01334      /* Dump gnuplot file to pipe... */
01335      if (!(in = fopen(ctl->atm_gpfile, "r")))
01336        ERRMSG("Cannot open file!");
01337      while (fgets(line, LEN, in))
01338        fprintf(out, "%s", line);
01339      fclose(in);
01340    }
01341
```

```
01342   else {
01343
01344     /* Write info... */
01345     printf("Write atmospheric data: %s\n", filename);
01346
01347     /* Create file... */
01348     if (!(out = fopen(filename, "w")))
01349       ERRMSG("Cannot create file!");
01350   }
01351
01352   /* Write binary data... */
01353   if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
01354     FWRITE(&atm->np, int,
01355            1,
01356            out);
01357     FWRITE(atm->time, double,
01358              (size_t) atm->np,
01359            out);
01360     FWRITE(atm->p, double,
01361              (size_t) atm->np,
01362            out);
01363     FWRITE(atm->lon, double,
01364              (size_t) atm->np,
01365            out);
01366     FWRITE(atm->lat, double,
01367              (size_t) atm->np,
01368            out);
01369     for (iq = 0; iq < ctl->nq; iq++)
01370       FWRITE(atm->q[iq], double,
01371                (size_t) atm->np,
01372              out);
01373   }
01374
01375   /* Write ASCII data... */
01376   else {
01377
01378     /* Write header... */
01379     fprintf(out,
01380             "# $1 = time [s]\n"
01381             "# $2 = altitude [km]\n"
01382             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01383     for (iq = 0; iq < ctl->nq; iq++)
01384       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01385               ctl->qnt_unit[iq]);
01386     fprintf(out, "\n");
01387
01388     /* Write data... */
01389     for (ip = 0; ip < atm->np; ip++) {
01390
01391       /* Check time... */
01392       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01393         continue;
01394
01395       /* Write output... */
01396       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01397               atm->lon[ip], atm->lat[ip]);
01398       for (iq = 0; iq < ctl->nq; iq++) {
01399         fprintf(out, " ");
01400         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01401       }
01402       fprintf(out, "\n");
01403     }
01404   }
01405
01406   /* Close file... */
01407   fclose(out);
01408 }
01409
01410 /*****************************************************************************/
01411
01412 void write_csi(
01413   const char *filename,
01414   ctl_t * ctl,
01415   atm_t * atm,
01416   double t) {
01417
01418   static FILE *in, *out;
01419
01420   static char line[LEN];
01421
01422   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01423     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01424
01425   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01426
01427   /* Init... */
01428   if (!init) {
```

```
01429     init = 1;
01430
01431     /* Check quantity index for mass... */
01432     if (ctl->qnt_m < 0)
01433       ERRMSG("Need quantity mass to analyze CSI!");
01434
01435     /* Open observation data file... */
01436     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01437     if (!(in = fopen(ctl->csi_obsfile, "r")))
01438       ERRMSG("Cannot open file!");
01439
01440     /* Create new file... */
01441     printf("Write CSI data: %s\n", filename);
01442     if (!(out = fopen(filename, "w")))
01443       ERRMSG("Cannot create file!");
01444
01445     /* Write header... */
01446     fprintf(out,
01447             "# $1 = time [s]\n"
01448             "# $2 = number of hits (cx)\n"
01449             "# $3 = number of misses (cy)\n"
01450             "# $4 = number of false alarms (cz)\n"
01451             "# $5 = number of observations (cx + cy)\n"
01452             "# $6 = number of forecasts (cx + cz)\n"
01453             "# $7 = bias (forecasts/observations) [%%]\n"
01454             "# $8 = probability of detection (POD) [%%]\n"
01455             "# $9 = false alarm rate (FAR) [%%]\n"
01456             "# $10 = critical success index (CSI) [%%]\n\n");
01457   }
01458
01459   /* Set time interval... */
01460   t0 = t - 0.5 * ctl->dt_mod;
01461   t1 = t + 0.5 * ctl->dt_mod;
01462
01463   /* Initialize grid cells... */
01464   for (ix = 0; ix < ctl->csi_nx; ix++)
01465     for (iy = 0; iy < ctl->csi_ny; iy++)
01466       for (iz = 0; iz < ctl->csi_nz; iz++)
01467         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01468
01469   /* Read data... */
01470   while (fgets(line, LEN, in)) {
01471
01472     /* Read data... */
01473     if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01474         5)
01475       continue;
01476
01477     /* Check time... */
01478     if (rt < t0)
01479       continue;
01480     if (rt > t1)
01481       break;
01482
01483     /* Calculate indices... */
01484     ix = (int) ((rlon - ctl->csi_lon0)
01485                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01486     iy = (int) ((rlat - ctl->csi_lat0)
01487                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01488     iz = (int) ((rz - ctl->csi_z0)
01489                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01490
01491     /* Check indices... */
01492     if (ix < 0 || ix >= ctl->csi_nx ||
01493         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01494       continue;
01495
01496     /* Get mean observation index... */
01497     obsmean[ix][iy][iz] += robs;
01498     obscount[ix][iy][iz]++;
01499   }
01500
01501   /* Analyze model data... */
01502   for (ip = 0; ip < atm->np; ip++) {
01503
01504     /* Check time... */
01505     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01506       continue;
01507
01508     /* Get indices... */
01509     ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01510                 / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01511     iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01512                 / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01513     iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01514                 / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01515
```

```
01516     /* Check indices... */
01517     if (ix < 0 || ix >= ctl->csi_nx ||
01518         iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01519       continue;
01520
01521     /* Get total mass in grid cell... */
01522     modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01523   }
01524
01525   /* Analyze all grid cells... */
01526   for (ix = 0; ix < ctl->csi_nx; ix++)
01527     for (iy = 0; iy < ctl->csi_ny; iy++)
01528       for (iz = 0; iz < ctl->csi_nz; iz++) {
01529
01530         /* Calculate mean observation index... */
01531         if (obscount[ix][iy][iz] > 0)
01532           obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01533
01534         /* Calculate column density... */
01535         if (modmean[ix][iy][iz] > 0) {
01536           dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01537           dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01538           lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01539           area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01540             * cos(lat * M_PI / 180.);
01541           modmean[ix][iy][iz] /= (1e6 * area);
01542         }
01543
01544         /* Calculate CSI... */
01545         if (obscount[ix][iy][iz] > 0) {
01546           if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01547               modmean[ix][iy][iz] >= ctl->csi_modmin)
01548             cx++;
01549           else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01550                    modmean[ix][iy][iz] < ctl->csi_modmin)
01551             cy++;
01552           else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
01553                    modmean[ix][iy][iz] >= ctl->csi_modmin)
01554             cz++;
01555         }
01556       }
01557
01558   /* Write output... */
01559   if (fmod(t, ctl->csi_dt_out) == 0) {
01560
01561     /* Write... */
01562     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01563             t, cx, cy, cz, cx + cy, cx + cz,
01564             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01565             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01566             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01567             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01568
01569     /* Set counters to zero... */
01570     cx = cy = cz = 0;
01571   }
01572
01573   /* Close file... */
01574   if (t == ctl->t_stop)
01575     fclose(out);
01576 }
01577
01578 /*****************************************************************************/
01579
01580 void write_ens(
01581   const char *filename,
01582   ctl_t * ctl,
01583   atm_t * atm,
01584   double t) {
01585
01586   static FILE *out;
01587
01588   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01589     t0, t1, x[NENS][3], xm[3];
01590
01591   static int init, ip, iq;
01592
01593   static size_t i, n;
01594
01595   /* Init... */
01596   if (!init) {
01597     init = 1;
01598
01599     /* Check quantities... */
01600     if (ctl->qnt_ens < 0)
01601       ERRMSG("Missing ensemble IDs!");
01602
```

```
01603     /* Create new file... */
01604     printf("Write ensemble data: %s\n", filename);
01605     if (!(out = fopen(filename, "w")))
01606       ERRMSG("Cannot create file!");
01607
01608     /* Write header... */
01609     fprintf(out,
01610             "# $1 = time [s]\n"
01611             "# $2 = altitude [km]\n"
01612             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01613     for (iq = 0; iq < ctl->nq; iq++)
01614       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01615               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01616     for (iq = 0; iq < ctl->nq; iq++)
01617       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01618               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01619     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01620   }
01621
01622   /* Set time interval... */
01623   t0 = t - 0.5 * ctl->dt_mod;
01624   t1 = t + 0.5 * ctl->dt_mod;
01625
01626   /* Init... */
01627   ens = GSL_NAN;
01628   n = 0;
01629
01630   /* Loop over air parcels... */
01631   for (ip = 0; ip < atm->np; ip++) {
01632
01633     /* Check time... */
01634     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01635       continue;
01636
01637     /* Check ensemble id... */
01638     if (atm->q[ctl->qnt_ens][ip] != ens) {
01639
01640       /* Write results... */
01641       if (n > 0) {
01642
01643         /* Get mean position... */
01644         xm[0] = xm[1] = xm[2] = 0;
01645         for (i = 0; i < n; i++) {
01646           xm[0] += x[i][0] / (double) n;
01647           xm[1] += x[i][1] / (double) n;
01648           xm[2] += x[i][2] / (double) n;
01649         }
01650         cart2geo(xm, &dummy, &lon, &lat);
01651         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01652                 lat);
01653
01654         /* Get quantity statistics... */
01655         for (iq = 0; iq < ctl->nq; iq++) {
01656           fprintf(out, " ");
01657           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01658         }
01659         for (iq = 0; iq < ctl->nq; iq++) {
01660           fprintf(out, " ");
01661           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01662         }
01663         fprintf(out, " %lu\n", n);
01664       }
01665
01666       /* Init new ensemble... */
01667       ens = atm->q[ctl->qnt_ens][ip];
01668       n = 0;
01669     }
01670
01671     /* Save data... */
01672     p[n] = atm->p[ip];
01673     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01674     for (iq = 0; iq < ctl->nq; iq++)
01675       q[iq][n] = atm->q[iq][ip];
01676     if ((++n) >= NENS)
01677       ERRMSG("Too many data points!");
01678   }
01679
01680   /* Write results... */
01681   if (n > 0) {
01682
01683     /* Get mean position... */
01684     xm[0] = xm[1] = xm[2] = 0;
01685     for (i = 0; i < n; i++) {
01686       xm[0] += x[i][0] / (double) n;
01687       xm[1] += x[i][1] / (double) n;
01688       xm[2] += x[i][2] / (double) n;
01689     }
```

```
01690     cart2geo(xm, &dummy, &lon, &lat);
01691     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01692
01693     /* Get quantity statistics... */
01694     for (iq = 0; iq < ctl->nq; iq++) {
01695       fprintf(out, " ");
01696       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01697     }
01698     for (iq = 0; iq < ctl->nq; iq++) {
01699       fprintf(out, " ");
01700       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01701     }
01702     fprintf(out, " %lu\n", n);
01703   }
01704
01705   /* Close file... */
01706   if (t == ctl->t_stop)
01707     fclose(out);
01708 }
01709
01710 /*****************************************************************************/
01711
01712 void write_grid(
01713   const char *filename,
01714   ctl_t * ctl,
01715   met_t * met0,
01716   met_t * met1,
01717   atm_t * atm,
01718   double t) {
01719
01720   FILE *in, *out;
01721
01722   char line[LEN];
01723
01724   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01725     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01726
01727   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01728
01729   /* Check dimensions... */
01730   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01731     ERRMSG("Grid dimensions too large!");
01732
01733   /* Check quantity index for mass... */
01734   if (ctl->qnt_m < 0)
01735     ERRMSG("Need quantity mass to write grid data!");
01736
01737   /* Set time interval for output... */
01738   t0 = t - 0.5 * ctl->dt_mod;
01739   t1 = t + 0.5 * ctl->dt_mod;
01740
01741   /* Set grid box size... */
01742   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01743   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01744   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01745
01746   /* Initialize grid... */
01747   for (ix = 0; ix < ctl->grid_nx; ix++)
01748     for (iy = 0; iy < ctl->grid_ny; iy++)
01749       for (iz = 0; iz < ctl->grid_nz; iz++)
01750         grid_m[ix][iy][iz] = 0;
01751
01752   /* Average data... */
01753   for (ip = 0; ip < atm->np; ip++)
01754     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01755
01756       /* Get index... */
01757       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01758       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01759       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01760
01761       /* Check indices... */
01762       if (ix < 0 || ix >= ctl->grid_nx ||
01763           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01764         continue;
01765
01766       /* Add mass... */
01767       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01768     }
01769
01770   /* Check if gnuplot output is requested... */
01771   if (ctl->grid_gpfile[0] != '-') {
01772
01773     /* Write info... */
01774     printf("Plot grid data: %s.png\n", filename);
01775
01776     /* Create gnuplot pipe... */
```

```
01777     if (!(out = popen("gnuplot", "w")))
01778       ERRMSG("Cannot create pipe to gnuplot!");
01779
01780     /* Set plot filename... */
01781     fprintf(out, "set out \"%s.png\"\n", filename);
01782
01783     /* Set time string... */
01784     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01785     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01786             year, mon, day, hour, min);
01787
01788     /* Dump gnuplot file to pipe... */
01789     if (!(in = fopen(ctl->grid_gpfile, "r")))
01790       ERRMSG("Cannot open file!");
01791     while (fgets(line, LEN, in))
01792       fprintf(out, "%s", line);
01793     fclose(in);
01794   }
01795
01796   else {
01797
01798     /* Write info... */
01799     printf("Write grid data: %s\n", filename);
01800
01801     /* Create file... */
01802     if (!(out = fopen(filename, "w")))
01803       ERRMSG("Cannot create file!");
01804   }
01805
01806   /* Write header... */
01807   fprintf(out,
01808           "# $1 = time [s]\n"
01809           "# $2 = altitude [km]\n"
01810           "# $3 = longitude [deg]\n"
01811           "# $4 = latitude [deg]\n"
01812           "# $5 = surface area [km^2]\n"
01813           "# $6 = layer width [km]\n"
01814           "# $7 = temperature [K]\n"
01815           "# $8 = column density [kg/m^2]\n"
01816           "# $9 = mass mixing ratio [1]\n\n");
01817
01818   /* Write data... */
01819   for (ix = 0; ix < ctl->grid_nx; ix++) {
01820     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01821       fprintf(out, "\n");
01822     for (iy = 0; iy < ctl->grid_ny; iy++) {
01823       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01824         fprintf(out, "\n");
01825       for (iz = 0; iz < ctl->grid_nz; iz++)
01826         if (!ctl->grid_sparse
01827             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01828
01829           /* Set coordinates... */
01830           z = ctl->grid_z0 + dz * (iz + 0.5);
01831           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01832           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01833
01834           /* Get pressure and temperature... */
01835           press = P(z);
01836           intpol_met_time(met0, met1, t, press, lon, lat,
01837                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01838
01839           /* Calculate surface area... */
01840           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01841             * cos(lat * M_PI / 180.);
01842
01843           /* Calculate column density... */
01844           cd = grid_m[ix][iy][iz] / (1e6 * area);
01845
01846           /* Calculate mass mixing ratio... */
01847           rho_air = 100. * press / (287.058 * temp);
01848           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01849
01850           /* Write output... */
01851           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01852                   t, z, lon, lat, area, dz, temp, cd, mmr);
01853         }
01854     }
01855   }
01856
01857   /* Close file... */
01858   fclose(out);
01859 }
01860
01861 /*****************************************************************************/
01862
01863 void write_prof(
```

```
01864   const char *filename,
01865   ctl_t * ctl,
01866   met_t * met0,
01867   met_t * met1,
01868   atm_t * atm,
01869   double t) {
01870
01871   static FILE *in, *out;
01872
01873   static char line[LEN];
01874
01875   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01876     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01877     press, temp, rho_air, mmr, h2o, o3;
01878
01879   static int init, obscount[GX][GY], ip, ix, iy, iz;
01880
01881   /* Init... */
01882   if (!init) {
01883     init = 1;
01884
01885     /* Check quantity index for mass... */
01886     if (ctl->qnt_m < 0)
01887       ERRMSG("Need quantity mass!");
01888
01889     /* Check dimensions... */
01890     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01891       ERRMSG("Grid dimensions too large!");
01892
01893     /* Open observation data file... */
01894     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01895     if (!(in = fopen(ctl->prof_obsfile, "r")))
01896       ERRMSG("Cannot open file!");
01897
01898     /* Create new file... */
01899     printf("Write profile data: %s\n", filename);
01900     if (!(out = fopen(filename, "w")))
01901       ERRMSG("Cannot create file!");
01902
01903     /* Write header... */
01904     fprintf(out,
01905             "# $1  = time [s]\n"
01906             "# $2  = altitude [km]\n"
01907             "# $3  = longitude [deg]\n"
01908             "# $4  = latitude [deg]\n"
01909             "# $5  = pressure [hPa]\n"
01910             "# $6  = temperature [K]\n"
01911             "# $7  = mass mixing ratio [1]\n"
01912             "# $8  = H2O volume mixing ratio [1]\n"
01913             "# $9  = O3 volume mixing ratio [1]\n"
01914             "# $10 = mean BT index [K]\n");
01915
01916     /* Set grid box size... */
01917     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01918     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01919     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01920   }
01921
01922   /* Set time interval... */
01923   t0 = t - 0.5 * ctl->dt_mod;
01924   t1 = t + 0.5 * ctl->dt_mod;
01925
01926   /* Initialize... */
01927   for (ix = 0; ix < ctl->prof_nx; ix++)
01928     for (iy = 0; iy < ctl->prof_ny; iy++) {
01929       obsmean[ix][iy] = 0;
01930       obscount[ix][iy] = 0;
01931       tmean[ix][iy] = 0;
01932       for (iz = 0; iz < ctl->prof_nz; iz++)
01933         mass[ix][iy][iz] = 0;
01934     }
01935
01936   /* Read data... */
01937   while (fgets(line, LEN, in)) {
01938
01939     /* Read data... */
01940     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01941       continue;
01942
01943     /* Check time... */
01944     if (rt < t0)
01945       continue;
01946     if (rt > t1)
01947       break;
01948
01949     /* Calculate indices... */
01950     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
```

```
01951     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01952
01953     /* Check indices... */
01954     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01955       continue;
01956
01957     /* Get mean observation index... */
01958     obsmean[ix][iy] += robs;
01959     tmean[ix][iy] += rt;
01960     obscount[ix][iy]++;
01961   }
01962
01963   /* Analyze model data... */
01964   for (ip = 0; ip < atm->np; ip++) {
01965
01966     /* Check time... */
01967     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01968       continue;
01969
01970     /* Get indices... */
01971     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01972     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01973     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01974
01975     /* Check indices... */
01976     if (ix < 0 || ix >= ctl->prof_nx ||
01977         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01978       continue;
01979
01980     /* Get total mass in grid cell... */
01981     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01982   }
01983
01984   /* Extract profiles... */
01985   for (ix = 0; ix < ctl->prof_nx; ix++)
01986     for (iy = 0; iy < ctl->prof_ny; iy++)
01987       if (obscount[ix][iy] > 0) {
01988
01989         /* Write output... */
01990         fprintf(out, "\n");
01991
01992         /* Loop over altitudes... */
01993         for (iz = 0; iz < ctl->prof_nz; iz++) {
01994
01995           /* Set coordinates... */
01996           z = ctl->prof_z0 + dz * (iz + 0.5);
01997           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01998           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01999
02000           /* Get meteorological data... */
02001           press = P(z);
02002           intpol_met_time(met0, met1, t, press, lon, lat,
02003                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
02004
02005           /* Calculate mass mixing ratio... */
02006           rho_air = 100. * press / (287.058 * temp);
02007           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
02008             * cos(lat * M_PI / 180.);
02009           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02010
02011           /* Write output... */
02012           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
02013                   tmean[ix][iy] / obscount[ix][iy],
02014                   z, lon, lat, press, temp, mmr, h2o, o3,
02015                   obsmean[ix][iy] / obscount[ix][iy]);
02016         }
02017       }
02018
02019   /* Close file... */
02020   if (t == ctl->t_stop)
02021     fclose(out);
02022 }
02023
02024 /*****************************************************************************/
02025
02026 void write_station(
02027   const char *filename,
02028   ctl_t * ctl,
02029   atm_t * atm,
02030   double t) {
02031
02032   static FILE *out;
02033
02034   static double rmax2, t0, t1, x0[3], x1[3];
02035
02036   static int init, ip, iq;
02037
```

```
02038   /* Init... */
02039   if (!init) {
02040     init = 1;
02041
02042     /* Write info... */
02043     printf("Write station data: %s\n", filename);
02044
02045     /* Create new file... */
02046     if (!(out = fopen(filename, "w")))
02047       ERRMSG("Cannot create file!");
02048
02049     /* Write header... */
02050     fprintf(out,
02051             "# $1 = time [s]\n"
02052             "# $2 = altitude [km]\n"
02053             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02054     for (iq = 0; iq < ctl->nq; iq++)
02055       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
02056               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02057     fprintf(out, "\n");
02058
02059     /* Set geolocation and search radius... */
02060     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02061     rmax2 = gsl_pow_2(ctl->stat_r);
02062   }
02063
02064   /* Set time interval for output... */
02065   t0 = t - 0.5 * ctl->dt_mod;
02066   t1 = t + 0.5 * ctl->dt_mod;
02067
02068   /* Loop over air parcels... */
02069   for (ip = 0; ip < atm->np; ip++) {
02070
02071     /* Check time... */
02072     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02073       continue;
02074
02075     /* Check station flag... */
02076     if (ctl->qnt_stat >= 0)
02077       if (atm->q[ctl->qnt_stat][ip])
02078         continue;
02079
02080     /* Get Cartesian coordinates... */
02081     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02082
02083     /* Check horizontal distance... */
02084     if (DIST2(x0, x1) > rmax2)
02085       continue;
02086
02087     /* Set station flag... */
02088     if (ctl->qnt_stat >= 0)
02089       atm->q[ctl->qnt_stat][ip] = 1;
02090
02091     /* Write data... */
02092     fprintf(out, "%.2f %g %g %g",
02093             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02094     for (iq = 0; iq < ctl->nq; iq++) {
02095       fprintf(out, " ");
02096       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02097     }
02098     fprintf(out, "\n");
02099   }
02100
02101   /* Close file... */
02102   if (t == ctl->t_stop)
02103     fclose(out);
02104 }
```

5.15 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.

5.15.1 Detailed Description

MPTRAC library declarations.

Definition in file libtrac.h.

5.15.2 Function Documentation

5.15.2.1 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 }
```

5.15.2.2 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 }
```

5.15.2.3 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 }
```

5.15.2.4 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 }
```

5.15.2.5 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 }
```

5.15.2.6 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 }
```

5.15.2.7 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 }
```

5.15.2.8 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 }
```

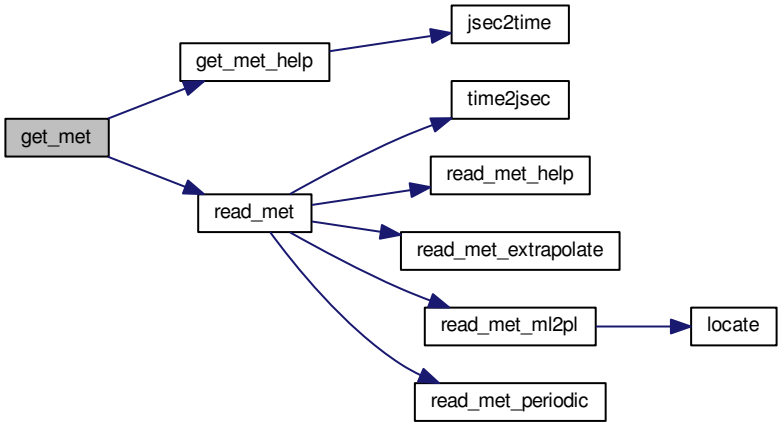
5.15.2.9 void get_met (ctl_t * *ctl*, char * *metbase*, double *t*, met_t * *met0*, met_t * *met1*)

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:



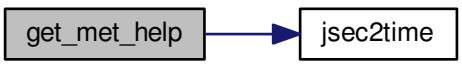
5.15.2.10 void get_met_help (double *t*, int *direct*, char * *metbase*, double *dt_met*, char * *filename*)

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:



5.15.2.11 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 }
```

5.15.2.12 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 }
```

5.15.2.13 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*)

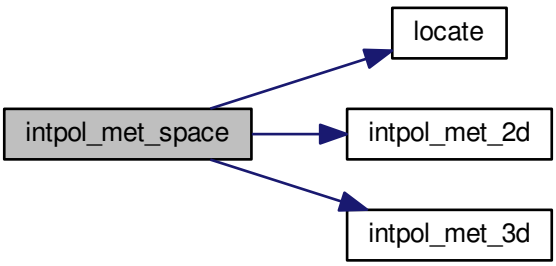
Spatial interpolation of meteorological data.

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:



5.15.2.14 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*)

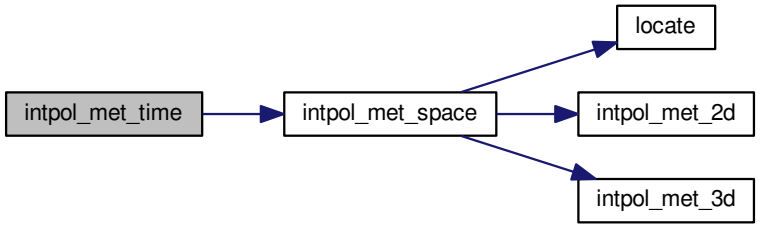
Temporal interpolation of meteorological data.

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:



5.15.2.15 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 }
```

5.15.2.16 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 }
```

5.15.2.17 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 binary data... */
00430   if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
00431     FREAD(&atm->np, int,
00432           1,
00433           in);
00434     FREAD(atm->time, double,
00435           (size_t) atm->np,
00436           in);
00437     FREAD(atm->p, double,
00438           (size_t) atm->np,
00439           in);
00440     FREAD(atm->lon, double,
00441           (size_t) atm->np,
00442           in);
00443     FREAD(atm->lat, double,
00444           (size_t) atm->np,
00445           in);
00446     for (iq = 0; iq < ctl->nq; iq++)
00447       FREAD(atm->q[iq], double,
```

```
00448               (size_t) atm->np,
00449             in);
00450   }
00451
00452   /* Read ASCII data... */
00453   else {
00454
00455     /* Read line... */
00456     while (fgets(line, LEN, in)) {
00457
00458       /* Read data... */
00459       TOK(line, tok, "%lg", atm->time[atm->np]);
00460       TOK(NULL, tok, "%lg", atm->p[atm->np]);
00461       TOK(NULL, tok, "%lg", atm->lon[atm->np]);
00462       TOK(NULL, tok, "%lg", atm->lat[atm->np]);
00463       for (iq = 0; iq < ctl->nq; iq++)
00464         TOK(NULL, tok, "%lg", atm->q[iq][atm->np]);
00465
00466       /* Convert altitude to pressure... */
00467       atm->p[atm->np] = P(atm->p[atm->np]);
00468
00469       /* Increment data point counter... */
00470       if ((++atm->np) > NP)
00471         ERRMSG("Too many data points!");
00472     }
00473   }
00474
00475   /* Close file... */
00476   fclose(in);
00477
00478   /* Check number of points... */
00479   if (atm->np < 1)
00480     ERRMSG("Can not read any data!");
00481 }
```

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

Read control parameters.

Definition at line 485 of file libtrac.c.

```
00489               {
00490
00491   int ip, iq;
00492
00493   /* Write info... */
00494   printf("\nMassive-Parallel Trajectory Calculations (MPTRAC)\n"
00495          "(executable: %s | compiled: %s, %s)\n\n",
00496          argv[0], __DATE__, __TIME__);
00497
00498   /* Initialize quantity indices... */
00499   ctl->qnt_ens = -1;
00500   ctl->qnt_m = -1;
00501   ctl->qnt_r = -1;
00502   ctl->qnt_rho = -1;
00503   ctl->qnt_ps = -1;
00504   ctl->qnt_p = -1;
00505   ctl->qnt_t = -1;
00506   ctl->qnt_u = -1;
00507   ctl->qnt_v = -1;
00508   ctl->qnt_w = -1;
00509   ctl->qnt_h2o = -1;
00510   ctl->qnt_o3 = -1;
00511   ctl->qnt_theta = -1;
00512   ctl->qnt_pv = -1;
00513   ctl->qnt_tice = -1;
00514   ctl->qnt_tsts = -1;
00515   ctl->qnt_tnat = -1;
00516   ctl->qnt_gw_var = -1;
00517   ctl->qnt_stat = -1;
00518
00519   /* Read quantities... */
00520   ctl->nq = (int) scan_ctl(filename, argc, argv, "NQ", -1, "0", NULL);
00521   if (ctl->nq > NQ)
00522     ERRMSG("Too many quantities!");
00523   for (iq = 0; iq < ctl->nq; iq++) {
00524
00525     /* Read quantity name and format... */
00526     scan_ctl(filename, argc, argv, "QNT_NAME", iq, "", ctl->qnt_name[iq]);
00527     scan_ctl(filename, argc, argv, "QNT_FORMAT", iq, "%g",
```

```
00528             ctl->qnt_format[iq]);
00529
00530     /* Try to identify quantity... */
00531     if (strcmp(ctl->qnt_name[iq], "ens") == 0) {
00532       ctl->qnt_ens = iq;
00533       sprintf(ctl->qnt_unit[iq], "-");
00534     } else if (strcmp(ctl->qnt_name[iq], "m") == 0) {
00535       ctl->qnt_m = iq;
00536       sprintf(ctl->qnt_unit[iq], "kg");
00537     } else if (strcmp(ctl->qnt_name[iq], "r") == 0) {
00538       ctl->qnt_r = iq;
00539       sprintf(ctl->qnt_unit[iq], "m");
00540     } else if (strcmp(ctl->qnt_name[iq], "rho") == 0) {
00541       ctl->qnt_rho = iq;
00542       sprintf(ctl->qnt_unit[iq], "kg/m^3");
00543     } else if (strcmp(ctl->qnt_name[iq], "ps") == 0) {
00544       ctl->qnt_ps = iq;
00545       sprintf(ctl->qnt_unit[iq], "hPa");
00546     } else if (strcmp(ctl->qnt_name[iq], "p") == 0) {
00547       ctl->qnt_p = iq;
00548       sprintf(ctl->qnt_unit[iq], "hPa");
00549     } else if (strcmp(ctl->qnt_name[iq], "t") == 0) {
00550       ctl->qnt_t = iq;
00551       sprintf(ctl->qnt_unit[iq], "K");
00552     } else if (strcmp(ctl->qnt_name[iq], "u") == 0) {
00553       ctl->qnt_u = iq;
00554       sprintf(ctl->qnt_unit[iq], "m/s");
00555     } else if (strcmp(ctl->qnt_name[iq], "v") == 0) {
00556       ctl->qnt_v = iq;
00557       sprintf(ctl->qnt_unit[iq], "m/s");
00558     } else if (strcmp(ctl->qnt_name[iq], "w") == 0) {
00559       ctl->qnt_w = iq;
00560       sprintf(ctl->qnt_unit[iq], "hPa/s");
00561     } else if (strcmp(ctl->qnt_name[iq], "h2o") == 0) {
00562       ctl->qnt_h2o = iq;
00563       sprintf(ctl->qnt_unit[iq], "1");
00564     } else if (strcmp(ctl->qnt_name[iq], "o3") == 0) {
00565       ctl->qnt_o3 = iq;
00566       sprintf(ctl->qnt_unit[iq], "1");
00567     } else if (strcmp(ctl->qnt_name[iq], "theta") == 0) {
00568       ctl->qnt_theta = iq;
00569       sprintf(ctl->qnt_unit[iq], "K");
00570     } else if (strcmp(ctl->qnt_name[iq], "pv") == 0) {
00571       ctl->qnt_pv = iq;
00572       sprintf(ctl->qnt_unit[iq], "PVU");
00573     } else if (strcmp(ctl->qnt_name[iq], "tice") == 0) {
00574       ctl->qnt_tice = iq;
00575       sprintf(ctl->qnt_unit[iq], "K");
00576     } else if (strcmp(ctl->qnt_name[iq], "tsts") == 0) {
00577       ctl->qnt_tsts = iq;
00578       sprintf(ctl->qnt_unit[iq], "K");
00579     } else if (strcmp(ctl->qnt_name[iq], "tnat") == 0) {
00580       ctl->qnt_tnat = iq;
00581       sprintf(ctl->qnt_unit[iq], "K");
00582     } else if (strcmp(ctl->qnt_name[iq], "gw_var") == 0) {
00583       ctl->qnt_gw_var = iq;
00584       sprintf(ctl->qnt_unit[iq], "K^2");
00585     } else if (strcmp(ctl->qnt_name[iq], "stat") == 0) {
00586       ctl->qnt_stat = iq;
00587       sprintf(ctl->qnt_unit[iq], "-");
00588     } else
00589       scan_ctl(filename, argc, argv, "QNT_UNIT", iq, "", ctl->qnt_unit[iq]);
00590   }
00591
00592   /* Time steps of simulation... */
00593   ctl->direction =
00594     (int) scan_ctl(filename, argc, argv, "DIRECTION", -1, "1", NULL);
00595   if (ctl->direction != -1 && ctl->direction != 1)
00596     ERRMSG("Set DIRECTION to -1 or 1!");
00597   ctl->t_start =
00598     scan_ctl(filename, argc, argv, "T_START", -1, "-1e100", NULL);
00599   ctl->t_stop = scan_ctl(filename, argc, argv, "T_STOP", -1, "-1e100", NULL);
00600   ctl->dt_mod = scan_ctl(filename, argc, argv, "DT_MOD", -1, "600", NULL);
00601
00602   /* Meteorological data... */
00603   ctl->dt_met = scan_ctl(filename, argc, argv, "DT_MET", -1, "21600", NULL);
00604   ctl->met_np = (int) scan_ctl(filename, argc, argv, "MET_NP", -1, "0", NULL);
00605   if (ctl->met_np > EP)
00606     ERRMSG("Too many levels!");
00607   for (ip = 0; ip < ctl->met_np; ip++)
00608     ctl->met_p[ip] = scan_ctl(filename, argc, argv, "MET_P", ip, "", NULL);
00609   scan_ctl(filename, argc, argv, "MET_STAGE", -1, "-", ctl->met_stage);
00610
00611   /* Isosurface parameters... */
00612   ctl->isosurf
00613     = (int) scan_ctl(filename, argc, argv, "ISOSURF", -1, "0", NULL);
00614   scan_ctl(filename, argc, argv, "BALLOON", -1, "-", ctl->balloon);
```

```
00615
00616   /* Diffusion parameters... */
00617   ctl->turb_dx_trop
00618     = scan_ctl(filename, argc, argv, "TURB_DX_TROP", -1, "50.0", NULL);
00619   ctl->turb_dx_strat
00620     = scan_ctl(filename, argc, argv, "TURB_DX_STRAT", -1, "0.0", NULL);
00621   ctl->turb_dz_trop
00622     = scan_ctl(filename, argc, argv, "TURB_DZ_TROP", -1, "0.0", NULL);
00623   ctl->turb_dz_strat
00624     = scan_ctl(filename, argc, argv, "TURB_DZ_STRAT", -1, "0.1", NULL);
00625   ctl->turb_meso =
00626     scan_ctl(filename, argc, argv, "TURB_MESO", -1, "0.16", NULL);
00627
00628   /* Life time of particles... */
00629   ctl->tdec_trop = scan_ctl(filename, argc, argv, "TDEC_TROP", -1, "0", NULL);
00630   ctl->tdec_strat =
00631     scan_ctl(filename, argc, argv, "TDEC_STRAT", -1, "0", NULL);
00632
00633   /* PSC analysis... */
00634   ctl->psc_h2o = scan_ctl(filename, argc, argv, "PSC_H2O", -1, "4e-6", NULL);
00635   ctl->psc_hno3 =
00636     scan_ctl(filename, argc, argv, "PSC_HNO3", -1, "9e-9", NULL);
00637
00638   /* Gravity wave analysis... */
00639   scan_ctl(filename, argc, argv, "GW_BASENAME", -1, "-", ctl->
      gw_basename);
00640
00641   /* Output of atmospheric data... */
00642   scan_ctl(filename, argc, argv, "ATM_BASENAME", -1, "-", ctl->
      atm_basename);
00643   scan_ctl(filename, argc, argv, "ATM_GPFILE", -1, "-", ctl->atm_gpfile);
00644   ctl->atm_dt_out =
00645     scan_ctl(filename, argc, argv, "ATM_DT_OUT", -1, "86400", NULL);
00646   ctl->atm_filter =
00647     (int) scan_ctl(filename, argc, argv, "ATM_FILTER", -1, "0", NULL);
00648   ctl->atm_bin =
00649     (int) scan_ctl(filename, argc, argv, "ATM_BIN", -1, "0", NULL);
00650
00651   /* Output of CSI data... */
00652   scan_ctl(filename, argc, argv, "CSI_BASENAME", -1, "-", ctl->
      csi_basename);
00653   ctl->csi_dt_out =
00654     scan_ctl(filename, argc, argv, "CSI_DT_OUT", -1, "86400", NULL);
00655   scan_ctl(filename, argc, argv, "CSI_OBSFILE", -1, "obs.tab",
00656            ctl->csi_obsfile);
00657   ctl->csi_obsmin =
00658     scan_ctl(filename, argc, argv, "CSI_OBSMIN", -1, "0", NULL);
00659   ctl->csi_modmin =
00660     scan_ctl(filename, argc, argv, "CSI_MODMIN", -1, "0", NULL);
00661   ctl->csi_z0 = scan_ctl(filename, argc, argv, "CSI_Z0", -1, "0", NULL);
00662   ctl->csi_z1 = scan_ctl(filename, argc, argv, "CSI_Z1", -1, "100", NULL);
00663   ctl->csi_nz = (int) scan_ctl(filename, argc, argv, "CSI_NZ", -1, "1", NULL);
00664   ctl->csi_lon0 =
00665     scan_ctl(filename, argc, argv, "CSI_LON0", -1, "-180", NULL);
00666   ctl->csi_lon1 = scan_ctl(filename, argc, argv, "CSI_LON1", -1, "180", NULL);
00667   ctl->csi_nx =
00668     (int) scan_ctl(filename, argc, argv, "CSI_NX", -1, "360", NULL);
00669   ctl->csi_lat0 = scan_ctl(filename, argc, argv, "CSI_LAT0", -1, "-90", NULL);
00670   ctl->csi_lat1 = scan_ctl(filename, argc, argv, "CSI_LAT1", -1, "90", NULL);
00671   ctl->csi_ny =
00672     (int) scan_ctl(filename, argc, argv, "CSI_NY", -1, "180", NULL);
00673
00674   /* Output of ensemble data... */
00675   scan_ctl(filename, argc, argv, "ENS_BASENAME", -1, "-", ctl->
      ens_basename);
00676
00677   /* Output of grid data... */
00678   scan_ctl(filename, argc, argv, "GRID_BASENAME", -1, "-",
00679            ctl->grid_basename);
00680   scan_ctl(filename, argc, argv, "GRID_GPFILE", -1, "-", ctl->
      grid_gpfile);
00681   ctl->grid_dt_out =
00682     scan_ctl(filename, argc, argv, "GRID_DT_OUT", -1, "86400", NULL);
00683   ctl->grid_sparse =
00684     (int) scan_ctl(filename, argc, argv, "GRID_SPARSE", -1, "0", NULL);
00685   ctl->grid_z0 = scan_ctl(filename, argc, argv, "GRID_Z0", -1, "0", NULL);
00686   ctl->grid_z1 = scan_ctl(filename, argc, argv, "GRID_Z1", -1, "100", NULL);
00687   ctl->grid_nz =
00688     (int) scan_ctl(filename, argc, argv, "GRID_NZ", -1, "1", NULL);
00689   ctl->grid_lon0 =
00690     scan_ctl(filename, argc, argv, "GRID_LON0", -1, "-180", NULL);
00691   ctl->grid_lon1 =
00692     scan_ctl(filename, argc, argv, "GRID_LON1", -1, "180", NULL);
00693   ctl->grid_nx =
00694     (int) scan_ctl(filename, argc, argv, "GRID_NX", -1, "360", NULL);
00695   ctl->grid_lat0 =
00696     scan_ctl(filename, argc, argv, "GRID_LAT0", -1, "-90", NULL);
```

```
00697   ctl->grid_lat1 =
00698     scan_ctl(filename, argc, argv, "GRID_LAT1", -1, "90", NULL);
00699   ctl->grid_ny =
00700     (int) scan_ctl(filename, argc, argv, "GRID_NY", -1, "180", NULL);
00701
00702   /* Output of profile data... */
00703   scan_ctl(filename, argc, argv, "PROF_BASENAME", -1, "-",
00704            ctl->prof_basename);
00705   scan_ctl(filename, argc, argv, "PROF_OBSFILE", -1, "-", ctl->
      prof_obsfile);
00706   ctl->prof_z0 = scan_ctl(filename, argc, argv, "PROF_Z0", -1, "0", NULL);
00707   ctl->prof_z1 = scan_ctl(filename, argc, argv, "PROF_Z1", -1, "60", NULL);
00708   ctl->prof_nz =
00709     (int) scan_ctl(filename, argc, argv, "PROF_NZ", -1, "60", NULL);
00710   ctl->prof_lon0 =
00711     scan_ctl(filename, argc, argv, "PROF_LON0", -1, "-180", NULL);
00712   ctl->prof_lon1 =
00713     scan_ctl(filename, argc, argv, "PROF_LON1", -1, "180", NULL);
00714   ctl->prof_nx =
00715     (int) scan_ctl(filename, argc, argv, "PROF_NX", -1, "360", NULL);
00716   ctl->prof_lat0 =
00717     scan_ctl(filename, argc, argv, "PROF_LAT0", -1, "-90", NULL);
00718   ctl->prof_lat1 =
00719     scan_ctl(filename, argc, argv, "PROF_LAT1", -1, "90", NULL);
00720   ctl->prof_ny =
00721     (int) scan_ctl(filename, argc, argv, "PROF_NY", -1, "180", NULL);
00722
00723   /* Output of station data... */
00724   scan_ctl(filename, argc, argv, "STAT_BASENAME", -1, "-",
00725            ctl->stat_basename);
00726   ctl->stat_lon = scan_ctl(filename, argc, argv, "STAT_LON", -1, "0", NULL);
00727   ctl->stat_lat = scan_ctl(filename, argc, argv, "STAT_LAT", -1, "0", NULL);
00728   ctl->stat_r = scan_ctl(filename, argc, argv, "STAT_R", -1, "50", NULL);
00729 }
```

Here is the call graph for this function:

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

Read meteorological data file.

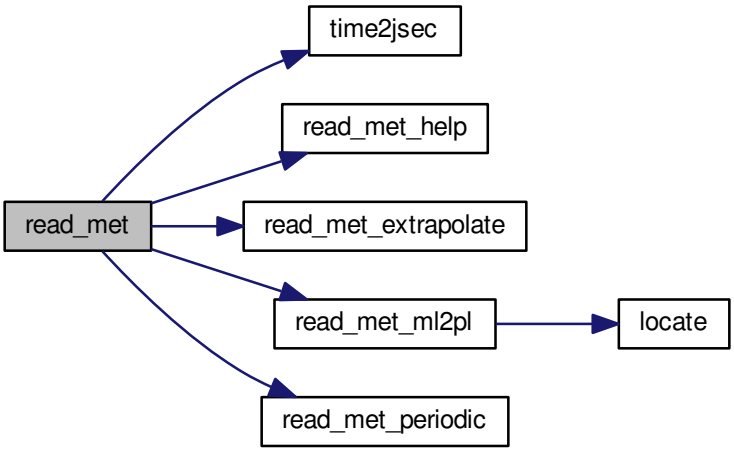
Definition at line 733 of file libtrac.c.

```
00736               {
00737
00738   char cmd[LEN], levname[LEN], tstr[10];
00739
00740   static float help[EX * EY];
00741
00742   int ix, iy, ip, dimid, ncid, varid, year, mon, day, hour;
00743
00744   size_t np, nx, ny;
00745
00746   /* Write info... */
00747   printf("Read meteorological data: %s\n", filename);
00748
00749   /* Get time from filename... */
00750   sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
00751   year = atoi(tstr);
00752   sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
00753   mon = atoi(tstr);
00754   sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
```

```
00755   day = atoi(tstr);
00756   sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
00757   hour = atoi(tstr);
00758   time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);
00759
00760   /* Open netCDF file... */
00761   if (nc_open(filename, NC_NOWRITE, &ncid) != NC_NOERR) {
00762
00763     /* Try to stage meteo file... */
00764     STOP_TIMER(TIMER_INPUT);
00765     START_TIMER(TIMER_STAGE);
00766     if (ctl->met_stage[0] != '-') {
00767       sprintf(cmd, "%s %d %02d %02d %02d %s", ctl->met_stage,
00768               year, mon, day, hour, filename);
00769       if (system(cmd) != 0)
00770         ERRMSG("Error while staging meteo data!");
00771     }
00772     STOP_TIMER(TIMER_STAGE);
00773     START_TIMER(TIMER_INPUT);
00774
00775     /* Try to open again... */
00776     NC(nc_open(filename, NC_NOWRITE, &ncid));
00777   }
00778
00779   /* Get dimensions... */
00780   NC(nc_inq_dimid(ncid, "lon", &dimid));
00781   NC(nc_inq_dimlen(ncid, dimid, &nx));
00782   if (nx < 2 || nx > EX)
00783     ERRMSG("Number of longitudes out of range!");
00784
00785   NC(nc_inq_dimid(ncid, "lat", &dimid));
00786   NC(nc_inq_dimlen(ncid, dimid, &ny));
00787   if (ny < 2 || ny > EY)
00788     ERRMSG("Number of latitudes out of range!");
00789
00790   sprintf(levname, "lev");
00791   NC(nc_inq_dimid(ncid, levname, &dimid));
00792   NC(nc_inq_dimlen(ncid, dimid, &np));
00793   if (np == 1) {
00794     sprintf(levname, "lev_2");
00795     NC(nc_inq_dimid(ncid, levname, &dimid));
00796     NC(nc_inq_dimlen(ncid, dimid, &np));
00797   }
00798   if (np < 2 || np > EP)
00799     ERRMSG("Number of levels out of range!");
00800
00801   /* Store dimensions... */
00802   met->np = (int) np;
00803   met->nx = (int) nx;
00804   met->ny = (int) ny;
00805
00806   /* Get horizontal grid... */
00807   NC(nc_inq_varid(ncid, "lon", &varid));
00808   NC(nc_get_var_double(ncid, varid, met->lon));
00809   NC(nc_inq_varid(ncid, "lat", &varid));
00810   NC(nc_get_var_double(ncid, varid, met->lat));
00811
00812   /* Read meteorological data... */
00813   read_met_help(ncid, "t", "T", met, met->t, 1.0);
00814   read_met_help(ncid, "u", "U", met, met->u, 1.0);
00815   read_met_help(ncid, "v", "V", met, met->v, 1.0);
00816   read_met_help(ncid, "w", "W", met, met->w, 0.01f);
00817   read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
00818   read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);
00819
00820   /* Meteo data on pressure levels... */
00821   if (ctl->met_np <= 0) {
00822
00823     /* Read pressure levels from file... */
00824     NC(nc_inq_varid(ncid, levname, &varid));
00825     NC(nc_get_var_double(ncid, varid, met->p));
00826     for (ip = 0; ip < met->np; ip++)
00827       met->p[ip] /= 100.;
00828
00829     /* Extrapolate data for lower boundary... */
00830     read_met_extrapolate(met);
00831   }
00832
00833   /* Meteo data on model levels... */
00834   else {
00835
00836     /* Read pressure data from file... */
00837     read_met_help(ncid, "pl", "PL", met, met->pl, 0.01f);
00838
00839     /* Interpolate from model levels to pressure levels... */
00840     read_met_ml2pl(ctl, met, met->t);
00841     read_met_ml2pl(ctl, met, met->u);
```

```
00842     read_met_ml2pl(ctl, met, met->v);
00843     read_met_ml2pl(ctl, met, met->w);
00844     read_met_ml2pl(ctl, met, met->h2o);
00845     read_met_ml2pl(ctl, met, met->o3);
00846
00847     /* Set pressure levels... */
00848     met->np = ctl->met_np;
00849     for (ip = 0; ip < met->np; ip++)
00850       met->p[ip] = ctl->met_p[ip];
00851   }
00852
00853   /* Check ordering of pressure levels... */
00854   for (ip = 1; ip < met->np; ip++)
00855     if (met->p[ip - 1] < met->p[ip])
00856       ERRMSG("Pressure levels must be descending!");
00857
00858   /* Read surface pressure... */
00859   if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
00860       || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
00861     NC(nc_get_var_float(ncid, varid, help));
00862     for (iy = 0; iy < met->ny; iy++)
00863       for (ix = 0; ix < met->nx; ix++)
00864         met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
00865   } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
00866              || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
00867     NC(nc_get_var_float(ncid, varid, help));
00868     for (iy = 0; iy < met->ny; iy++)
00869       for (ix = 0; ix < met->nx; ix++)
00870         met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
00871   } else
00872     for (ix = 0; ix < met->nx; ix++)
00873       for (iy = 0; iy < met->ny; iy++)
00874         met->ps[ix][iy] = met->p[0];
00875
00876   /* Create periodic boundary conditions... */
00877   read_met_periodic(met);
00878
00879   /* Close file... */
00880   NC(nc_close(ncid));
00881 }
```

Here is the call graph for this function:



5.15.2.20 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 885 of file libtrac.c.

```
00886               {
00887
00888   int ip, ip0, ix, iy;
00889
00890   /* Loop over columns... */
00891   for (ix = 0; ix < met->nx; ix++)
00892     for (iy = 0; iy < met->ny; iy++) {
00893
00894       /* Find lowest valid data point... */
00895       for (ip0 = met->np - 1; ip0 >= 0; ip0--)
00896         if (!gsl_finite(met->t[ix][iy][ip0])
00897             || !gsl_finite(met->u[ix][iy][ip0])
00898             || !gsl_finite(met->v[ix][iy][ip0])
00899             || !gsl_finite(met->w[ix][iy][ip0]))
00900           break;
00901
00902       /* Extrapolate... */
00903       for (ip = ip0; ip >= 0; ip--) {
00904         met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
00905         met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
00906         met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
00907         met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
00908         met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
00909         met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
00910       }
00911     }
00912 }
```

5.15.2.21 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 916 of file libtrac.c.

```
00922             {
00923
00924   static float help[EX * EY * EP];
00925
00926   int ip, ix, iy, n = 0, varid;
00927
00928   /* Check if variable exists... */
00929   if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
00930     if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
00931       return;
00932
00933   /* Read data... */
00934   NC(nc_get_var_float(ncid, varid, help));
00935
00936   /* Copy and check data... */
00937   for (ip = 0; ip < met->np; ip++)
00938     for (iy = 0; iy < met->ny; iy++)
00939       for (ix = 0; ix < met->nx; ix++) {
00940         dest[ix][iy][ip] = scl * help[n++];
00941         if (fabs(dest[ix][iy][ip] / scl) > 1e14)
00942           dest[ix][iy][ip] = GSL_NAN;
00943       }
00944 }
```

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

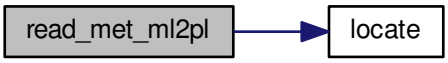
Convert meteorological data from model levels to pressure levels.

Definition at line 948 of file libtrac.c.

```
00951                         {
00952
00953   double aux[EP], p[EP], pt;
00954
00955   int ip, ip2, ix, iy;
00956
00957   /* Loop over columns... */
00958   for (ix = 0; ix < met->nx; ix++)
00959     for (iy = 0; iy < met->ny; iy++) {
00960
```

```
00961       /* Copy pressure profile... */
00962       for (ip = 0; ip < met->np; ip++)
00963         p[ip] = met->pl[ix][iy][ip];
00964
00965       /* Interpolate... */
00966       for (ip = 0; ip < ctl->met_np; ip++) {
00967         pt = ctl->met_p[ip];
00968         if ((pt > p[0] && p[0] > p[1]) || (pt < p[0] && p[0] < p[1]))
00969           pt = p[0];
00970         else if ((pt > p[met->np - 1] && p[1] > p[0])
00971                  || (pt < p[met->np - 1] && p[1] < p[0]))
00972           pt = p[met->np - 1];
00973         ip2 = locate(p, met->np, pt);
00974         aux[ip] = LIN(p[ip2], var[ix][iy][ip2],
00975                       p[ip2 + 1], var[ix][iy][ip2 + 1], pt);
00976       }
00977
00978       /* Copy data... */
00979       for (ip = 0; ip < ctl->met_np; ip++)
00980         var[ix][iy][ip] = (float) aux[ip];
00981     }
00982 }
```

Here is the call graph for this function:



5.15.2.23 void read_met_periodic (met_t * *met*)

Create meteorological data with periodic boundary conditions.

Definition at line 986 of file libtrac.c.

```
00987                {
00988
00989   int ip, iy;
00990
00991   /* Check longitudes... */
00992   if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
00993              + met->lon[1] - met->lon[0] - 360) < 0.01))
00994     return;
00995
00996   /* Increase longitude counter... */
00997   if ((++met->nx) > EX)
00998     ERRMSG("Cannot create periodic boundary conditions!");
00999
01000   /* Set longitude... */
01001   met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->
      lon[0];
01002
01003   /* Loop over latitudes and pressure levels... */
01004   for (iy = 0; iy < met->ny; iy++)
01005     for (ip = 0; ip < met->np; ip++) {
01006       met->ps[met->nx - 1][iy] = met->ps[0][iy];
01007       met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
01008       met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
01009       met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
01010       met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
01011       met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
01012       met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
01013     }
01014 }
```

5.15.2.24 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 1018 of file libtrac.c.

```
01025               {
01026
01027   FILE *in = NULL;
01028
01029   char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
01030     msg[LEN], rvarname[LEN], rval[LEN];
01031
01032   int contain = 0, i;
01033
01034   /* Open file... */
01035   if (filename[strlen(filename) - 1] != '-')
01036     if (!(in = fopen(filename, "r")))
01037       ERRMSG("Cannot open file!");
01038
01039   /* Set full variable name... */
01040   if (arridx >= 0) {
01041     sprintf(fullname1, "%s[%d]", varname, arridx);
01042     sprintf(fullname2, "%s[*]", varname);
01043   } else {
01044     sprintf(fullname1, "%s", varname);
01045     sprintf(fullname2, "%s", varname);
01046   }
01047
01048   /* Read data... */
01049   if (in != NULL)
01050     while (fgets(line, LEN, in))
01051       if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
01052         if (strcasecmp(rvarname, fullname1) == 0 ||
01053             strcasecmp(rvarname, fullname2) == 0) {
01054           contain = 1;
01055           break;
01056         }
01057   for (i = 1; i < argc - 1; i++)
01058     if (strcasecmp(argv[i], fullname1) == 0 ||
01059         strcasecmp(argv[i], fullname2) == 0) {
01060       sprintf(rval, "%s", argv[i + 1]);
01061       contain = 1;
01062       break;
01063     }
01064
01065   /* Close file... */
01066   if (in != NULL)
01067     fclose(in);
01068
01069   /* Check for missing variables... */
01070   if (!contain) {
01071     if (strlen(defvalue) > 0)
01072       sprintf(rval, "%s", defvalue);
01073     else {
01074       sprintf(msg, "Missing variable %s!\n", fullname1);
01075       ERRMSG(msg);
01076     }
01077   }
01078
01079   /* Write info... */
01080   printf("%s = %s\n", fullname1, rval);
01081
01082   /* Return values... */
01083   if (value != NULL)
01084     sprintf(value, "%s", rval);
01085   return atof(rval);
01086 }
```

5.15.2.25 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 1090 of file libtrac.c.

```
01098                {
01099
01100   struct tm t0, t1;
01101
01102   t0.tm_year = 100;
01103   t0.tm_mon = 0;
01104   t0.tm_mday = 1;
01105   t0.tm_hour = 0;
01106   t0.tm_min = 0;
01107   t0.tm_sec = 0;
01108
01109   t1.tm_year = year - 1900;
01110   t1.tm_mon = mon - 1;
01111   t1.tm_mday = day;
01112   t1.tm_hour = hour;
01113   t1.tm_min = min;
01114   t1.tm_sec = sec;
01115
01116   *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
01117 }
```

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

Measure wall-clock time.

Definition at line 1121 of file libtrac.c.

```
01124            {
01125
01126   static double starttime[NTIMER], runtime[NTIMER];
01127
01128   /* Check id... */
01129   if (id < 0 || id >= NTIMER)
01130     ERRMSG("Too many timers!");
01131
01132   /* Start timer... */
01133   if (mode == 1) {
01134     if (starttime[id] <= 0)
01135       starttime[id] = omp_get_wtime();
01136     else
01137       ERRMSG("Timer already started!");
01138   }
01139
01140   /* Stop timer... */
01141   else if (mode == 2) {
01142     if (starttime[id] > 0) {
01143       runtime[id] = runtime[id] + omp_get_wtime() - starttime[id];
01144       starttime[id] = -1;
01145     }
01146   }
01147
01148   /* Print timer... */
01149   else if (mode == 3)
01150     printf("%s = %.3f s\n", name, runtime[id]);
01151 }
```

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

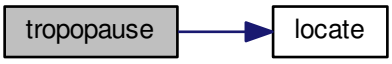
Definition at line 1155 of file libtrac.c.

```
01157              {
01158
01159   static double doys[12]
01160   = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
01161
01162   static double lats[73]
01163     = { -90, -87.5, -85, -82.5, -80, -77.5, -75, -72.5, -70, -67.5,
01164     -65, -62.5, -60, -57.5, -55, -52.5, -50, -47.5, -45, -42.5,
01165     -40, -37.5, -35, -32.5, -30, -27.5, -25, -22.5, -20, -17.5,
01166     -15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5,
01167     15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40, 42.5,
01168     45, 47.5, 50, 52.5, 55, 57.5, 60, 62.5, 65, 67.5, 70, 72.5,
01169     75, 77.5, 80, 82.5, 85, 87.5, 90
01170   };
01171
```

```
01172   static double tps[12][73]
01173     = { {324.1, 325.6, 325, 324.3, 322.5, 319.7, 314, 307.2, 301.8, 299.6,
01174          297.1, 292.2, 285.6, 276.1, 264, 248.9, 231.9, 213.5, 194.4,
01175          175.3, 157, 140.4, 126.7, 116.3, 109.5, 105.4, 103, 101.4, 100.4,
01176          99.69, 99.19, 98.84, 98.56, 98.39, 98.39, 98.42, 98.44, 98.54,
01177          98.68, 98.81, 98.89, 98.96, 99.12, 99.65, 101.4, 105.4, 113.5, 128,
01178          152.1, 184.7, 214, 234.1, 247.3, 255.8, 262.6, 267.7, 271.7, 275,
01179          277.2, 279, 280.1, 280.4, 280.6, 280.1, 279.3, 278.3, 276.8, 275.8,
01180          275.3, 275.6, 275.4, 274.1, 273.5},
01181    {337.3, 338.7, 337.8, 336.4, 333, 328.8, 321.1, 312.6, 306.6, 303.7,
01182     300.2, 293.8, 285.4, 273.8, 259.6, 242.7, 224.4, 205.2, 186, 167.5,
01183     150.3, 135, 122.8, 113.9, 108.2, 104.7, 102.5, 101.1, 100.2, 99.42,
01184     98.88, 98.52, 98.25, 98.09, 98.07, 98.1, 98.12, 98.2, 98.25, 98.27,
01185     98.26, 98.27, 98.36, 98.79, 100.2, 104.2, 113.7, 131.2, 159.5, 193,
01186     220.4, 238.1, 250.2, 258.1, 264.7, 269.7, 273.7, 277.3, 280.2, 282.8,
01187     284.9, 286.5, 288.1, 288.8, 289, 288.5, 287.2, 286.3, 286.1, 287.2,
01188     287.5, 286.2, 285.8},
01189    {335, 336, 335.7, 335.1, 332.3, 328.1, 320.6, 311.8, 305.1, 301.9,
01190     297.6, 290, 280.4, 268.3, 254.6, 239.6, 223.9, 207.9, 192.2, 176.9,
01191     161.7, 146.4, 132.2, 120.6, 112.3, 107.2, 104.3, 102.4, 101.3,
01192     100.4, 99.86, 99.47, 99.16, 98.97, 98.94, 98.97, 99, 99.09, 99.2,
01193     99.31, 99.35, 99.41, 99.51, 99.86, 101.1, 104.9, 114.3, 131, 156.8,
01194     186.3, 209.3, 224.6, 236.8, 246.3, 254.9, 262.3, 268.8, 274.8,
01195     279.9, 284.6, 288.6, 291.6, 294.9, 297.5, 299.8, 301.8, 303.1,
01196     304.3, 304.9, 306, 306.6, 306.2, 306},
01197    {306.2, 306.7, 305.7, 307.1, 307.3, 306.4, 301.8, 296.2, 292.4,
01198     290.3, 287.1, 280.9, 273.4, 264.3, 254.1, 242.8, 231, 219, 207.2,
01199     195.5, 183.3, 169.7, 154.7, 138.7, 124.1, 113.6, 107.8, 104.7,
01200     102.8, 101.7, 100.9, 100.4, 100, 99.79, 99.7, 99.66, 99.68, 99.79,
01201     99.94, 100.2, 100.5, 100.9, 101.4, 102.1, 103.4, 107, 115.2, 129.1,
01202     148.7, 171, 190.8, 205.6, 218.4, 229.4, 239.6, 248.6, 256.5,
01203     263.7, 270.3, 276.6, 282.6, 288.1, 294.5, 300.4, 306.3, 311.4,
01204     315.1, 318.3, 320.3, 322.2, 322.8, 321.5, 321.1},
01205    {266.5, 264.9, 260.8, 261, 262, 263, 261.3, 259.7, 259.2, 259.8,
01206     260.1, 258.6, 256.7, 253.6, 249.5, 243.9, 237.4, 230, 222.1, 213.9,
01207     205, 194.4, 180.4, 161.8, 140.7, 122.9, 112.1, 106.7, 104.1, 102.7,
01208     101.8, 101.4, 101.1, 101, 101, 101, 101.1, 101.2, 101.5, 101.9,
01209     102.4, 103, 103.8, 104.9, 106.8, 110.1, 115.6, 124, 135.2, 148.9,
01210     165.2, 181.3, 198, 211.8, 223.5, 233.8, 242.9, 251.5, 259, 266.2,
01211     273.1, 279.2, 286.2, 292.8, 299.6, 306, 311.1, 315.5, 318.8, 322.6,
01212     325.3, 325.8, 325.8},
01213    {220.1, 218.1, 210.8, 207.2, 207.6, 210.5, 211.4, 213.5, 217.3,
01214     222.4, 227.9, 232.8, 237.4, 240.8, 242.8, 243, 241.5, 238.6, 234.2,
01215     228.5, 221, 210.7, 195.1, 172.9, 147.8, 127.6, 115.6, 109.9, 107.1,
01216     105.7, 105, 104.8, 104.8, 104.9, 105, 105.1, 105.3, 105.5, 105.8,
01217     106.4, 107, 107.6, 108.1, 108.8, 110, 111.8, 114.2, 117.4, 121.6,
01218     127.9, 137.3, 151.2, 169.5, 189, 205.8, 218.9, 229.1, 237.8, 245,
01219     251.5, 257.1, 262.3, 268.2, 274, 280.4, 286.7, 292.4, 297.9, 302.9,
01220     308.5, 312.2, 313.1, 313.3},
01221    {187.4, 184.5, 173.3, 166.1, 165.4, 167.8, 169.6, 173.6, 179.6,
01222     187.9, 198.9, 210, 220.5, 229.2, 235.7, 239.9, 241.8, 241.6, 239.6,
01223     235.8, 229.4, 218.6, 200.9, 175.9, 149.4, 129.4, 118.3, 113.1,
01224     110.8, 109.7, 109.3, 109.4, 109.7, 110, 110.2, 110.4, 110.5, 110.7,
01225     111, 111.4, 111.8, 112.1, 112.3, 112.7, 113.2, 113.9, 115, 116.4,
01226     117.9, 120.4, 124.1, 130.9, 142.2, 159.6, 179.6, 198.5, 212.9,
01227     224.2, 232.7, 239.1, 243.8, 247.7, 252.4, 257.3, 263.2, 269.5,
01228     275.4, 281.1, 286.3, 292, 296.3, 298.2, 298.8},
01229    {166, 166.4, 155.7, 148.3, 147.1, 149, 152.1, 157, 163.6, 172.4,
01230     185.3, 199.2, 212.6, 224, 233.2, 239.6, 243.3, 244.6, 243.6, 240.3,
01231     233.9, 222.6, 203.7, 177, 149.5, 129.7, 119, 114, 111.7, 110.7,
01232     110.3, 110.3, 110.6, 110.9, 111.1, 111.3, 111.5, 111.6, 111.9,
01233     112.2, 112.5, 112.6, 112.8, 113, 113.4, 114, 115.1, 116.5, 118.3,
01234     120.9, 124.4, 130.2, 139.4, 154.6, 173.8, 193.1, 208.1, 220.4,
01235     230.1, 238.2, 244.7, 249.5, 254.5, 259.3, 264.5, 269.4, 273.7,
01236     278.2, 282.6, 287.4, 290.9, 292.5, 293},
01237    {171.9, 172.8, 166.2, 162.3, 161.4, 162.5, 165.2, 169.6, 175.3,
01238     183.1, 193.8, 205.9, 218.3, 229.6, 238.5, 244.3, 246.9, 246.7,
01239     243.8, 238.4, 230.2, 217.9, 199.6, 174.9, 148.9, 129.8, 119.5,
01240     114.8, 112.3, 110.9, 110.3, 110.1, 110.2, 110.3, 110.4, 110.5,
01241     110.6, 110.8, 111, 111.4, 111.8, 112, 112.2, 112.4, 112.9, 113.6,
01242     114.7, 116.3, 118.4, 121.9, 127.1, 136.1, 149.8, 168.4, 186.9,
01243     203.3, 217, 229.1, 238.7, 247, 254, 259.3, 264.3, 268.3, 272.5,
01244     276.6, 280.4, 284.4, 288.4, 293.3, 297.2, 298.7, 299.1},
01245    {191.6, 192.2, 189, 188.1, 190.2, 193.7, 197.8, 202.9, 208.5,
01246     215.6, 224.2, 233.1, 241.2, 247.3, 250.8, 251.3, 248.9, 244.2,
01247     237.3, 228.4, 217.2, 202.9, 184.5, 162.5, 140.7, 124.8, 116.2,
01248     111.8, 109.4, 107.9, 107, 106.7, 106.6, 106.6, 106.7, 106.7,
01249     106.8, 107, 107.4, 108, 108.7, 109.3, 109.8, 110.4, 111.2,
01250     112.4, 114.2, 116.9, 121.1, 127.9, 139.3, 155.2, 173.6, 190.7,
01251     206.1, 220.1, 232.3, 243, 251.8, 259.2, 265.7, 270.6, 275.3,
01252     279.3, 283.3, 286.9, 289.7, 292.8, 296.1, 300.5, 303.9, 304.8,
01253     305.1},
01254    {241.5, 239.6, 236.8, 237.4, 239.4, 242.3, 244.2, 246.4, 249.2,
01255     253.6, 258.6, 262.7, 264.8, 264.2, 260.6, 254.1, 245.5, 235.3,
01256     223.9, 211.7, 198.3, 183.1, 165.6, 147.1, 130.5, 118.7, 111.9,
01257     108.1, 105.8, 104.3, 103.4, 102.8, 102.5, 102.4, 102.5, 102.5,
01258     102.5, 102.7, 103.1, 103.8, 104.6, 105.4, 106.1, 107, 108.2,
```

```
01259     109.9, 112.8, 117.5, 126, 140.4, 161, 181.9, 201.2, 216.8, 230.4,
01260     241.8, 251.4, 259.9, 266.9, 272.8, 277.4, 280.4, 282.9, 284.6,
01261     286.1, 287.4, 288.3, 289.5, 290.9, 294.2, 296.9, 297.5, 297.6},
01262    {301.2, 300.3, 296.6, 295.4, 295, 294.3, 291.2, 287.4, 284.9, 284.7,
01263     284.1, 281.5, 277.1, 270.4, 261.7, 250.6, 237.6, 223.1, 207.9, 192,
01264     175.8, 158.8, 142.1, 127.6, 116.8, 109.9, 106, 103.6, 102.1, 101.1,
01265     100.4, 99.96, 99.6, 99.37, 99.32, 99.32, 99.31, 99.46, 99.77, 100.2,
01266     100.7, 101.3, 101.8, 102.7, 104.1, 106.8, 111.9, 121, 136.7, 160,
01267     186.9, 209.9, 228.1, 241.2, 251.5, 259.5, 265.7, 270.9, 274.8, 278,
01268     280.3, 281.8, 283, 283.3, 283.7, 283.8, 283, 282.2, 281.2, 281.4,
01269     281.7, 281.1, 281.2}
01270    };
01271
01272    double doy, p0, p1, pt;
01273
01274    int imon, ilat;
01275
01276    /* Get day of year... */
01277    doy = fmod(t / 86400., 365.25);
01278    while (doy < 0)
01279      doy += 365.25;
01280
01281    /* Get indices... */
01282    imon = locate(doys, 12, doy);
01283    ilat = locate(lats, 73, lat);
01284
01285    /* Get tropopause pressure... */
01286    p0 = LIN(lats[ilat], tps[imon][ilat],
01287             lats[ilat + 1], tps[imon][ilat + 1], lat);
01288    p1 = LIN(lats[ilat], tps[imon + 1][ilat],
01289             lats[ilat + 1], tps[imon + 1][ilat + 1], lat);
01290    pt = LIN(doys[imon], p0, doys[imon + 1], p1, doy);
01291
01292    /* Return tropopause pressure... */
01293    return pt;
01294 }
```

Here is the call graph for this function:



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

Write atmospheric data.

Definition at line 1298 of file libtrac.c.

```
01302             {
01303
01304    FILE *in, *out;
01305
01306    char line[LEN];
01307
01308    double r, t0, t1;
01309
01310    int ip, iq, year, mon, day, hour, min, sec;
01311
01312    /* Set time interval for output... */
01313    t0 = t - 0.5 * ctl->dt_mod;
01314    t1 = t + 0.5 * ctl->dt_mod;
01315
01316    /* Check if gnuplot output is requested... */
01317    if (ctl->atm_gpfile[0] != '-') {
01318
```

```
01319     /* Write info... */
01320     printf("Plot atmospheric data: %s.png\n", filename);
01321
01322     /* Create gnuplot pipe... */
01323     if (!(out = popen("gnuplot", "w")))
01324       ERRMSG("Cannot create pipe to gnuplot!");
01325
01326     /* Set plot filename... */
01327     fprintf(out, "set out \"%s.png\"\n", filename);
01328
01329     /* Set time string... */
01330     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01331     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01332             year, mon, day, hour, min);
01333
01334     /* Dump gnuplot file to pipe... */
01335     if (!(in = fopen(ctl->atm_gpfile, "r")))
01336       ERRMSG("Cannot open file!");
01337     while (fgets(line, LEN, in))
01338       fprintf(out, "%s", line);
01339     fclose(in);
01340   }
01341
01342   else {
01343
01344     /* Write info... */
01345     printf("Write atmospheric data: %s\n", filename);
01346
01347     /* Create file... */
01348     if (!(out = fopen(filename, "w")))
01349       ERRMSG("Cannot create file!");
01350   }
01351
01352   /* Write binary data... */
01353   if (ctl->atm_bin && ctl->atm_gpfile[0] == '-') {
01354     FWRITE(&atm->np, int,
01355            1,
01356            out);
01357     FWRITE(atm->time, double,
01358              (size_t) atm->np,
01359            out);
01360     FWRITE(atm->p, double,
01361              (size_t) atm->np,
01362            out);
01363     FWRITE(atm->lon, double,
01364              (size_t) atm->np,
01365            out);
01366     FWRITE(atm->lat, double,
01367              (size_t) atm->np,
01368            out);
01369     for (iq = 0; iq < ctl->nq; iq++)
01370       FWRITE(atm->q[iq], double,
01371                (size_t) atm->np,
01372              out);
01373   }
01374
01375   /* Write ASCII data... */
01376   else {
01377
01378     /* Write header... */
01379     fprintf(out,
01380             "# $1 = time [s]\n"
01381             "# $2 = altitude [km]\n"
01382             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01383     for (iq = 0; iq < ctl->nq; iq++)
01384       fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl->qnt_name[iq],
01385               ctl->qnt_unit[iq]);
01386     fprintf(out, "\n");
01387
01388     /* Write data... */
01389     for (ip = 0; ip < atm->np; ip++) {
01390
01391       /* Check time... */
01392       if (ctl->atm_filter && (atm->time[ip] < t0 || atm->time[ip] > t1))
01393         continue;
01394
01395       /* Write output... */
01396       fprintf(out, "%.2f %g %g %g", atm->time[ip], Z(atm->p[ip]),
01397               atm->lon[ip], atm->lat[ip]);
01398       for (iq = 0; iq < ctl->nq; iq++) {
01399         fprintf(out, " ");
01400         fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
01401       }
01402       fprintf(out, "\n");
01403     }
01404   }
01405
```

```
01406   /* Close file... */
01407   fclose(out);
01408 }
```

Here is the call graph for this function:

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

Write CSI data.

Definition at line 1412 of file libtrac.c.

```
01416            {
01417
01418   static FILE *in, *out;
01419
01420   static char line[LEN];
01421
01422   static double modmean[GX][GY][GZ], obsmean[GX][GY][GZ],
01423     rt, rz, rlon, rlat, robs, t0, t1, area, dlon, dlat, lat;
01424
01425   static int init, obscount[GX][GY][GZ], cx, cy, cz, ip, ix, iy, iz;
01426
01427   /* Init... */
01428   if (!init) {
01429     init = 1;
01430
01431     /* Check quantity index for mass... */
01432     if (ctl->qnt_m < 0)
01433       ERRMSG("Need quantity mass to analyze CSI!");
01434
01435     /* Open observation data file... */
01436     printf("Read CSI observation data: %s\n", ctl->csi_obsfile);
01437     if (!(in = fopen(ctl->csi_obsfile, "r")))
01438       ERRMSG("Cannot open file!");
01439
01440     /* Create new file... */
01441     printf("Write CSI data: %s\n", filename);
01442     if (!(out = fopen(filename, "w")))
01443       ERRMSG("Cannot create file!");
01444
01445     /* Write header... */
01446     fprintf(out,
01447             "# $1 = time [s]\n"
01448             "# $2 = number of hits (cx)\n"
01449             "# $3 = number of misses (cy)\n"
01450             "# $4 = number of false alarms (cz)\n"
01451             "# $5 = number of observations (cx + cy)\n"
01452             "# $6 = number of forecasts (cx + cz)\n"
01453             "# $7 = bias (forecasts/observations) [%%]\n"
01454             "# $8 = probability of detection (POD) [%%]\n"
01455             "# $9 = false alarm rate (FAR) [%%]\n"
01456             "# $10 = critical success index (CSI) [%%]\n\n");
01457   }
01458
01459   /* Set time interval... */
01460   t0 = t - 0.5 * ctl->dt_mod;
01461   t1 = t + 0.5 * ctl->dt_mod;
01462
01463   /* Initialize grid cells... */
01464   for (ix = 0; ix < ctl->csi_nx; ix++)
01465     for (iy = 0; iy < ctl->csi_ny; iy++)
```

```
01466       for (iz = 0; iz < ctl->csi_nz; iz++)
01467         modmean[ix][iy][iz] = obsmean[ix][iy][iz] = obscount[ix][iy][iz] = 0;
01468
01469     /* Read data... */
01470     while (fgets(line, LEN, in)) {
01471
01472       /* Read data... */
01473       if (sscanf(line, "%lg %lg %lg %lg %lg", &rt, &rz, &rlon, &rlat, &robs) !=
01474           5)
01475         continue;
01476
01477       /* Check time... */
01478       if (rt < t0)
01479         continue;
01480       if (rt > t1)
01481         break;
01482
01483       /* Calculate indices... */
01484       ix = (int) ((rlon - ctl->csi_lon0)
01485                   / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01486       iy = (int) ((rlat - ctl->csi_lat0)
01487                   / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01488       iz = (int) ((rz - ctl->csi_z0)
01489                   / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01490
01491       /* Check indices... */
01492       if (ix < 0 || ix >= ctl->csi_nx ||
01493           iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01494         continue;
01495
01496       /* Get mean observation index... */
01497       obsmean[ix][iy][iz] += robs;
01498       obscount[ix][iy][iz]++;
01499     }
01500
01501     /* Analyze model data... */
01502     for (ip = 0; ip < atm->np; ip++) {
01503
01504       /* Check time... */
01505       if (atm->time[ip] < t0 || atm->time[ip] > t1)
01506         continue;
01507
01508       /* Get indices... */
01509       ix = (int) ((atm->lon[ip] - ctl->csi_lon0)
01510                   / (ctl->csi_lon1 - ctl->csi_lon0) * ctl->csi_nx);
01511       iy = (int) ((atm->lat[ip] - ctl->csi_lat0)
01512                   / (ctl->csi_lat1 - ctl->csi_lat0) * ctl->csi_ny);
01513       iz = (int) ((Z(atm->p[ip]) - ctl->csi_z0)
01514                   / (ctl->csi_z1 - ctl->csi_z0) * ctl->csi_nz);
01515
01516       /* Check indices... */
01517       if (ix < 0 || ix >= ctl->csi_nx ||
01518           iy < 0 || iy >= ctl->csi_ny || iz < 0 || iz >= ctl->csi_nz)
01519         continue;
01520
01521       /* Get total mass in grid cell... */
01522       modmean[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01523     }
01524
01525     /* Analyze all grid cells... */
01526     for (ix = 0; ix < ctl->csi_nx; ix++)
01527       for (iy = 0; iy < ctl->csi_ny; iy++)
01528         for (iz = 0; iz < ctl->csi_nz; iz++) {
01529
01530           /* Calculate mean observation index... */
01531           if (obscount[ix][iy][iz] > 0)
01532             obsmean[ix][iy][iz] /= obscount[ix][iy][iz];
01533
01534           /* Calculate column density... */
01535           if (modmean[ix][iy][iz] > 0) {
01536             dlon = (ctl->csi_lon1 - ctl->csi_lon0) / ctl->csi_nx;
01537             dlat = (ctl->csi_lat1 - ctl->csi_lat0) / ctl->csi_ny;
01538             lat = ctl->csi_lat0 + dlat * (iy + 0.5);
01539             area = dlat * M_PI * RE / 180. * dlon * M_PI * RE / 180.
01540               * cos(lat * M_PI / 180.);
01541             modmean[ix][iy][iz] /= (1e6 * area);
01542           }
01543
01544           /* Calculate CSI... */
01545           if (obscount[ix][iy][iz] > 0) {
01546             if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01547                 modmean[ix][iy][iz] >= ctl->csi_modmin)
01548               cx++;
01549             else if (obsmean[ix][iy][iz] >= ctl->csi_obsmin &&
01550                      modmean[ix][iy][iz] < ctl->csi_modmin)
01551               cy++;
01552             else if (obsmean[ix][iy][iz] < ctl->csi_obsmin &&
```

```
01553                   modmean[ix][iy][iz] >= ctl->csi_modmin)
01554             cz++;
01555         }
01556       }
01557
01558   /* Write output... */
01559   if (fmod(t, ctl->csi_dt_out) == 0) {
01560
01561     /* Write... */
01562     fprintf(out, "%.2f %d %d %d %d %d %g %g %g %g\n",
01563             t, cx, cy, cz, cx + cy, cx + cz,
01564             (cx + cy > 0) ? 100. * (cx + cz) / (cx + cy) : GSL_NAN,
01565             (cx + cy > 0) ? (100. * cx) / (cx + cy) : GSL_NAN,
01566             (cx + cz > 0) ? (100. * cz) / (cx + cz) : GSL_NAN,
01567             (cx + cy + cz > 0) ? (100. * cx) / (cx + cy + cz) : GSL_NAN);
01568
01569     /* Set counters to zero... */
01570     cx = cy = cz = 0;
01571   }
01572
01573   /* Close file... */
01574   if (t == ctl->t_stop)
01575     fclose(out);
01576 }
```

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

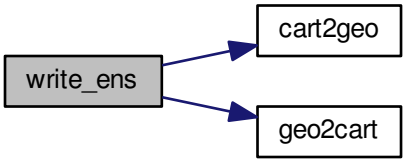
Write ensemble data.

Definition at line 1580 of file libtrac.c.

```
01584             {
01585
01586   static FILE *out;
01587
01588   static double dummy, ens, lat, lon, p[NENS], q[NQ][NENS],
01589     t0, t1, x[NENS][3], xm[3];
01590
01591   static int init, ip, iq;
01592
01593   static size_t i, n;
01594
01595   /* Init... */
01596   if (!init) {
01597     init = 1;
01598
01599     /* Check quantities... */
01600     if (ctl->qnt_ens < 0)
01601       ERRMSG("Missing ensemble IDs!");
01602
01603     /* Create new file... */
01604     printf("Write ensemble data: %s\n", filename);
01605     if (!(out = fopen(filename, "w")))
01606       ERRMSG("Cannot create file!");
01607
01608     /* Write header... */
01609     fprintf(out,
01610             "# $1 = time [s]\n"
01611             "# $2 = altitude [km]\n"
01612             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
01613     for (iq = 0; iq < ctl->nq; iq++)
01614       fprintf(out, "# $%d = %s (mean) [%s]\n", 5 + iq,
01615               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01616     for (iq = 0; iq < ctl->nq; iq++)
01617       fprintf(out, "# $%d = %s (sigma) [%s]\n", 5 + ctl->nq + iq,
01618               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
01619     fprintf(out, "# $%d = number of members\n\n", 5 + 2 * ctl->nq);
01620   }
01621
01622   /* Set time interval... */
01623   t0 = t - 0.5 * ctl->dt_mod;
01624   t1 = t + 0.5 * ctl->dt_mod;
01625
01626   /* Init... */
01627   ens = GSL_NAN;
01628   n = 0;
01629
01630   /* Loop over air parcels... */
01631   for (ip = 0; ip < atm->np; ip++) {
01632
```

```
01633     /* Check time... */
01634     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01635       continue;
01636
01637     /* Check ensemble id... */
01638     if (atm->q[ctl->qnt_ens][ip] != ens) {
01639
01640       /* Write results... */
01641       if (n > 0) {
01642
01643         /* Get mean position... */
01644         xm[0] = xm[1] = xm[2] = 0;
01645         for (i = 0; i < n; i++) {
01646           xm[0] += x[i][0] / (double) n;
01647           xm[1] += x[i][1] / (double) n;
01648           xm[2] += x[i][2] / (double) n;
01649         }
01650         cart2geo(xm, &dummy, &lon, &lat);
01651         fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon,
01652                 lat);
01653
01654         /* Get quantity statistics... */
01655         for (iq = 0; iq < ctl->nq; iq++) {
01656           fprintf(out, " ");
01657           fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01658         }
01659         for (iq = 0; iq < ctl->nq; iq++) {
01660           fprintf(out, " ");
01661           fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01662         }
01663         fprintf(out, " %lu\n", n);
01664       }
01665
01666       /* Init new ensemble... */
01667       ens = atm->q[ctl->qnt_ens][ip];
01668       n = 0;
01669     }
01670
01671     /* Save data... */
01672     p[n] = atm->p[ip];
01673     geo2cart(0, atm->lon[ip], atm->lat[ip], x[n]);
01674     for (iq = 0; iq < ctl->nq; iq++)
01675       q[iq][n] = atm->q[iq][ip];
01676     if ((++n) >= NENS)
01677       ERRMSG("Too many data points!");
01678   }
01679
01680   /* Write results... */
01681   if (n > 0) {
01682
01683     /* Get mean position... */
01684     xm[0] = xm[1] = xm[2] = 0;
01685     for (i = 0; i < n; i++) {
01686       xm[0] += x[i][0] / (double) n;
01687       xm[1] += x[i][1] / (double) n;
01688       xm[2] += x[i][2] / (double) n;
01689     }
01690     cart2geo(xm, &dummy, &lon, &lat);
01691     fprintf(out, "%.2f %g %g %g", t, Z(gsl_stats_mean(p, 1, n)), lon, lat);
01692
01693     /* Get quantity statistics... */
01694     for (iq = 0; iq < ctl->nq; iq++) {
01695       fprintf(out, " ");
01696       fprintf(out, ctl->qnt_format[iq], gsl_stats_mean(q[iq], 1, n));
01697     }
01698     for (iq = 0; iq < ctl->nq; iq++) {
01699       fprintf(out, " ");
01700       fprintf(out, ctl->qnt_format[iq], gsl_stats_sd(q[iq], 1, n));
01701     }
01702     fprintf(out, " %lu\n", n);
01703   }
01704
01705   /* Close file... */
01706   if (t == ctl->t_stop)
01707     fclose(out);
01708 }
```

Here is the call graph for this function:



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

Write gridded data.

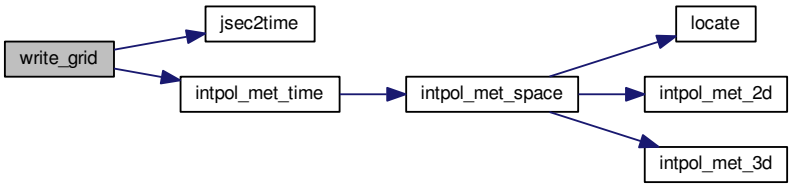
Definition at line 1712 of file libtrac.c.

```
01718            {
01719
01720   FILE *in, *out;
01721
01722   char line[LEN];
01723
01724   static double grid_m[GX][GY][GZ], z, dz, lon, dlon, lat, dlat,
01725     area, rho_air, press, temp, cd, mmr, t0, t1, r;
01726
01727   static int ip, ix, iy, iz, year, mon, day, hour, min, sec;
01728
01729   /* Check dimensions... */
01730   if (ctl->grid_nx > GX || ctl->grid_ny > GY || ctl->grid_nz > GZ)
01731     ERRMSG("Grid dimensions too large!");
01732
01733   /* Check quantity index for mass... */
01734   if (ctl->qnt_m < 0)
01735     ERRMSG("Need quantity mass to write grid data!");
01736
01737   /* Set time interval for output... */
01738   t0 = t - 0.5 * ctl->dt_mod;
01739   t1 = t + 0.5 * ctl->dt_mod;
01740
01741   /* Set grid box size... */
01742   dz = (ctl->grid_z1 - ctl->grid_z0) / ctl->grid_nz;
01743   dlon = (ctl->grid_lon1 - ctl->grid_lon0) / ctl->grid_nx;
01744   dlat = (ctl->grid_lat1 - ctl->grid_lat0) / ctl->grid_ny;
01745
01746   /* Initialize grid... */
01747   for (ix = 0; ix < ctl->grid_nx; ix++)
01748     for (iy = 0; iy < ctl->grid_ny; iy++)
01749       for (iz = 0; iz < ctl->grid_nz; iz++)
01750         grid_m[ix][iy][iz] = 0;
01751
01752   /* Average data... */
01753   for (ip = 0; ip < atm->np; ip++)
01754     if (atm->time[ip] >= t0 && atm->time[ip] <= t1) {
01755
01756       /* Get index... */
01757       ix = (int) ((atm->lon[ip] - ctl->grid_lon0) / dlon);
01758       iy = (int) ((atm->lat[ip] - ctl->grid_lat0) / dlat);
01759       iz = (int) ((Z(atm->p[ip]) - ctl->grid_z0) / dz);
01760
01761       /* Check indices... */
01762       if (ix < 0 || ix >= ctl->grid_nx ||
01763           iy < 0 || iy >= ctl->grid_ny || iz < 0 || iz >= ctl->grid_nz)
01764         continue;
01765
01766       /* Add mass... */
01767       grid_m[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01768     }
01769
```

```
01770   /* Check if gnuplot output is requested... */
01771   if (ctl->grid_gpfile[0] != '-') {
01772
01773     /* Write info... */
01774     printf("Plot grid data: %s.png\n", filename);
01775
01776     /* Create gnuplot pipe... */
01777     if (!(out = popen("gnuplot", "w")))
01778       ERRMSG("Cannot create pipe to gnuplot!");
01779
01780     /* Set plot filename... */
01781     fprintf(out, "set out \"%s.png\"\n", filename);
01782
01783     /* Set time string... */
01784     jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01785     fprintf(out, "timestr=\"%d-%02d-%02d, %02d:%02d UTC\"\n",
01786             year, mon, day, hour, min);
01787
01788     /* Dump gnuplot file to pipe... */
01789     if (!(in = fopen(ctl->grid_gpfile, "r")))
01790       ERRMSG("Cannot open file!");
01791     while (fgets(line, LEN, in))
01792       fprintf(out, "%s", line);
01793     fclose(in);
01794   }
01795
01796   else {
01797
01798     /* Write info... */
01799     printf("Write grid data: %s\n", filename);
01800
01801     /* Create file... */
01802     if (!(out = fopen(filename, "w")))
01803       ERRMSG("Cannot create file!");
01804   }
01805
01806   /* Write header... */
01807   fprintf(out,
01808           "# $1 = time [s]\n"
01809           "# $2 = altitude [km]\n"
01810           "# $3 = longitude [deg]\n"
01811           "# $4 = latitude [deg]\n"
01812           "# $5 = surface area [km^2]\n"
01813           "# $6 = layer width [km]\n"
01814           "# $7 = temperature [K]\n"
01815           "# $8 = column density [kg/m^2]\n"
01816           "# $9 = mass mixing ratio [1]\n\n");
01817
01818   /* Write data... */
01819   for (ix = 0; ix < ctl->grid_nx; ix++) {
01820     if (ix > 0 && ctl->grid_ny > 1 && !ctl->grid_sparse)
01821       fprintf(out, "\n");
01822     for (iy = 0; iy < ctl->grid_ny; iy++) {
01823       if (iy > 0 && ctl->grid_nz > 1 && !ctl->grid_sparse)
01824         fprintf(out, "\n");
01825       for (iz = 0; iz < ctl->grid_nz; iz++)
01826         if (!ctl->grid_sparse
01827             || ix == 0 || iy == 0 || iz == 0 || grid_m[ix][iy][iz] > 0) {
01828
01829           /* Set coordinates... */
01830           z = ctl->grid_z0 + dz * (iz + 0.5);
01831           lon = ctl->grid_lon0 + dlon * (ix + 0.5);
01832           lat = ctl->grid_lat0 + dlat * (iy + 0.5);
01833
01834           /* Get pressure and temperature... */
01835           press = P(z);
01836           intpol_met_time(met0, met1, t, press, lon, lat,
01837                           NULL, &temp, NULL, NULL, NULL, NULL, NULL);
01838
01839           /* Calculate surface area... */
01840           area = dlat * dlon * gsl_pow_2(RE * M_PI / 180.)
01841             * cos(lat * M_PI / 180.);
01842
01843           /* Calculate column density... */
01844           cd = grid_m[ix][iy][iz] / (1e6 * area);
01845
01846           /* Calculate mass mixing ratio... */
01847           rho_air = 100. * press / (287.058 * temp);
01848           mmr = grid_m[ix][iy][iz] / (rho_air * 1e6 * area * 1e3 * dz);
01849
01850           /* Write output... */
01851           fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01852                   t, z, lon, lat, area, dz, temp, cd, mmr);
01853         }
01854     }
01855   }
01856
```

```
01857   /* Close file... */
01858   fclose(out);
01859 }
```

Here is the call graph for this function:



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

Write profile data.

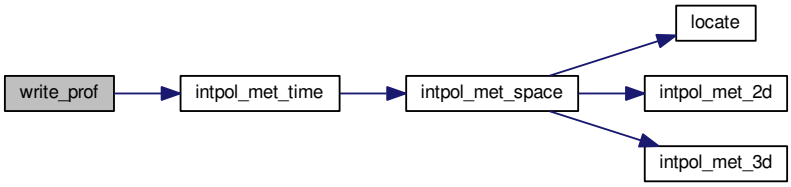
Definition at line 1863 of file libtrac.c.

```
01869             {
01870
01871   static FILE *in, *out;
01872
01873   static char line[LEN];
01874
01875   static double mass[GX][GY][GZ], obsmean[GX][GY], tmean[GX][GY],
01876     rt, rlon, rlat, robs, t0, t1, area, dz, dlon, dlat, lon, lat, z,
01877     press, temp, rho_air, mmr, h2o, o3;
01878
01879   static int init, obscount[GX][GY], ip, ix, iy, iz;
01880
01881   /* Init... */
01882   if (!init) {
01883     init = 1;
01884
01885     /* Check quantity index for mass... */
01886     if (ctl->qnt_m < 0)
01887       ERRMSG("Need quantity mass!");
01888
01889     /* Check dimensions... */
01890     if (ctl->prof_nx > GX || ctl->prof_ny > GY || ctl->prof_nz > GZ)
01891       ERRMSG("Grid dimensions too large!");
01892
01893     /* Open observation data file... */
01894     printf("Read profile observation data: %s\n", ctl->prof_obsfile);
01895     if (!(in = fopen(ctl->prof_obsfile, "r")))
01896       ERRMSG("Cannot open file!");
01897
01898     /* Create new file... */
01899     printf("Write profile data: %s\n", filename);
01900     if (!(out = fopen(filename, "w")))
01901       ERRMSG("Cannot create file!");
01902
01903     /* Write header... */
01904     fprintf(out,
01905             "# $1  = time [s]\n"
01906             "# $2  = altitude [km]\n"
01907             "# $3  = longitude [deg]\n"
01908             "# $4  = latitude [deg]\n"
01909             "# $5  = pressure [hPa]\n"
01910             "# $6  = temperature [K]\n"
01911             "# $7  = mass mixing ratio [1]\n"
01912             "# $8  = H2O volume mixing ratio [1]\n"
01913             "# $9  = O3 volume mixing ratio [1]\n"
01914             "# $10 = mean BT index [K]\n");
01915
01916     /* Set grid box size... */
```

```
01917     dz = (ctl->prof_z1 - ctl->prof_z0) / ctl->prof_nz;
01918     dlon = (ctl->prof_lon1 - ctl->prof_lon0) / ctl->prof_nx;
01919     dlat = (ctl->prof_lat1 - ctl->prof_lat0) / ctl->prof_ny;
01920   }
01921
01922   /* Set time interval... */
01923   t0 = t - 0.5 * ctl->dt_mod;
01924   t1 = t + 0.5 * ctl->dt_mod;
01925
01926   /* Initialize... */
01927   for (ix = 0; ix < ctl->prof_nx; ix++)
01928     for (iy = 0; iy < ctl->prof_ny; iy++) {
01929       obsmean[ix][iy] = 0;
01930       obscount[ix][iy] = 0;
01931       tmean[ix][iy] = 0;
01932       for (iz = 0; iz < ctl->prof_nz; iz++)
01933         mass[ix][iy][iz] = 0;
01934     }
01935
01936   /* Read data... */
01937   while (fgets(line, LEN, in)) {
01938
01939     /* Read data... */
01940     if (sscanf(line, "%lg %lg %lg %lg", &rt, &rlon, &rlat, &robs) != 4)
01941       continue;
01942
01943     /* Check time... */
01944     if (rt < t0)
01945       continue;
01946     if (rt > t1)
01947       break;
01948
01949     /* Calculate indices... */
01950     ix = (int) ((rlon - ctl->prof_lon0) / dlon);
01951     iy = (int) ((rlat - ctl->prof_lat0) / dlat);
01952
01953     /* Check indices... */
01954     if (ix < 0 || ix >= ctl->prof_nx || iy < 0 || iy >= ctl->prof_ny)
01955       continue;
01956
01957     /* Get mean observation index... */
01958     obsmean[ix][iy] += robs;
01959     tmean[ix][iy] += rt;
01960     obscount[ix][iy]++;
01961   }
01962
01963   /* Analyze model data... */
01964   for (ip = 0; ip < atm->np; ip++) {
01965
01966     /* Check time... */
01967     if (atm->time[ip] < t0 || atm->time[ip] > t1)
01968       continue;
01969
01970     /* Get indices... */
01971     ix = (int) ((atm->lon[ip] - ctl->prof_lon0) / dlon);
01972     iy = (int) ((atm->lat[ip] - ctl->prof_lat0) / dlat);
01973     iz = (int) ((Z(atm->p[ip]) - ctl->prof_z0) / dz);
01974
01975     /* Check indices... */
01976     if (ix < 0 || ix >= ctl->prof_nx ||
01977         iy < 0 || iy >= ctl->prof_ny || iz < 0 || iz >= ctl->prof_nz)
01978       continue;
01979
01980     /* Get total mass in grid cell... */
01981     mass[ix][iy][iz] += atm->q[ctl->qnt_m][ip];
01982   }
01983
01984   /* Extract profiles... */
01985   for (ix = 0; ix < ctl->prof_nx; ix++)
01986     for (iy = 0; iy < ctl->prof_ny; iy++)
01987       if (obscount[ix][iy] > 0) {
01988
01989         /* Write output... */
01990         fprintf(out, "\n");
01991
01992         /* Loop over altitudes... */
01993         for (iz = 0; iz < ctl->prof_nz; iz++) {
01994
01995           /* Set coordinates... */
01996           z = ctl->prof_z0 + dz * (iz + 0.5);
01997           lon = ctl->prof_lon0 + dlon * (ix + 0.5);
01998           lat = ctl->prof_lat0 + dlat * (iy + 0.5);
01999
02000           /* Get meteorological data... */
02001           press = P(z);
02002           intpol_met_time(met0, met1, t, press, lon, lat,
02003                           NULL, &temp, NULL, NULL, NULL, &h2o, &o3);
```

```
02004
02005           /* Calculate mass mixing ratio... */
02006           rho_air = 100. * press / (287.058 * temp);
02007           area = dlat * dlon * gsl_pow_2(M_PI * RE / 180.)
02008             * cos(lat * M_PI / 180.);
02009           mmr = mass[ix][iy][iz] / (rho_air * area * dz * 1e9);
02010
02011           /* Write output... */
02012           fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
02013                   tmean[ix][iy] / obscount[ix][iy],
02014                   z, lon, lat, press, temp, mmr, h2o, o3,
02015                   obsmean[ix][iy] / obscount[ix][iy]);
02016         }
02017       }
02018
02019   /* Close file... */
02020   if (t == ctl->t_stop)
02021     fclose(out);
02022 }
```

Here is the call graph for this function:



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

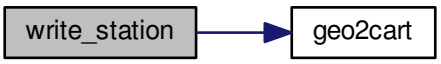
Write station data.

Definition at line 2026 of file libtrac.c.

```
02030           {
02031
02032   static FILE *out;
02033
02034   static double rmax2, t0, t1, x0[3], x1[3];
02035
02036   static int init, ip, iq;
02037
02038   /* Init... */
02039   if (!init) {
02040     init = 1;
02041
02042     /* Write info... */
02043     printf("Write station data: %s\n", filename);
02044
02045     /* Create new file... */
02046     if (!(out = fopen(filename, "w")))
02047       ERRMSG("Cannot create file!");
02048
02049     /* Write header... */
02050     fprintf(out,
02051             "# $1 = time [s]\n"
02052             "# $2 = altitude [km]\n"
02053             "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
02054     for (iq = 0; iq < ctl->nq; iq++)
02055       fprintf(out, "# $%i = %s [%s]\n", (iq + 5),
02056               ctl->qnt_name[iq], ctl->qnt_unit[iq]);
02057     fprintf(out, "\n");
02058
02059     /* Set geolocation and search radius... */
02060     geo2cart(0, ctl->stat_lon, ctl->stat_lat, x0);
02061     rmax2 = gsl_pow_2(ctl->stat_r);
```

```
02062   }
02063
02064   /* Set time interval for output... */
02065   t0 = t - 0.5 * ctl->dt_mod;
02066   t1 = t + 0.5 * ctl->dt_mod;
02067
02068   /* Loop over air parcels... */
02069   for (ip = 0; ip < atm->np; ip++) {
02070
02071     /* Check time... */
02072     if (atm->time[ip] < t0 || atm->time[ip] > t1)
02073       continue;
02074
02075     /* Check station flag... */
02076     if (ctl->qnt_stat >= 0)
02077       if (atm->q[ctl->qnt_stat][ip])
02078         continue;
02079
02080     /* Get Cartesian coordinates... */
02081     geo2cart(0, atm->lon[ip], atm->lat[ip], x1);
02082
02083     /* Check horizontal distance... */
02084     if (DIST2(x0, x1) > rmax2)
02085       continue;
02086
02087     /* Set station flag... */
02088     if (ctl->qnt_stat >= 0)
02089       atm->q[ctl->qnt_stat][ip] = 1;
02090
02091     /* Write data... */
02092     fprintf(out, "%.2f %g %g %g",
02093             atm->time[ip], Z(atm->p[ip]), atm->lon[ip], atm->lat[ip]);
02094     for (iq = 0; iq < ctl->nq; iq++) {
02095       fprintf(out, " ");
02096       fprintf(out, ctl->qnt_format[iq], atm->q[iq][ip]);
02097     }
02098     fprintf(out, "\n");
02099   }
02100
02101   /* Close file... */
02102   if (t == ctl->t_stop)
02103     fclose(out);
02104 }
```

Here is the call graph for this function:



5.16 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   Copyright (C) 2013-2018 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    Constants...
00052    ----------------------------------------------------------- */
00053
00055 #define G0 9.80665
00056
00058 #define H0 7.0
00059
00061 #define P0 1013.25
00062
00064 #define RE 6367.421
00065
00066 /* -----------------------------------------------------------
00067    Dimensions...
00068    ----------------------------------------------------------- */
00069
00071 #define LEN 5000
00072
00074 #define NP 10000000
00075
00077 #define NQ 12
00078
00080 #define EP 111
00081
00083 #define EX 1201
00084
00086 #define EY 601
00087
00089 #define GX 720
00090
00092 #define GY 360
00093
00095 #define GZ 100
00096
00098 #define NENS 2000
00099
00101 #define NTHREADS 128
00102
00103 /* -----------------------------------------------------------
00104    Macros...
00105    ----------------------------------------------------------- */
00106
00108 #define ALLOC(ptr, type, n)                              \
00109   if((ptr=calloc((size_t)(n), sizeof(type)))==NULL)      \
00110     ERRMSG("Out of memory!");
00111
00113 #define DIST(a, b) sqrt(DIST2(a, b))
00114
00116 #define DIST2(a, b)                                                  \
00117   ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00118
00120 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00121
00123 #define ERRMSG(msg) {                                                \
00124     printf("\nError (%s, %s, l%d): %s\n\n",                          \
00125            __FILE__, __func__, __LINE__, msg);                       \
00126     exit(EXIT_FAILURE);                                              \
00127   }
00128
00130 #define FREAD(ptr, type, size, out) {                                \
00131     if(fread(ptr, sizeof(type), size, out)!=size)                    \
00132       ERRMSG("Error while reading!");                                \
00133   }
00134
00136 #define FWRITE(ptr, type, size, out) {                               \
00137     if(fwrite(ptr, sizeof(type), size, out)!=size)                   \
00138       ERRMSG("Error while writing!");                                \
00139   }
00140
```

```
00142 #define LIN(x0, y0, x1, y1, x)                  \
00143   ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00144
00146 #define NC(cmd) {                                \
00147     if((cmd)!=NC_NOERR)                          \
00148       ERRMSG(nc_strerror(cmd));                  \
00149   }
00150
00152 #define NORM(a) sqrt(DOTP(a, a))
00153
00155 #define PRINT(format, var)                                          \
00156   printf("Print (%s, %s, l%d): %s= "format"\n",                     \
00157          __FILE__, __func__, __LINE__, #var, var);
00158
00160 #define P(z) (P0*exp(-(z)/H0))
00161
00163 #define TOK(line, tok, format, var) {                               \
00164     if(((tok)=strtok((line), " \t"))) {                             \
00165       if(sscanf(tok, format, &(var))!=1) continue;                  \
00166     } else ERRMSG("Error while reading!");                          \
00167   }
00168
00170 #define Z(p) (H0*log(P0/(p)))
00171
00172 /* ------------------------------------------------------------
00173    Timers...
00174    ------------------------------------------------------------ */
00175
00177 #define START_TIMER(id) timer(#id, id, 1)
00178
00180 #define STOP_TIMER(id) timer(#id, id, 2)
00181
00183 #define PRINT_TIMER(id) timer(#id, id, 3)
00184
00186 #define NTIMER 13
00187
00189 #define TIMER_TOTAL 0
00190
00192 #define TIMER_INIT 1
00193
00195 #define TIMER_STAGE 2
00196
00198 #define TIMER_INPUT 3
00199
00201 #define TIMER_OUTPUT 4
00202
00204 #define TIMER_ADVECT 5
00205
00207 #define TIMER_DECAY 6
00208
00210 #define TIMER_DIFFMESO 7
00211
00213 #define TIMER_DIFFTURB 8
00214
00216 #define TIMER_ISOSURF 9
00217
00219 #define TIMER_METEO 10
00220
00222 #define TIMER_POSITION 11
00223
00225 #define TIMER_SEDI 12
00226
00227 /* ------------------------------------------------------------
00228    Structs...
00229    ------------------------------------------------------------ */
00230
00232 typedef struct {
00233
00235   int nq;
00236
00238   char qnt_name[NQ][LEN];
00239
00241   char qnt_unit[NQ][LEN];
00242
00244   char qnt_format[NQ][LEN];
00245
00247   int qnt_ens;
00248
00250   int qnt_m;
00251
00253   int qnt_rho;
00254
00256   int qnt_r;
00257
00259   int qnt_ps;
00260
00262   int qnt_p;
```

```
00263
00265   int qnt_t;
00266
00268   int qnt_u;
00269
00271   int qnt_v;
00272
00274   int qnt_w;
00275
00277   int qnt_h2o;
00278
00280   int qnt_o3;
00281
00283   int qnt_theta;
00284
00286   int qnt_pv;
00287
00289   int qnt_tice;
00290
00292   int qnt_tsts;
00293
00295   int qnt_tnat;
00296
00298   int qnt_stat;
00299
00301   int qnt_gw_u750;
00302
00304   int qnt_gw_v750;
00305
00307   int qnt_gw_sso;
00308
00310   int qnt_gw_var;
00311
00313   int direction;
00314
00316   double t_start;
00317
00319   double t_stop;
00320
00322   double dt_mod;
00323
00325   double dt_met;
00326
00328   int met_np;
00329
00331   double met_p[EP];
00332
00334   char met_stage[LEN];
00335
00338   int isosurf;
00339
00341   char balloon[LEN];
00342
00344   double turb_dx_trop;
00345
00347   double turb_dx_strat;
00348
00350   double turb_dz_trop;
00351
00353   double turb_dz_strat;
00354
00356   double turb_meso;
00357
00359   double tdec_trop;
00360
00362   double tdec_strat;
00363
00365   double psc_h2o;
00366
00368   double psc_hno3;
00369
00371   char gw_basename[LEN];
00372
00374   char atm_basename[LEN];
00375
00377   char atm_gpfile[LEN];
00378
00380   double atm_dt_out;
00381
00383   int atm_filter;
00384
00386   int atm_bin;
00387
00389   char csi_basename[LEN];
00390
00392   double csi_dt_out;
00393
```

```
00395   char csi_obsfile[LEN];
00396
00398   double csi_obsmin;
00399
00401   double csi_modmin;
00402
00404   int csi_nz;
00405
00407   double csi_z0;
00408
00410   double csi_z1;
00411
00413   int csi_nx;
00414
00416   double csi_lon0;
00417
00419   double csi_lon1;
00420
00422   int csi_ny;
00423
00425   double csi_lat0;
00426
00428   double csi_lat1;
00429
00431   char grid_basename[LEN];
00432
00434   char grid_gpfile[LEN];
00435
00437   double grid_dt_out;
00438
00440   int grid_sparse;
00441
00443   int grid_nz;
00444
00446   double grid_z0;
00447
00449   double grid_z1;
00450
00452   int grid_nx;
00453
00455   double grid_lon0;
00456
00458   double grid_lon1;
00459
00461   int grid_ny;
00462
00464   double grid_lat0;
00465
00467   double grid_lat1;
00468
00470   char prof_basename[LEN];
00471
00473   char prof_obsfile[LEN];
00474
00476   int prof_nz;
00477
00479   double prof_z0;
00480
00482   double prof_z1;
00483
00485   int prof_nx;
00486
00488   double prof_lon0;
00489
00491   double prof_lon1;
00492
00494   int prof_ny;
00495
00497   double prof_lat0;
00498
00500   double prof_lat1;
00501
00503   char ens_basename[LEN];
00504
00506   char stat_basename[LEN];
00507
00509   double stat_lon;
00510
00512   double stat_lat;
00513
00515   double stat_r;
00516
00517 } ctl_t;
00518
00520 typedef struct {
00521
00523   int np;
```

```
00524
00526   double time[NP];
00527
00529   double p[NP];
00530
00532   double lon[NP];
00533
00535   double lat[NP];
00536
00538   double q[NQ][NP];
00539
00541   float up[NP];
00542
00544   float vp[NP];
00545
00547   float wp[NP];
00548
00549 } atm_t;
00550
00552 typedef struct {
00553
00555   double time;
00556
00558   int nx;
00559
00561   int ny;
00562
00564   int np;
00565
00567   double lon[EX];
00568
00570   double lat[EY];
00571
00573   double p[EP];
00574
00576   double ps[EX][EY];
00577
00579   float pl[EX][EY][EP];
00580
00582   float t[EX][EY][EP];
00583
00585   float u[EX][EY][EP];
00586
00588   float v[EX][EY][EP];
00589
00591   float w[EX][EY][EP];
00592
00594   float h2o[EX][EY][EP];
00595
00597   float o3[EX][EY][EP];
00598
00599 } met_t;
00600
00601 /* ------------------------------------------------------------
00602    Functions...
00603    ------------------------------------------------------------ */
00604
00606 void cart2geo(
00607   double *x,
00608   double *z,
00609   double *lon,
00610   double *lat);
00611
00613 double deg2dx(
00614   double dlon,
00615   double lat);
00616
00618 double deg2dy(
00619   double dlat);
00620
00622 double dp2dz(
00623   double dp,
00624   double p);
00625
00627 double dx2deg(
00628   double dx,
00629   double lat);
00630
00632 double dy2deg(
00633   double dy);
00634
00636 double dz2dp(
00637   double dz,
00638   double p);
00639
00641 void geo2cart(
00642   double z,
```

```
00643   double lon,
00644   double lat,
00645   double *x);
00646
00648 void get_met(
00649   ctl_t * ctl,
00650   char *metbase,
00651   double t,
00652   met_t * met0,
00653   met_t * met1);
00654
00656 void get_met_help(
00657   double t,
00658   int direct,
00659   char *metbase,
00660   double dt_met,
00661   char *filename);
00662
00664 void intpol_met_2d(
00665   double array[EX][EY],
00666   int ix,
00667   int iy,
00668   double wx,
00669   double wy,
00670   double *var);
00671
00673 void intpol_met_3d(
00674   float array[EX][EY][EP],
00675   int ip,
00676   int ix,
00677   int iy,
00678   double wp,
00679   double wx,
00680   double wy,
00681   double *var);
00682
00684 void intpol_met_space(
00685   met_t * met,
00686   double p,
00687   double lon,
00688   double lat,
00689   double *ps,
00690   double *t,
00691   double *u,
00692   double *v,
00693   double *w,
00694   double *h2o,
00695   double *o3);
00696
00698 void intpol_met_time(
00699   met_t * met0,
00700   met_t * met1,
00701   double ts,
00702   double p,
00703   double lon,
00704   double lat,
00705   double *ps,
00706   double *t,
00707   double *u,
00708   double *v,
00709   double *w,
00710   double *h2o,
00711   double *o3);
00712
00714 void jsec2time(
00715   double jsec,
00716   int *year,
00717   int *mon,
00718   int *day,
00719   int *hour,
00720   int *min,
00721   int *sec,
00722   double *remain);
00723
00725 int locate(
00726   double *xx,
00727   int n,
00728   double x);
00729
00731 void read_atm(
00732   const char *filename,
00733   ctl_t * ctl,
00734   atm_t * atm);
00735
00737 void read_ctl(
00738   const char *filename,
00739   int argc,
```

```
00740   char *argv[],
00741   ctl_t * ctl);
00742
00744 void read_met(
00745   ctl_t * ctl,
00746   char *filename,
00747   met_t * met);
00748
00750 void read_met_extrapolate(
00751   met_t * met);
00752
00754 void read_met_help(
00755   int ncid,
00756   char *varname,
00757   char *varname2,
00758   met_t * met,
00759   float dest[EX][EY][EP],
00760   float scl);
00761
00763 void read_met_ml2pl(
00764   ctl_t * ctl,
00765   met_t * met,
00766   float var[EX][EY][EP]);
00767
00769 void read_met_periodic(
00770   met_t * met);
00771
00773 double scan_ctl(
00774   const char *filename,
00775   int argc,
00776   char *argv[],
00777   const char *varname,
00778   int arridx,
00779   const char *defvalue,
00780   char *value);
00781
00783 void time2jsec(
00784   int year,
00785   int mon,
00786   int day,
00787   int hour,
00788   int min,
00789   int sec,
00790   double remain,
00791   double *jsec);
00792
00794 void timer(
00795   const char *name,
00796   int id,
00797   int mode);
00798
00799 /* Get tropopause pressure... */
00800 double tropopause(
00801   double t,
00802   double lat);
00803
00805 void write_atm(
00806   const char *filename,
00807   ctl_t * ctl,
00808   atm_t * atm,
00809   double t);
00810
00812 void write_csi(
00813   const char *filename,
00814   ctl_t * ctl,
00815   atm_t * atm,
00816   double t);
00817
00819 void write_ens(
00820   const char *filename,
00821   ctl_t * ctl,
00822   atm_t * atm,
00823   double t);
00824
00826 void write_grid(
00827   const char *filename,
00828   ctl_t * ctl,
00829   met_t * met0,
00830   met_t * met1,
00831   atm_t * atm,
00832   double t);
00833
00835 void write_prof(
00836   const char *filename,
00837   ctl_t * ctl,
00838   met_t * met0,
00839   met_t * met1,
```

```
00840   atm_t * atm,
00841   double t);
00842
00844 void write_station(
00845   const char *filename,
00846   ctl_t * ctl,
00847   atm_t * atm,
00848   double t);
```

5.17 match.c File Reference

Calculate deviations between two trajectories.

Functions

- int main (int argc, char ∗argv[ ])

5.17.1 Detailed Description

Calculate deviations between two trajectories.

Definition in file match.c.

5.17.2 Function Documentation

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

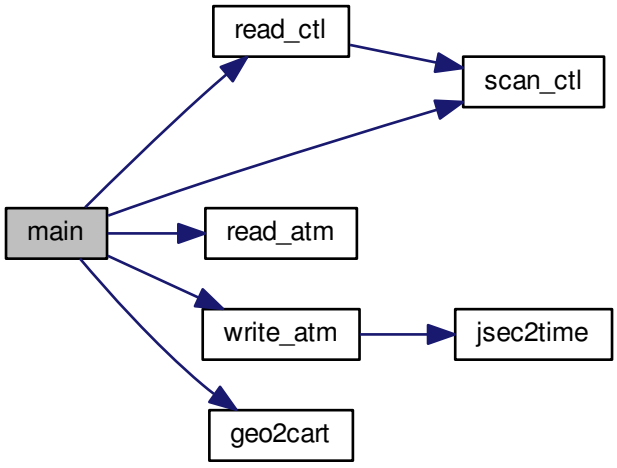
Definition at line 27 of file match.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm1, *atm2, *atm3;
00034
00035   FILE *out;
00036
00037   char filename[LEN];
00038
00039   double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00040
00041   int filter, ip1, ip2, iq, n;
00042
00043   /* Allocate... */
00044   ALLOC(atm1, atm_t, 1);
00045   ALLOC(atm2, atm_t, 1);
00046   ALLOC(atm3, atm_t, 1);
00047
00048   /* Check arguments... */
00049   if (argc < 5)
00050     ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00051
00052   /* Read control parameters... */
00053   read_ctl(argv[1], argc, argv, &ctl);
00054   filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
00055   filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00056
00057   /* Read atmospheric data... */
00058   read_atm(argv[2], &ctl, atm1);
00059   read_atm(argv[3], &ctl, atm2);
00060
00061   /* Write info... */
00062   printf("Write transport deviations: %s\n", argv[4]);
00063
00064   /* Create output file... */
```

```
00065   if (!(out = fopen(argv[4], "w")))
00066     ERRMSG("Cannot create file!");
00067
00068   /* Write header... */
00069   fprintf(out,
00070           "# $1 = time [s]\n"
00071           "# $2 = altitude [km]\n"
00072           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00073   for (iq = 0; iq < ctl.nq; iq++)
00074     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00075             ctl.qnt_unit[iq]);
00076   fprintf(out,
00077           "# $%d = trajectory time [s]\n"
00078           "# $%d = vertical length of trajectory [km]\n"
00079           "# $%d = horizontal length of trajectory [km]\n"
00080           "# $%d = vertical deviation [km]\n"
00081           "# $%d = horizontal deviation [km]\n",
00082           5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
00083   for (iq = 0; iq < ctl.nq; iq++)
00084     fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,
00085             ctl.qnt_name[iq], ctl.qnt_unit[iq]);
00086   fprintf(out, "\n");
00087
00088   /* Filtering of reference time series... */
00089   if (filter) {
00090
00091     /* Copy data... */
00092     memcpy(atm3, atm2, sizeof(atm_t));
00093
00094     /* Loop over data points... */
00095     for (ip1 = 0; ip1 < atm2->np; ip1++) {
00096       n = 0;
00097       atm2->p[ip1] = 0;
00098       for (iq = 0; iq < ctl.nq; iq++)
00099         atm2->q[iq][ip1] = 0;
00100       for (ip2 = 0; ip2 < atm2->np; ip2++)
00101         if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
00102           atm2->p[ip1] += atm3->p[ip2];
00103           for (iq = 0; iq < ctl.nq; iq++)
00104             atm2->q[iq][ip1] += atm3->q[iq][ip2];
00105           n++;
00106         }
00107       atm2->p[ip1] /= n;
00108       for (iq = 0; iq < ctl.nq; iq++)
00109         atm2->q[iq][ip1] /= n;
00110     }
00111
00112     /* Write filtered data... */
00113     sprintf(filename, "%s.filt", argv[3]);
00114     write_atm(filename, &ctl, atm2, 0);
00115   }
00116
00117   /* Loop over air parcels (reference data)... */
00118   for (ip2 = 0; ip2 < atm2->np; ip2++) {
00119
00120     /* Get trajectory length... */
00121     if (ip2 > 0) {
00122       geo2cart(0, atm2->lon[ip2 - 1], atm2->lat[ip2 - 1], x1);
00123       geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00124       lh += DIST(x1, x2);
00125       lv += fabs(Z(atm2->p[ip2 - 1]) - Z(atm2->p[ip2]));
00126       lt = fabs(atm2->time[ip2] - atm2->time[0]);
00127     }
00128
00129     /* Init... */
00130     n = 0;
00131     dh = 0;
00132     dv = 0;
00133     for (iq = 0; iq < ctl.nq; iq++)
00134       dq[iq] = 0;
00135     geo2cart(0, atm2->lon[ip2], atm2->lat[ip2], x2);
00136
00137     /* Find corresponding time step (test data)... */
00138     for (ip1 = 0; ip1 < atm1->np; ip1++)
00139       if (fabs(atm1->time[ip1] - atm2->time[ip2])
00140           < (filter ? filter_dt : 0.1)) {
00141
00142         /* Calculate deviations... */
00143         geo2cart(0, atm1->lon[ip1], atm1->lat[ip1], x1);
00144         dh += DIST(x1, x2);
00145         dv += Z(atm1->p[ip1]) - Z(atm2->p[ip2]);
00146         for (iq = 0; iq < ctl.nq; iq++)
00147           dq[iq] += atm1->q[iq][ip1] - atm2->q[iq][ip2];
00148         n++;
00149       }
00150
00151     /* Write output... */
```

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

Here is the call graph for this function:



5.18 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
00027 int main(
00028   int argc,
00029   char *argv[]) {
00030
00031   ctl_t ctl;
00032
00033   atm_t *atm1, *atm2, *atm3;
00034
00035   FILE *out;
00036
00037   char filename[LEN];
00038
00039   double filter_dt, x1[3], x2[3], dh, dq[NQ], dv, lh = 0, lt = 0, lv = 0;
00040
00041   int filter, ip1, ip2, iq, n;
00042
00043   /* Allocate... */
00044   ALLOC(atm1, atm_t, 1);
00045   ALLOC(atm2, atm_t, 1);
00046   ALLOC(atm3, atm_t, 1);
00047
00048   /* Check arguments... */
00049   if (argc < 5)
00050     ERRMSG("Give parameters: <ctl> <atm_test> <atm_ref> <outfile>");
00051
00052   /* Read control parameters... */
00053   read_ctl(argv[1], argc, argv, &ctl);
00054   filter = (int) scan_ctl(argv[1], argc, argv, "FILTER", -1, "0", NULL);
00055   filter_dt = scan_ctl(argv[1], argc, argv, "FILTER_DT", -1, "0", NULL);
00056
00057   /* Read atmospheric data... */
00058   read_atm(argv[2], &ctl, atm1);
00059   read_atm(argv[3], &ctl, atm2);
00060
00061   /* Write info... */
00062   printf("Write transport deviations: %s\n", argv[4]);
00063
00064   /* Create output file... */
00065   if (!(out = fopen(argv[4], "w")))
00066     ERRMSG("Cannot create file!");
00067
00068   /* Write header... */
00069   fprintf(out,
00070           "# $1 = time [s]\n"
00071           "# $2 = altitude [km]\n"
00072           "# $3 = longitude [deg]\n" "# $4 = latitude [deg]\n");
00073   for (iq = 0; iq < ctl.nq; iq++)
00074     fprintf(out, "# $%i = %s [%s]\n", iq + 5, ctl.qnt_name[iq],
00075             ctl.qnt_unit[iq]);
00076   fprintf(out,
00077           "# $%d = trajectory time [s]\n"
00078           "# $%d = vertical length of trajectory [km]\n"
00079           "# $%d = horizontal length of trajectory [km]\n"
00080           "# $%d = vertical deviation [km]\n"
00081           "# $%d = horizontal deviation [km]\n",
00082           5 + ctl.nq, 6 + ctl.nq, 7 + ctl.nq, 8 + ctl.nq, 9 + ctl.nq);
00083   for (iq = 0; iq < ctl.nq; iq++)
00084     fprintf(out, "# $%d = %s deviation [%s]\n", ctl.nq + iq + 10,
00085             ctl.qnt_name[iq], ctl.qnt_unit[iq]);
00086   fprintf(out, "\n");
00087
00088   /* Filtering of reference time series... */
00089   if (filter) {
00090
00091     /* Copy data... */
00092     memcpy(atm3, atm2, sizeof(atm_t));
00093
00094     /* Loop over data points... */
00095     for (ip1 = 0; ip1 < atm2->np; ip1++) {
00096       n = 0;
00097       atm2->p[ip1] = 0;
00098       for (iq = 0; iq < ctl.nq; iq++)
00099         atm2->q[iq][ip1] = 0;
00100       for (ip2 = 0; ip2 < atm2->np; ip2++)
00101         if (fabs(atm2->time[ip1] - atm2->time[ip2]) < filter_dt) {
00102           atm2->p[ip1] += atm3->p[ip2];
00103           for (iq = 0; iq < ctl.nq; iq++)
00104             atm2->q[iq][ip1] += atm3->q[iq][ip2];
00105           n++;
00106         }
```

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

5.19 met_map.c File Reference

Extract global map from meteorological data.

Functions

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

5.19.1 Detailed Description

Extract global map from meteorological data.

Definition in file met_map.c.

5.19.2 Function Documentation

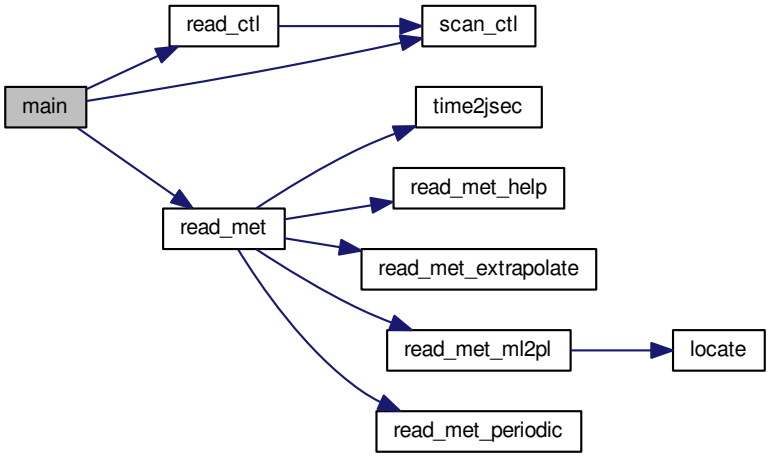
5.19.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 27 of file met_map.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *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     read_met(&ctl, argv[i], met);
00058
00059     /* Find nearest pressure level... */
00060     for (ip2 = 0; ip2 < met->np; ip2++) {
00061       dz = fabs(Z(met->p[ip2]) - z);
00062       if (dz < dzmin) {
00063         dzmin = dz;
00064         ip = ip2;
00065       }
00066     }
00067
00068     /* Average data... */
00069     for (ix = 0; ix < met->nx; ix++)
00070       for (iy = 0; iy < met->ny; iy++) {
00071         timem[ix][iy] += met->time;
00072         tm[ix][iy] += met->t[ix][iy][ip];
00073         um[ix][iy] += met->u[ix][iy][ip];
00074         vm[ix][iy] += met->v[ix][iy][ip];
00075         wm[ix][iy] += met->w[ix][iy][ip];
00076         h2om[ix][iy] += met->h2o[ix][iy][ip];
00077         o3m[ix][iy] += met->o3[ix][iy][ip];
00078         psm[ix][iy] += met->ps[ix][iy];
00079         np[ix][iy]++;
00080       }
00081   }
00082
00083   /* Create output file... */
00084   printf("Write meteorological data file: %s\n", argv[2]);
00085   if (!(out = fopen(argv[2], "w")))
00086     ERRMSG("Cannot create file!");
00087
00088   /* Write header... */
00089   fprintf(out,
00090           "# $1  = time [s]\n"
00091           "# $2  = altitude [km]\n"
00092           "# $3  = longitude [deg]\n"
00093           "# $4  = latitude [deg]\n"
```

```
00094           "# $5  = pressure [hPa]\n"
00095           "# $6  = temperature [K]\n"
00096           "# $7  = zonal wind [m/s]\n"
00097           "# $8  = meridional wind [m/s]\n"
00098           "# $9  = vertical wind [hPa/s]\n"
00099           "# $10 = H2O volume mixing ratio [1]\n"
00100           "# $11 = O3 volume mixing ratio [1]\n"
00101           "# $12 = surface pressure [hPa]\n");
00102
00103   /* Write data... */
00104   for (iy = 0; iy < met->ny; iy++) {
00105     fprintf(out, "\n");
00106     for (ix = 0; ix < met->nx; ix++)
00107       if (met->lon[ix] >= 180)
00108         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00109                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00110                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00111                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00112                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00113                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00114                 psm[ix][iy] / np[ix][iy]);
00115     for (ix = 0; ix < met->nx; ix++)
00116       if (met->lon[ix] <= 180)
00117         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00118                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00119                 met->lon[ix], met->lat[iy], met->p[ip],
00120                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00121                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00122                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00123                 psm[ix][iy] / np[ix][iy]);
00124   }
00125
00126   /* Close file... */
00127   fclose(out);
00128
00129   /* Free... */
00130   free(met);
00131
00132   return EXIT_SUCCESS;
00133 }
```

Here is the call graph for this function:



5.20 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 *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     read_met(&ctl, argv[i], met);
00058
00059     /* Find nearest pressure level... */
00060     for (ip2 = 0; ip2 < met->np; ip2++) {
00061       dz = fabs(Z(met->p[ip2]) - z);
00062       if (dz < dzmin) {
00063         dzmin = dz;
00064         ip = ip2;
00065       }
00066     }
00067
00068     /* Average data... */
00069     for (ix = 0; ix < met->nx; ix++)
00070       for (iy = 0; iy < met->ny; iy++) {
00071         timem[ix][iy] += met->time;
00072         tm[ix][iy] += met->t[ix][iy][ip];
00073         um[ix][iy] += met->u[ix][iy][ip];
00074         vm[ix][iy] += met->v[ix][iy][ip];
00075         wm[ix][iy] += met->w[ix][iy][ip];
00076         h2om[ix][iy] += met->h2o[ix][iy][ip];
00077         o3m[ix][iy] += met->o3[ix][iy][ip];
00078         psm[ix][iy] += met->ps[ix][iy];
00079         np[ix][iy]++;
00080       }
00081   }
00082
00083   /* Create output file... */
00084   printf("Write meteorological data file: %s\n", argv[2]);
00085   if (!(out = fopen(argv[2], "w")))
00086     ERRMSG("Cannot create file!");
00087
00088   /* Write header... */
00089   fprintf(out,
00090           "# $1  = time [s]\n"
00091           "# $2  = altitude [km]\n"
00092           "# $3  = longitude [deg]\n"
00093           "# $4  = latitude [deg]\n"
00094           "# $5  = pressure [hPa]\n"
00095           "# $6  = temperature [K]\n"
```

```
00096           "# $7  = zonal wind [m/s]\n"
00097           "# $8  = meridional wind [m/s]\n"
00098           "# $9  = vertical wind [hPa/s]\n"
00099           "# $10 = H2O volume mixing ratio [1]\n"
00100           "# $11 = O3 volume mixing ratio [1]\n"
00101           "# $12 = surface pressure [hPa]\n");
00102
00103   /* Write data... */
00104   for (iy = 0; iy < met->ny; iy++) {
00105     fprintf(out, "\n");
00106     for (ix = 0; ix < met->nx; ix++)
00107       if (met->lon[ix] >= 180)
00108         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00109                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00110                 met->lon[ix] - 360.0, met->lat[iy], met->p[ip],
00111                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00112                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00113                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00114                 psm[ix][iy] / np[ix][iy]);
00115     for (ix = 0; ix < met->nx; ix++)
00116       if (met->lon[ix] <= 180)
00117         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00118                 timem[ix][iy] / np[ix][iy], Z(met->p[ip]),
00119                 met->lon[ix], met->lat[iy], met->p[ip],
00120                 tm[ix][iy] / np[ix][iy], um[ix][iy] / np[ix][iy],
00121                 vm[ix][iy] / np[ix][iy], wm[ix][iy] / np[ix][iy],
00122                 h2om[ix][iy] / np[ix][iy], o3m[ix][iy] / np[ix][iy],
00123                 psm[ix][iy] / np[ix][iy]);
00124   }
00125
00126   /* Close file... */
00127   fclose(out);
00128
00129   /* Free... */
00130   free(met);
00131
00132   return EXIT_SUCCESS;
00133 }
```

5.21 met_prof.c File Reference

Extract vertical profile from meteorological data.

Functions

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

5.21.1 Detailed Description

Extract vertical profile from meteorological data.

Definition in file met_prof.c.

5.21.2 Function Documentation

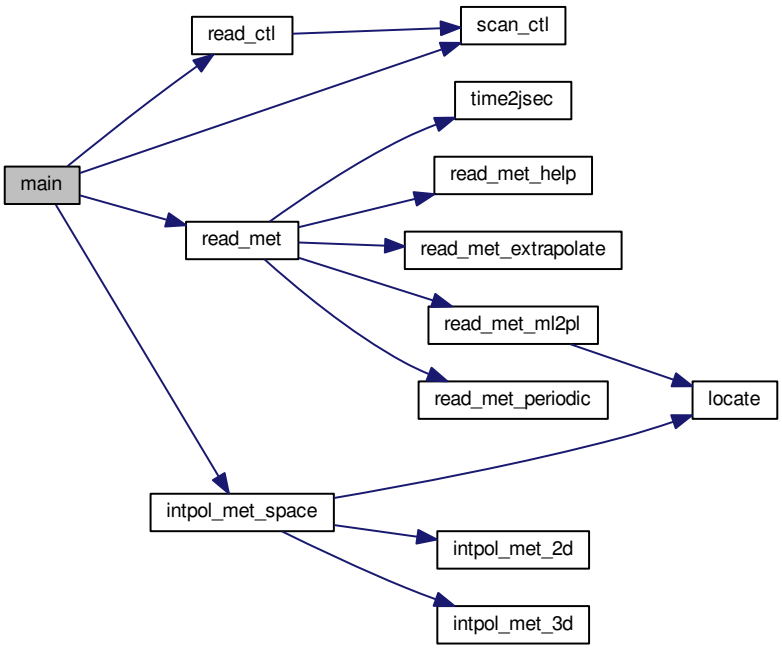
5.21.2.1 int main (int *argc*, char * *argv[]*)

Definition at line 38 of file met_prof.c.

```
00040                {
00041
00042   ctl_t ctl;
00043
00044   met_t *met;
00045
00046   FILE *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     read_met(&ctl, argv[i], met);
00078
00079     /* Average... */
00080     for (z = z0; z <= z1; z += dz) {
00081       iz = (int) ((z - z0) / dz);
00082       if (iz < 0 || iz > NZ)
00083         ERRMSG("Too many altitudes!");
00084       for (lon = lon0; lon <= lon1; lon += dlon)
00085         for (lat = lat0; lat <= lat1; lat += dlat) {
00086           intpol_met_space(met, P(z), lon, lat, &ps,
00087                            &t, &u, &v, &w, &h2o, &o3);
00088           if (gsl_finite(t) && gsl_finite(u)
00089               && gsl_finite(v) && gsl_finite(w)) {
00090             timem[iz] += met->time;
00091             lonm[iz] += lon;
00092             latm[iz] += lat;
00093             tm[iz] += t;
00094             um[iz] += u;
00095             vm[iz] += v;
00096             wm[iz] += w;
00097             h2om[iz] += h2o;
00098             o3m[iz] += o3;
00099             psm[iz] += ps;
00100             np[iz]++;
00101           }
00102         }
00103     }
00104   }
00105
00106   /* Normalize... */
00107   for (z = z0; z <= z1; z += dz) {
00108     iz = (int) ((z - z0) / dz);
00109     if (np[iz] > 0) {
00110       timem[iz] /= np[iz];
00111       lonm[iz] /= np[iz];
00112       latm[iz] /= np[iz];
00113       tm[iz] /= np[iz];
00114       um[iz] /= np[iz];
00115       vm[iz] /= np[iz];
00116       wm[iz] /= np[iz];
00117       h2om[iz] /= np[iz];
00118       o3m[iz] /= np[iz];
00119       psm[iz] /= np[iz];
00120     } else {
00121       timem[iz] = GSL_NAN;
00122       lonm[iz] = GSL_NAN;
00123       latm[iz] = GSL_NAN;
00124       tm[iz] = GSL_NAN;
00125       um[iz] = GSL_NAN;
00126       vm[iz] = GSL_NAN;
```

```
00127       wm[iz] = GSL_NAN;
00128       h2om[iz] = GSL_NAN;
00129       o3m[iz] = GSL_NAN;
00130       psm[iz] = GSL_NAN;
00131     }
00132   }
00133
00134   /* Create output file... */
00135   printf("Write meteorological data file: %s\n", argv[2]);
00136   if (!(out = fopen(argv[2], "w")))
00137     ERRMSG("Cannot create file!");
00138
00139   /* Write header... */
00140   fprintf(out,
00141           "# $1  = time [s]\n"
00142           "# $2  = altitude [km]\n"
00143           "# $3  = longitude [deg]\n"
00144           "# $4  = latitude [deg]\n"
00145           "# $5  = pressure [hPa]\n"
00146           "# $6  = temperature [K]\n"
00147           "# $7  = zonal wind [m/s]\n"
00148           "# $8  = meridional wind [m/s]\n"
00149           "# $9  = vertical wind [hPa/s]\n"
00150           "# $10 = H2O volume mixing ratio [1]\n"
00151           "# $11 = O3 volume mixing ratio [1]\n"
00152           "# $12 = surface pressure [hPa]\n\n");
00153
00154   /* Write data... */
00155   for (z = z0; z <= z1; z += dz) {
00156     iz = (int) ((z - z0) / dz);
00157     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g\n",
00158             timem[iz], z, lonm[iz], latm[iz], P(z), tm[iz],
00159             um[iz], vm[iz], wm[iz], h2om[iz], o3m[iz], psm[iz]);
00160   }
00161
00162   /* Close file... */
00163   fclose(out);
00164
00165   /* Free... */
00166   free(met);
00167
00168   return EXIT_SUCCESS;
00169 }
```

Here is the call graph for this function:



5.22 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 *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     read_met(&ctl, argv[i], met);
00078
00079     /* Average... */
00080     for (z = z0; z <= z1; z += dz) {
00081       iz = (int) ((z - z0) / dz);
00082       if (iz < 0 || iz > NZ)
00083         ERRMSG("Too many altitudes!");
00084       for (lon = lon0; lon <= lon1; lon += dlon)
00085         for (lat = lat0; lat <= lat1; lat += dlat) {
00086           intpol_met_space(met, P(z), lon, lat, &ps,
00087                            &t, &u, &v, &w, &h2o, &o3);
00088           if (gsl_finite(t) && gsl_finite(u)
00089               && gsl_finite(v) && gsl_finite(w)) {
```

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

5.23 met_sample.c File Reference

Sample meteorological data at given geolocations.

Functions

- int main (int argc, char ∗argv[ ])

5.23.1 Detailed Description

Sample meteorological data at given geolocations.

Definition in file met_sample.c.

5.23.2 Function Documentation

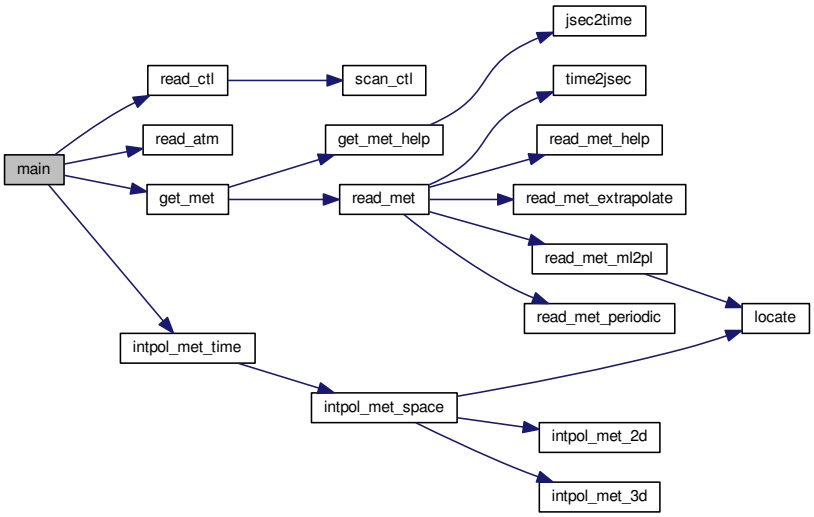
5.23.2.1 int main (int *argc*, char ∗ *argv[ ]*)

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\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:



5.24 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   Copright (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 }
```

5.25 met_zm.c File Reference

Extract zonal mean from meteorological data.

Functions

- int main (int argc, char ∗argv[ ])

5.25.1 Detailed Description

Extract zonal mean from meteorological data.

Definition in file met_zm.c.

5.25.2 Function Documentation

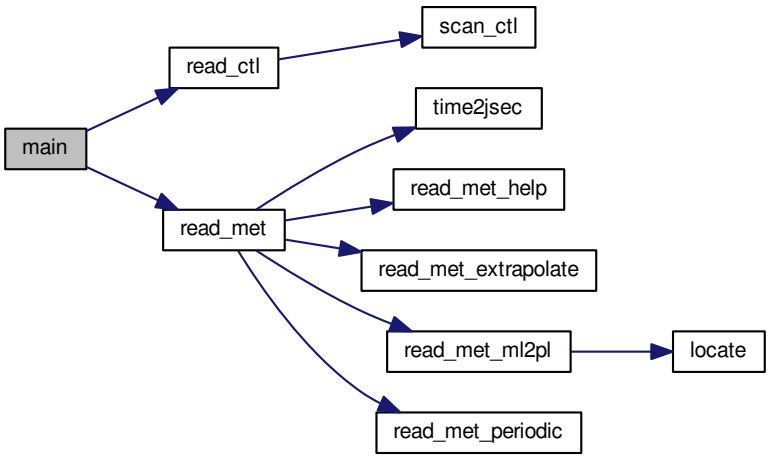
5.25.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 27 of file met_zm.c.

```
00029                {
00030
00031   ctl_t ctl;
00032
00033   met_t *met;
00034
00035   FILE *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     read_met(&ctl, argv[i], met);
00059
00060     /* Average data... */
00061     for (ix = 0; ix < met->nx; ix++)
00062       for (iy = 0; iy < met->ny; iy++)
00063         for (ip = 0; ip < met->np; ip++) {
00064           timem[ip][iy] += met->time;
00065           tm[ip][iy] += met->t[ix][iy][ip];
00066           um[ip][iy] += met->u[ix][iy][ip];
00067           vm[ip][iy] += met->v[ix][iy][ip];
00068           vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00069                               + gsl_pow_2(met->v[ix][iy][ip]));
00070           wm[ip][iy] += met->w[ix][iy][ip];
00071           h2om[ip][iy] += met->h2o[ix][iy][ip];
00072           o3m[ip][iy] += met->o3[ix][iy][ip];
00073           psm[ip][iy] += met->ps[ix][iy];
00074           tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00075           um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00076           vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
00077           vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00078             + gsl_pow_2(met->v[ix][iy][ip]);
00079           wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00080           h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00081           o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
00082           psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00083           np[ip][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  = latitude [deg]\n"
00097           "# $4  = temperature mean [K]\n"
00098           "# $5  = temperature standard deviation [K]\n"
00099           "# $6  = zonal wind mean [m/s]\n"
00100           "# $7  = zonal wind standard deviation [m/s]\n"
00101           "# $8  = meridional wind mean [m/s]\n"
00102           "# $9  = meridional wind standard deviation [m/s]\n"
00103           "# $10 = horizontal wind mean [m/s]\n"
00104           "# $11 = horizontal wind standard deviation [m/s]\n"
00105           "# $12 = vertical wind mean [hPa/s]\n"
00106           "# $13 = vertical wind standard deviation [hPa/s]\n"
00107           "# $14 = H2O vmr mean [1]\n"
00108           "# $15 = H2O vmr standard deviation [1]\n"
00109           "# $16 = O3 vmr mean [1]\n"
00110           "# $17 = O3 vmr standard deviation [1]\n"
00111           "# $18 = surface pressure mean [hPa]\n"
00112           "# $19 = surface pressure standard deviation [hPa]\n");
00113
00114   /* Write data... */
00115   for (iy = 0; iy < met->ny; iy++) {
00116     fprintf(out, "\n");
00117     for (ip = 0; ip < met->np; ip++)
00118       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g"
00119               " %g %g %g %g %g %g %g %g\n",
00120               timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
00121               tm[ip][iy] / np[ip][iy],
00122               sqrt(tm2[ip][iy] / np[ip][iy] -
00123                    gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00124               um[ip][iy] / np[ip][iy],
00125               sqrt(um2[ip][iy] / np[ip][iy] -
00126                    gsl_pow_2(um[ip][iy] / np[ip][iy])),
00127               vm[ip][iy] / np[ip][iy],
00128               sqrt(vm2[ip][iy] / np[ip][iy] -
00129                    gsl_pow_2(vm[ip][iy] / np[ip][iy])),
00130               vhm[ip][iy] / np[ip][iy],
00131               sqrt(vhm2[ip][iy] / np[ip][iy] -
00132                    gsl_pow_2(vhm[ip][iy] / np[ip][iy])),
00133               wm[ip][iy] / np[ip][iy],
00134               sqrt(wm2[ip][iy] / np[ip][iy] -
00135                    gsl_pow_2(wm[ip][iy] / np[ip][iy])),
00136               h2om[ip][iy] / np[ip][iy],
00137               sqrt(h2om2[ip][iy] / np[ip][iy] -
00138                    gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
00139               o3m[ip][iy] / np[ip][iy],
00140               sqrt(o3m2[ip][iy] / np[ip][iy] -
00141                    gsl_pow_2(o3m[ip][iy] / np[ip][iy])),
00142               psm[ip][iy] / np[ip][iy],
00143               sqrt(psm2[ip][iy] / np[ip][iy] -
00144                    gsl_pow_2(psm[ip][iy] / np[ip][iy])));
00145   }
00146
00147   /* Close file... */
00148   fclose(out);
00149
00150   /* Free... */
00151   free(met);
00152
00153   return EXIT_SUCCESS;
00154 }
```

Here is the call graph for this function:



5.26 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 *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     read_met(&ctl, argv[i], met);
00059
00060     /* Average data... */
00061     for (ix = 0; ix < met->nx; ix++)
00062       for (iy = 0; iy < met->ny; iy++)
00063         for (ip = 0; ip < met->np; ip++) {
00064           timem[ip][iy] += met->time;
00065           tm[ip][iy] += met->t[ix][iy][ip];
00066           um[ip][iy] += met->u[ix][iy][ip];
00067           vm[ip][iy] += met->v[ix][iy][ip];
00068           vhm[ip][iy] += sqrt(gsl_pow_2(met->u[ix][iy][ip])
00069                               + gsl_pow_2(met->v[ix][iy][ip]));
00070           wm[ip][iy] += met->w[ix][iy][ip];
00071           h2om[ip][iy] += met->h2o[ix][iy][ip];
00072           o3m[ip][iy] += met->o3[ix][iy][ip];
00073           psm[ip][iy] += met->ps[ix][iy];
00074           tm2[ip][iy] += gsl_pow_2(met->t[ix][iy][ip]);
00075           um2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip]);
00076           vm2[ip][iy] += gsl_pow_2(met->v[ix][iy][ip]);
00077           vhm2[ip][iy] += gsl_pow_2(met->u[ix][iy][ip])
00078             + gsl_pow_2(met->v[ix][iy][ip]);
00079           wm2[ip][iy] += gsl_pow_2(met->w[ix][iy][ip]);
00080           h2om2[ip][iy] += gsl_pow_2(met->h2o[ix][iy][ip]);
00081           o3m2[ip][iy] += gsl_pow_2(met->o3[ix][iy][ip]);
00082           psm2[ip][iy] += gsl_pow_2(met->ps[ix][iy]);
00083           np[ip][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  = latitude [deg]\n"
00097           "# $4  = temperature mean [K]\n"
00098           "# $5  = temperature standard deviation [K]\n"
00099           "# $6  = zonal wind mean [m/s]\n"
00100          "# $7  = zonal wind standard deviation [m/s]\n"
00101          "# $8  = meridional wind mean [m/s]\n"
00102          "# $9  = meridional wind standard deviation [m/s]\n"
00103          "# $10 = horizontal wind mean [m/s]\n"
00104          "# $11 = horizontal wind standard deviation [m/s]\n"
00105          "# $12 = vertical wind mean [hPa/s]\n"
00106          "# $13 = vertical wind standard deviation [hPa/s]\n"
00107          "# $14 = H2O vmr mean [1]\n"
00108          "# $15 = H2O vmr standard deviation [1]\n"
00109          "# $16 = O3 vmr mean [1]\n"
00110          "# $17 = O3 vmr standard deviation [1]\n"
00111          "# $18 = surface pressure mean [hPa]\n"
00112          "# $19 = surface pressure standard deviation [hPa]\n");
00113
00114   /* Write data... */
00115   for (iy = 0; iy < met->ny; iy++) {
00116     fprintf(out, "\n");
00117     for (ip = 0; ip < met->np; ip++)
00118       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g"
00119               " %g %g %g %g %g %g %g %g\n",
00120               timem[ip][iy] / np[ip][iy], Z(met->p[ip]), met->lat[iy],
00121               tm[ip][iy] / np[ip][iy],
00122               sqrt(tm2[ip][iy] / np[ip][iy] -
00123                    gsl_pow_2(tm[ip][iy] / np[ip][iy])),
00124               um[ip][iy] / np[ip][iy],
00125               sqrt(um2[ip][iy] / np[ip][iy] -
00126                    gsl_pow_2(um[ip][iy] / np[ip][iy])),
00127               vm[ip][iy] / np[ip][iy],
00128               sqrt(vm2[ip][iy] / np[ip][iy] -
00129                    gsl_pow_2(vm[ip][iy] / np[ip][iy])),
00130               vhm[ip][iy] / np[ip][iy],
00131               sqrt(vhm2[ip][iy] / np[ip][iy] -
00132                    gsl_pow_2(vhm[ip][iy] / np[ip][iy])),
00133               wm[ip][iy] / np[ip][iy],
00134               sqrt(wm2[ip][iy] / np[ip][iy] -
00135                    gsl_pow_2(wm[ip][iy] / np[ip][iy])),
00136               h2om[ip][iy] / np[ip][iy],
00137               sqrt(h2om2[ip][iy] / np[ip][iy] -
00138                    gsl_pow_2(h2om[ip][iy] / np[ip][iy])),
00139               o3m[ip][iy] / np[ip][iy],
```

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

5.27 smago.c File Reference

Estimate horizontal diffusivity based on Smagorinsky theory.

Functions

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

5.27.1 Detailed Description

Estimate horizontal diffusivity based on Smagorinsky theory.

Definition in file smago.c.

5.27.2 Function Documentation

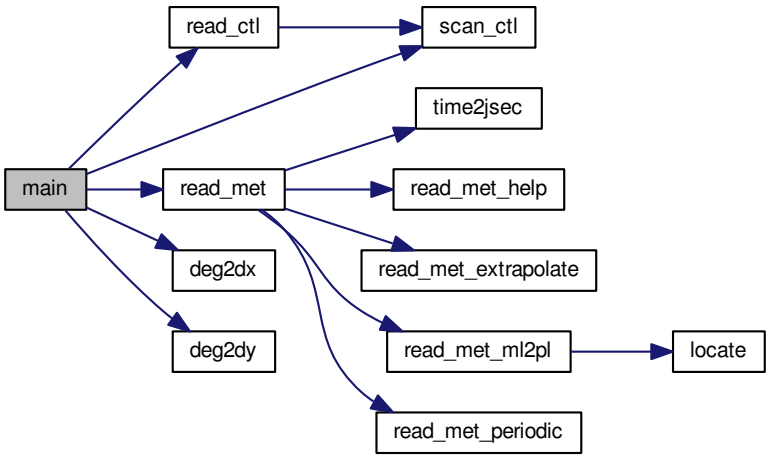
5.27.2.1 int main (int *argc*, char * *argv[]*)

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:



5.28 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 }
```

5.29 split.c File Reference

Split air parcels into a larger number of parcels.

Functions

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

5.29.1 Detailed Description

Split air parcels into a larger number of parcels.

Definition in file split.c.

5.29.2 Function Documentation

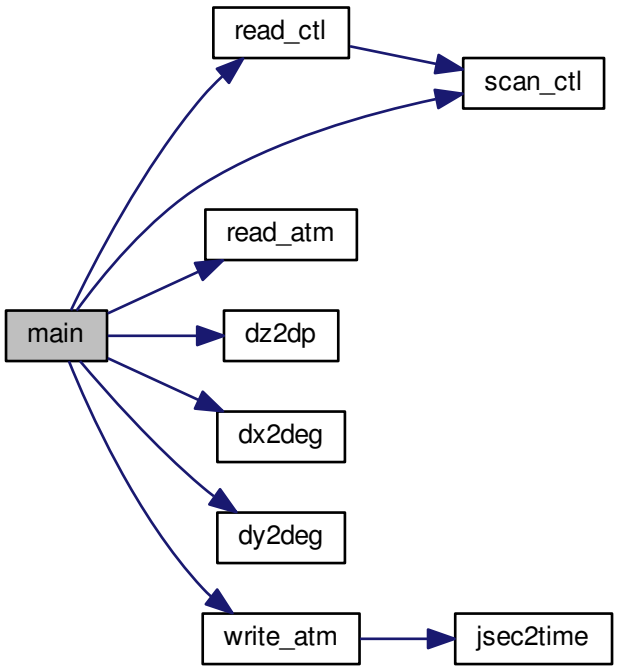
5.29.2.1 int main (int *argc*, char * *argv[]*)

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:



5.30 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 }
```

5.31 time2jsec.c File Reference

Convert date to Julian seconds.

Functions

- int main (int argc, char ∗argv[ ])

5.31.1 Detailed Description

Convert date to Julian seconds.

Definition in file time2jsec.c.

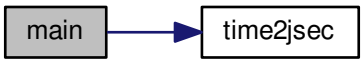
5.31.2 Function Documentation

5.31.2.1 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:



5.32 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   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 < 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 }
```

5.33 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.
- double module_meteo_hno3 (double t, double lat, double p)
  Auxiliary function for meteo module.
- 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[])

5.33.1 Detailed Description

Lagrangian particle dispersion model.

Definition in file trac.c.

5.33.2 Function Documentation

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

Set simulation time interval.

Definition at line 371 of file trac.c.

```
00373                {
00374
00375   /* Set inital and final time... */
00376   if (ctl->direction == 1) {
00377     if (ctl->t_start < -1e99)
00378       ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00379     if (ctl->t_stop < -1e99)
00380       ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00381   } else if (ctl->direction == -1) {
00382     if (ctl->t_stop < -1e99)
00383       ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00384     if (ctl->t_start < -1e99)
00385       ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00386   }
00387
00388   /* Check time... */
00389   if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
00390     ERRMSG("Nothing to do!");
00391 }
```

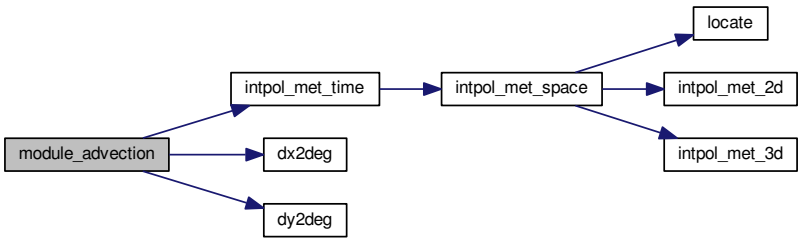
5.33.2.2 void module_advection (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate advection of air parcels.

Definition at line 395 of file trac.c.

```
00400              {
00401
00402   double v[3], xm[3];
00403
00404   /* Interpolate meteorological data... */
00405   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00406                   atm->lon[ip], atm->lat[ip], NULL, NULL,
00407                   &v[0], &v[1], &v[2], NULL, NULL);
00408
00409   /* Get position of the mid point... */
00410   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00411   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00412   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00413
00414   /* Interpolate meteorological data for mid point... */
00415   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00416                   xm[2], xm[0], xm[1], NULL, NULL,
00417                   &v[0], &v[1], &v[2], NULL, NULL);
00418
00419   /* Save new position... */
00420   atm->time[ip] += dt;
00421   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00422   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00423   atm->p[ip] += dt * v[2];
00424 }
```

Here is the call graph for this function:



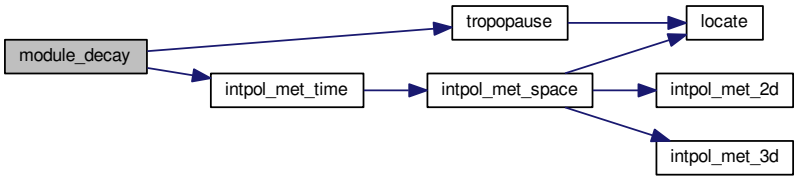
5.33.2.3 void module_decay (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate exponential decay of particle mass.

Definition at line 428 of file trac.c.

```
00434             {
00435
00436   double ps, pt, tdec;
00437
00438   /* Check lifetime values... */
00439   if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
      qnt_m < 0)
00440     return;
00441
00442   /* Set constant lifetime... */
00443   if (ctl->tdec_trop == ctl->tdec_strat)
00444     tdec = ctl->tdec_trop;
00445
00446   /* Set altitude-dependent lifetime... */
00447   else {
00448
00449     /* Get surface pressure... */
00450     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00451                     atm->lon[ip], atm->lat[ip], &ps, NULL,
00452                     NULL, NULL, NULL, NULL, NULL);
00453
00454     /* Get tropopause pressure... */
00455     pt = tropopause(atm->time[ip], atm->lat[ip]);
00456
00457     /* Set lifetime... */
00458     if (atm->p[ip] <= pt)
00459       tdec = ctl->tdec_strat;
00460     else
00461       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00462   }
00463
00464   /* Calculate exponential decay... */
00465   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00466 }
```

Here is the call graph for this function:



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

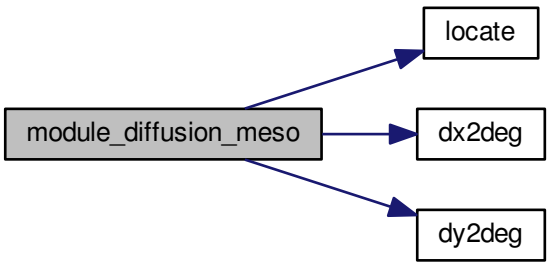
Calculate mesoscale diffusion.

Definition at line 470 of file trac.c.

```
00477                 {
00478
00479   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00480
00481   int ix, iy, iz;
00482
00483   /* Calculate mesoscale velocity fluctuations... */
00484   if (ctl->turb_meso > 0) {
00485
00486     /* Get indices... */
00487     ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00488     iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00489     iz = locate(met0->p, met0->np, atm->p[ip]);
00490
00491     /* Collect local wind data... */
00492     u[0] = met0->u[ix][iy][iz];
00493     u[1] = met0->u[ix + 1][iy][iz];
00494     u[2] = met0->u[ix][iy + 1][iz];
00495     u[3] = met0->u[ix + 1][iy + 1][iz];
00496     u[4] = met0->u[ix][iy][iz + 1];
00497     u[5] = met0->u[ix + 1][iy][iz + 1];
00498     u[6] = met0->u[ix][iy + 1][iz + 1];
00499     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00500
00501     v[0] = met0->v[ix][iy][iz];
00502     v[1] = met0->v[ix + 1][iy][iz];
00503     v[2] = met0->v[ix][iy + 1][iz];
00504     v[3] = met0->v[ix + 1][iy + 1][iz];
00505     v[4] = met0->v[ix][iy][iz + 1];
00506     v[5] = met0->v[ix + 1][iy][iz + 1];
00507     v[6] = met0->v[ix][iy + 1][iz + 1];
00508     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00509
00510     w[0] = met0->w[ix][iy][iz];
00511     w[1] = met0->w[ix + 1][iy][iz];
00512     w[2] = met0->w[ix][iy + 1][iz];
00513     w[3] = met0->w[ix + 1][iy + 1][iz];
00514     w[4] = met0->w[ix][iy][iz + 1];
00515     w[5] = met0->w[ix + 1][iy][iz + 1];
00516     w[6] = met0->w[ix][iy + 1][iz + 1];
00517     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00518
00519     /* Get indices... */
00520     ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00521     iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00522     iz = locate(met1->p, met1->np, atm->p[ip]);
00523
00524     /* Collect local wind data... */
00525     u[8] = met1->u[ix][iy][iz];
00526     u[9] = met1->u[ix + 1][iy][iz];
00527     u[10] = met1->u[ix][iy + 1][iz];
00528     u[11] = met1->u[ix + 1][iy + 1][iz];
00529     u[12] = met1->u[ix][iy][iz + 1];
00530     u[13] = met1->u[ix + 1][iy][iz + 1];
00531     u[14] = met1->u[ix][iy + 1][iz + 1];
00532     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00533
00534     v[8] = met1->v[ix][iy][iz];
00535     v[9] = met1->v[ix + 1][iy][iz];
00536     v[10] = met1->v[ix][iy + 1][iz];
00537     v[11] = met1->v[ix + 1][iy + 1][iz];
00538     v[12] = met1->v[ix][iy][iz + 1];
00539     v[13] = met1->v[ix + 1][iy][iz + 1];
00540     v[14] = met1->v[ix][iy + 1][iz + 1];
00541     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00542
00543     w[8] = met1->w[ix][iy][iz];
00544     w[9] = met1->w[ix + 1][iy][iz];
00545     w[10] = met1->w[ix][iy + 1][iz];
00546     w[11] = met1->w[ix + 1][iy + 1][iz];
00547     w[12] = met1->w[ix][iy][iz + 1];
00548     w[13] = met1->w[ix + 1][iy][iz + 1];
00549     w[14] = met1->w[ix][iy + 1][iz + 1];
00550     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00551
00552     /* Get standard deviations of local wind data... */
00553     usig = gsl_stats_sd(u, 1, 16);
00554     vsig = gsl_stats_sd(v, 1, 16);
00555     wsig = gsl_stats_sd(w, 1, 16);
00556
00557     /* Set temporal correlations for mesoscale fluctuations... */
00558     r = 1 - 2 * fabs(dt) / ctl->dt_met;
00559     rs = sqrt(1 - r * r);
00560
00561     /* Calculate mesoscale wind fluctuations... */
00562     atm->up[ip] = (float)
00563       (r * atm->up[ip]
```

```
00564         + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * usig));
00565       atm->vp[ip] = (float)
00566         (r * atm->vp[ip]
00567          + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * vsig));
00568       atm->wp[ip] = (float)
00569         (r * atm->wp[ip]
00570          + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * wsig));
00571
00572       /* Calculate air parcel displacement... */
00573       atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00574       atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00575       atm->p[ip] += atm->wp[ip] * dt;
00576     }
00577 }
```

Here is the call graph for this function:



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

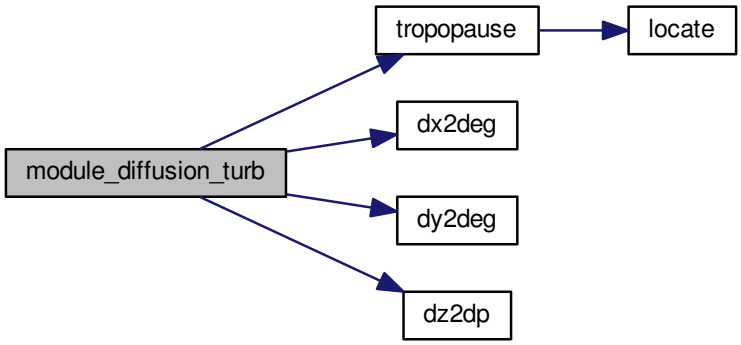
Calculate turbulent diffusion.

Definition at line 581 of file trac.c.

```
00586                 {
00587
00588   double dx, dz, pt, p0, p1, w;
00589
00590   /* Get tropopause pressure... */
00591   pt = tropopause(atm->time[ip], atm->lat[ip]);
00592
00593   /* Get weighting factor... */
00594   p1 = pt * 0.866877899;
00595   p0 = pt / 0.866877899;
00596   if (atm->p[ip] > p0)
00597     w = 1;
00598   else if (atm->p[ip] < p1)
00599     w = 0;
00600   else
00601     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00602
00603   /* Set diffusivitiy... */
00604   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00605   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00606
00607   /* Horizontal turbulent diffusion... */
00608   if (dx > 0) {
00609     atm->lon[ip]
00610       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00611                 / 1000., atm->lat[ip]);
00612     atm->lat[ip]
00613       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00614                 / 1000.);
00615   }
```

```
00616
00617   /* Vertical turbulent diffusion... */
00618   if (dz > 0)
00619     atm->p[ip]
00620       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00621                / 1000., atm->p[ip]);
00622 }
```

Here is the call graph for this function:



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

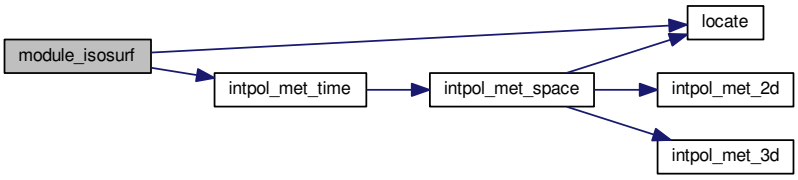
Force air parcels to stay on isosurface.

Definition at line 626 of file trac.c.

```
00631           {
00632
00633   static double *iso, *ps, t, *ts;
00634
00635   static int idx, ip2, n;
00636
00637   FILE *in;
00638
00639   char line[LEN];
00640
00641   /* Check control parameter... */
00642   if (ctl->isosurf < 1 || ctl->isosurf > 4)
00643     return;
00644
00645   /* Initialize... */
00646   if (ip < 0) {
00647
00648     /* Allocate... */
00649     ALLOC(iso, double,
00650           NP);
00651     ALLOC(ps, double,
00652           NP);
00653     ALLOC(ts, double,
00654           NP);
00655
00656     /* Save pressure... */
00657     if (ctl->isosurf == 1)
00658       for (ip2 = 0; ip2 < atm->np; ip2++)
00659         iso[ip2] = atm->p[ip2];
00660
00661     /* Save density... */
00662     else if (ctl->isosurf == 2)
00663       for (ip2 = 0; ip2 < atm->np; ip2++) {
00664         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
```

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

Here is the call graph for this function:



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

Interpolate meteorological data for air parcel positions.

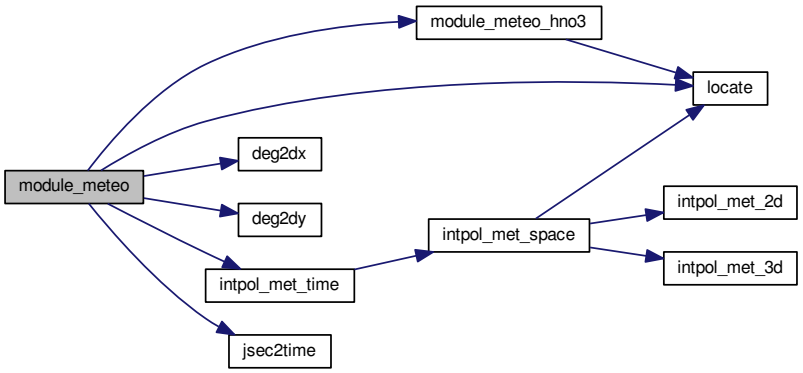
Definition at line 741 of file trac.c.

```
00746           {
00747
00748   static FILE *in;
00749
00750   static char filename[LEN], line[LEN];
00751
00752   static double lon[GX], lat[GY], var[GX][GY],
00753     rdum, rlat, rlat_old = -999, rlon, rvar;
00754
00755   static int year_old, mon_old, day_old, nlon, nlat;
00756
00757   double a, b, c, ps, p1, p_hno3, p_h2o, t, t1, u, u1, v, v1, w,
00758     x1, x2, h2o, o3, grad, vort, var0, var1;
00759
00760   int day, mon, year, idum, ilat, ilon;
00761
00762   /* Interpolate meteorological data... */
00763   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00764                   atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00765
00766   /* Set surface pressure... */
00767   if (ctl->qnt_ps >= 0)
00768     atm->q[ctl->qnt_ps][ip] = ps;
00769
00770   /* Set pressure... */
00771   if (ctl->qnt_p >= 0)
00772     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00773
00774   /* Set temperature... */
00775   if (ctl->qnt_t >= 0)
00776     atm->q[ctl->qnt_t][ip] = t;
00777
00778   /* Set zonal wind... */
00779   if (ctl->qnt_u >= 0)
00780     atm->q[ctl->qnt_u][ip] = u;
00781
00782   /* Set meridional wind... */
00783   if (ctl->qnt_v >= 0)
00784     atm->q[ctl->qnt_v][ip] = v;
00785
00786   /* Set vertical velocity... */
00787   if (ctl->qnt_w >= 0)
00788     atm->q[ctl->qnt_w][ip] = w;
00789
00790   /* Set water vapor vmr... */
00791   if (ctl->qnt_h2o >= 0)
00792     atm->q[ctl->qnt_h2o][ip] = h2o;
00793
00794   /* Set ozone vmr... */
00795   if (ctl->qnt_o3 >= 0)
00796     atm->q[ctl->qnt_o3][ip] = o3;
00797
00798   /* Calculate potential temperature... */
```

```
00799   if (ctl->qnt_theta >= 0)
00800     atm->q[ctl->qnt_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00801
00802   /* Calculate potential vorticity... */
00803   if (ctl->qnt_pv >= 0) {
00804
00805     /* Absolute vorticity... */
00806     vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00807     if (fabs(atm->lat[ip]) < 89.) {
00808       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00809                       (atm->lon[ip] >=
00810                        0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00811                       atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00812       vort += (v1 - v) / 1000.
00813         / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00814     }
00815     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00816                     (atm->lat[ip] >=
00817                      0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
00818                     &u1, NULL, NULL, NULL, NULL);
00819     vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00820
00821     /* Potential temperature gradient... */
00822     p1 = 0.85 * atm->p[ip];
00823     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
00824                     atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL);
00825     grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00826       / (100. * (p1 - atm->p[ip]));
00827
00828     /* Calculate PV... */
00829     atm->q[ctl->qnt_pv][ip] = -1e6 * G0 * vort * grad;
00830   }
00831
00832   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00833   if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00834     atm->q[ctl->qnt_tice][ip] =
00835       -2663.5 /
00836       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00837        12.537);
00838
00839   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00840   if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
00841     if (ctl->psc_hno3 > 0)
00842       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00843     else
00844       p_hno3 = module_meteo_hno3(atm->time[ip], atm->lat[ip], atm->
      p[ip])
00845         * 1e-9 * atm->p[ip] / 1.333224;
00846     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00847     a = 0.009179 - 0.00088 * log10(p_h2o);
00848     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00849     c = -11397.0 / a;
00850     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00851     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00852     if (x1 > 0)
00853       atm->q[ctl->qnt_tnat][ip] = x1;
00854     if (x2 > 0)
00855       atm->q[ctl->qnt_tnat][ip] = x2;
00856   }
00857
00858   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00859   if (ctl->qnt_tsts >= 0) {
00860     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00861       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00862     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00863                                        + atm->q[ctl->qnt_tnat][ip]);
00864   }
00865
00866   /* Read variance data for current day... */
00867   if (ip == 0 && ctl->qnt_gw_var >= 0) {
00868     jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00869     if (year != year_old || mon != mon_old || day != day_old) {
00870       year_old = year;
00871       mon_old = mon;
00872       day_old = day;
00873       nlon = nlat = -1;
00874       sprintf(filename, "%s_%d_%02d_%02d.tab",
00875               ctl->gw_basename, year, mon, day);
00876       if ((in = fopen(filename, "r"))) {
00877         printf("Read gravity wave data: %s\n", filename);
00878         while (fgets(line, LEN, in)) {
00879           if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
00880             continue;
00881           if (rlat != rlat_old) {
00882             rlat_old = rlat;
00883             if ((++nlat) > GY)
```

```
00884               ERRMSG("Too many latitudes!");
00885             nlon = -1;
00886           }
00887           if ((++nlon) > GX)
00888             ERRMSG("Too many longitudes!");
00889           lon[nlon] = rlon;
00890           lat[nlat] = rlat;
00891           var[nlon][nlat] = GSL_MAX(0, rvar);
00892         }
00893         fclose(in);
00894         nlat++;
00895         nlon++;
00896       } else
00897         printf("Warning: Missing gravity wave data: %s\n", filename);
00898     }
00899   }
00900
00901   /* Interpolate variance data... */
00902   if (ctl->qnt_gw_var >= 0) {
00903     if (nlat >= 2 && nlon >= 2) {
00904       ilat = locate(lat, nlat, atm->lat[ip]);
00905       ilon = locate(lon, nlon, atm->lon[ip]);
00906       var0 = LIN(lat[ilat], var[ilon][ilat],
00907                  lat[ilat + 1], var[ilon][ilat + 1], atm->lat[ip]);
00908       var1 = LIN(lat[ilat], var[ilon + 1][ilat],
00909                  lat[ilat + 1], var[ilon + 1][ilat + 1], atm->lat[ip]);
00910       atm->q[ctl->qnt_gw_var][ip]
00911         = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
00912     } else
00913       atm->q[ctl->qnt_gw_var][ip] = GSL_NAN;
00914   }
00915 }
```

Here is the call graph for this function:



5.33.2.8 double module_meteo_hno3 (double *t*, double *lat*, double *p*)

Auxiliary function for meteo module.

Definition at line 919 of file trac.c.

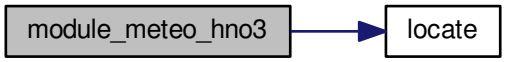
```
00922           {
00923
00924   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00925     9072000.00, 11664000.00, 14342400.00,
00926     16934400.00, 19612800.00, 22291200.00,
00927     24883200.00, 27561600.00, 30153600.00
00928   };
00929
00930   static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00931     5, 15, 25, 35, 45, 55, 65, 75, 85
```

```
00932   };
00933
00934   static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00935     31.6228, 46.4159, 68.1292, 100, 146.78
00936   };
00937
00938   static double hno3[12][18][10] = {
00939     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00940      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00941      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00942      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00943      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00944      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00945      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00946      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00947      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00948      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00949      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00950      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00951      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00952      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00953      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00954      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00955      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00956      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00957     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00958      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00959      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00960      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00961      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00962      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00963      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00964      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00965      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00966      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00967      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00968      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00969      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00970      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00971      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00972      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00973      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00974      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00975     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00976      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00977      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00978      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00979      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00980      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00981      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00982      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00983      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00984      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00985      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00986      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00987      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00988      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00989      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00990      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00991      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
00992      {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00993     {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00994      {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00995      {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00996      {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00997      {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00998      {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00999      {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
01000      {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
01001      {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
01002      {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
01003      {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
01004      {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
01005      {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
01006      {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
01007      {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
01008      {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
01009      {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
01010      {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
01011     {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
01012      {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
01013      {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
01014      {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
01015      {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
01016      {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
01017      {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
01018      {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
```

```
01019       {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
01020       {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
01021       {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
01022       {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
01023       {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
01024       {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
01025       {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
01026       {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
01027       {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
01028       {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
01029      {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
01030       {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
01031       {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
01032       {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
01033       {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
01034       {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
01035       {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
01036       {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
01037       {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
01038       {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
01039       {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
01040       {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
01041       {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
01042       {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
01043       {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
01044       {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
01045       {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
01046       {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
01047      {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
01048       {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
01049       {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
01050       {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
01051       {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
01052       {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
01053       {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
01054       {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
01055       {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
01056       {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
01057       {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
01058       {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
01059       {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
01060       {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
01061       {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
01062       {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
01063       {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
01064       {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
01065      {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
01066       {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
01067       {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
01068       {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
01069       {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
01070       {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
01071       {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
01072       {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
01073       {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
01074       {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
01075       {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
01076       {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
01077       {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
01078       {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
01079       {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
01080       {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
01081       {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
01082       {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
01083      {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
01084       {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
01085       {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
01086       {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
01087       {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
01088       {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
01089       {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
01090       {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
01091       {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
01092       {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
01093       {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
01094       {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
01095       {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
01096       {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
01097       {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
01098       {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
01099       {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
01100       {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
01101      {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
01102       {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
01103       {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
01104       {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
01105       {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
```

```
01106       {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
01107       {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
01108       {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
01109       {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
01110       {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
01111       {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
01112       {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
01113       {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
01114       {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
01115       {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
01116       {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
01117       {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
01118       {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
01119      {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
01120       {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
01121       {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
01122       {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
01123       {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
01124       {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
01125       {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
01126       {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
01127       {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
01128       {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
01129       {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
01130       {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
01131       {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
01132       {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
01133       {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
01134       {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
01135       {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
01136       {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
01137      {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
01138       {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
01139       {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
01140       {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
01141       {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
01142       {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
01143       {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
01144       {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
01145       {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
01146       {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
01147       {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
01148       {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
01149       {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
01150       {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
01151       {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
01152       {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
01153       {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
01154       {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
01155   };
01156
01157   double aux00, aux01, aux10, aux11, sec;
01158
01159   int ilat, ip, isec;
01160
01161   /* Get seconds since begin of year... */
01162   sec = fmod(t, 365.25 * 86400.);
01163
01164   /* Get indices... */
01165   ilat = locate(lats, 18, lat);
01166   ip = locate(ps, 10, p);
01167   isec = locate(secs, 12, sec);
01168
01169   /* Interpolate... */
01170   aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
01171               ps[ip + 1], hno3[isec][ilat][ip + 1], p);
01172   aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
01173               ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
01174   aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
01175               ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
01176   aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
01177               ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
01178   aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
01179   aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
01180   return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
01181 }
```

Here is the call graph for this function:



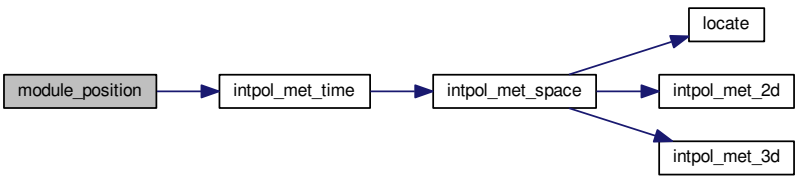
5.33.2.9 void module_position (met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*)

Check position of air parcels.

Definition at line 1185 of file trac.c.

```
01189           {
01190
01191   double ps;
01192
01193   /* Calculate modulo... */
01194   atm->lon[ip] = fmod(atm->lon[ip], 360);
01195   atm->lat[ip] = fmod(atm->lat[ip], 360);
01196
01197   /* Check latitude... */
01198   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01199     if (atm->lat[ip] > 90) {
01200       atm->lat[ip] = 180 - atm->lat[ip];
01201       atm->lon[ip] += 180;
01202     }
01203     if (atm->lat[ip] < -90) {
01204       atm->lat[ip] = -180 - atm->lat[ip];
01205       atm->lon[ip] += 180;
01206     }
01207   }
01208
01209   /* Check longitude... */
01210   while (atm->lon[ip] < -180)
01211     atm->lon[ip] += 360;
01212   while (atm->lon[ip] >= 180)
01213     atm->lon[ip] -= 360;
01214
01215   /* Get surface pressure... */
01216   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
01217                   atm->lon[ip], atm->lat[ip], &ps, NULL,
01218                   NULL, NULL, NULL, NULL, NULL);
01219
01220   /* Check pressure... */
01221   if (atm->p[ip] > ps)
01222     atm->p[ip] = ps;
01223   else if (atm->p[ip] < met0->p[met0->np - 1])
01224     atm->p[ip] = met0->p[met0->np - 1];
01225 }
```

Here is the call graph for this function:



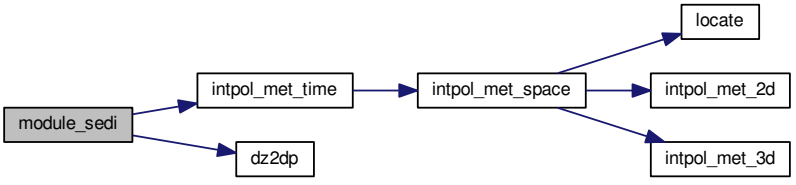
5.33.2.10 void module_sedi (ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, int *ip*, double *dt*)

Calculate sedimentation of air parcels.

Definition at line 1229 of file trac.c.

```
01235             {
01236
01237   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
01238   const double A = 1.249, B = 0.42, C = 0.87;
01239
01240   /* Specific gas constant for dry air [J/(kg K)]: */
01241   const double R = 287.058;
01242
01243   /* Average mass of an air molecule [kg/molec]: */
01244   const double m = 4.8096e-26;
01245
01246   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01247
01248   /* Check if parameters are available... */
01249   if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
01250     return;
01251
01252   /* Convert units... */
01253   p = 100 * atm->p[ip];
01254   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01255   rho_p = atm->q[ctl->qnt_rho][ip];
01256
01257   /* Get temperature... */
01258   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
01259                   atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
01260
01261   /* Density of dry air... */
01262   rho = p / (R * T);
01263
01264   /* Dynamic viscosity of air... */
01265   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01266
01267   /* Thermal velocity of an air molecule... */
01268   v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
01269
01270   /* Mean free path of an air molecule... */
01271   lambda = 2 * eta / (rho * v);
01272
01273   /* Knudsen number for air... */
01274   K = lambda / r_p;
01275
01276   /* Cunningham slip-flow correction... */
01277   G = 1 + K * (A + B * exp(-C / K));
01278
01279   /* Sedimentation (fall) velocity... */
01280   v_p =
01281     2. * gsl_pow_2(r_p) * (rho_p -
01282                            rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
01283
01284   /* Calculate pressure change... */
01285   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
01286 }
```

Here is the call graph for this function:



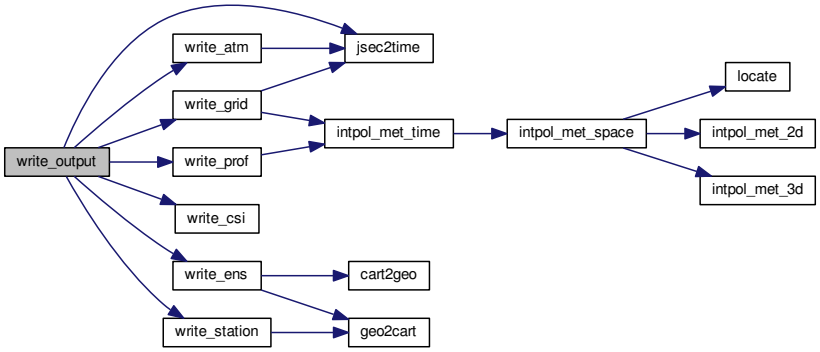
5.33.2.11 void write_output (const char * *dirname*, ctl_t * *ctl*, met_t * *met0*, met_t * *met1*, atm_t * *atm*, double *t*)

Write simulation output.

Definition at line 1290 of file trac.c.

```
01296             {
01297
01298   char filename[LEN];
01299
01300   double r;
01301
01302   int year, mon, day, hour, min, sec;
01303
01304   /* Get time... */
01305   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01306
01307   /* Write atmospheric data... */
01308   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01309     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01310             dirname, ctl->atm_basename, year, mon, day, hour, min);
01311     write_atm(filename, ctl, atm, t);
01312   }
01313
01314   /* Write CSI data... */
01315   if (ctl->csi_basename[0] != '-') {
01316     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01317     write_csi(filename, ctl, atm, t);
01318   }
01319
01320   /* Write ensemble data... */
01321   if (ctl->ens_basename[0] != '-') {
01322     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01323     write_ens(filename, ctl, atm, t);
01324   }
01325
01326   /* Write gridded data... */
01327   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01328     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01329             dirname, ctl->grid_basename, year, mon, day, hour, min);
01330     write_grid(filename, ctl, met0, met1, atm, t);
01331   }
01332
01333   /* Write profile data... */
01334   if (ctl->prof_basename[0] != '-') {
01335     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01336     write_prof(filename, ctl, met0, met1, atm, t);
01337   }
01338
01339   /* Write station data... */
01340   if (ctl->stat_basename[0] != '-') {
01341     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01342     write_station(filename, ctl, atm, t);
01343   }
01344 }
```

Here is the call graph for this function:



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

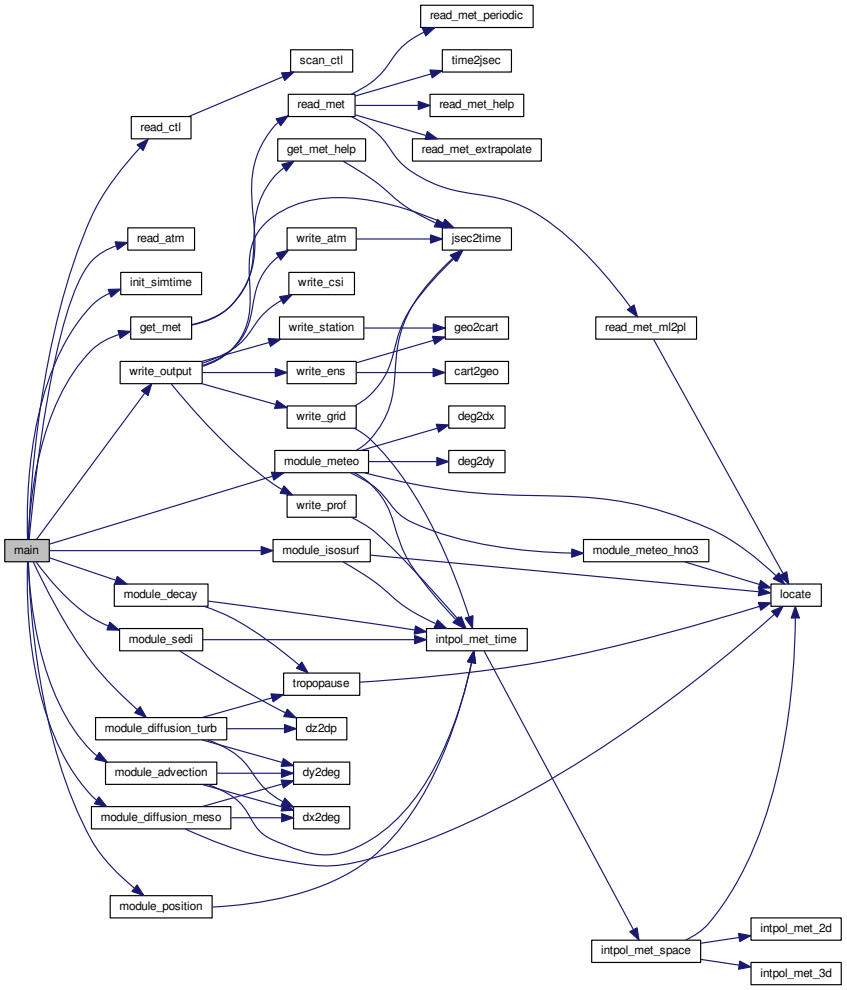
Definition at line 126 of file trac.c.

```
00128                {
00129
00130   ctl_t ctl;
00131
00132   atm_t *atm;
00133
00134   met_t *met0, *met1;
00135
00136   gsl_rng *rng[NTHREADS];
00137
00138   FILE *dirlist;
00139
00140   char dirname[LEN], filename[LEN];
00141
00142   double *dt, t, t0;
00143
00144   int i, ip, ntask = 0, rank = 0, size = 1;
00145
00146 #ifdef MPI
00147   /* Initialize MPI... */
00148   MPI_Init(&argc, &argv);
00149   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00150   MPI_Comm_size(MPI_COMM_WORLD, &size);
00151 #endif
00152
00153   /* Check arguments... */
00154   if (argc < 5)
00155     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00156
00157   /* Open directory list... */
00158   if (!(dirlist = fopen(argv[1], "r")))
00159     ERRMSG("Cannot open directory list!");
00160
00161   /* Loop over directories... */
00162   while (fscanf(dirlist, "%s", dirname) != EOF) {
00163
00164     /* MPI parallelization... */
00165     if ((++ntask) % size != rank)
00166       continue;
00167
00168     /* ------------------------------------------------------------
00169        Initialize model run...
00170        ------------------------------------------------------------ */
00171
00172     /* Set timers... */
00173     START_TIMER(TIMER_TOTAL);
00174     START_TIMER(TIMER_INIT);
00175
00176     /* Allocate... */
00177     ALLOC(atm, atm_t, 1);
00178     ALLOC(met0, met_t, 1);
00179     ALLOC(met1, met_t, 1);
00180     ALLOC(dt, double,
00181           NP);
00182
00183     /* Read control parameters... */
00184     sprintf(filename, "%s/%s", dirname, argv[2]);
00185     read_ctl(filename, argc, argv, &ctl);
00186
00187     /* Initialize random number generators... */
00188     gsl_rng_env_setup();
00189     if (omp_get_max_threads() > NTHREADS)
00190       ERRMSG("Too many threads!");
00191     for (i = 0; i < NTHREADS; i++)
00192       rng[i] = gsl_rng_alloc(gsl_rng_default);
00193
00194     /* Read atmospheric data... */
00195     sprintf(filename, "%s/%s", dirname, argv[3]);
00196     read_atm(filename, &ctl, atm);
00197
00198     /* Get simulation time interval... */
00199     init_simtime(&ctl, atm);
00200
00201     /* Get rounded start time... */
00202     if (ctl.direction == 1)
00203       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00204     else
00205       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00206
00207     /* Set timers... */
```

```
00208     STOP_TIMER(TIMER_INIT);
00209
00210     /* ------------------------------------------------------------
00211        Loop over timesteps...
00212        ------------------------------------------------------------ */
00213
00214     /* Loop over timesteps... */
00215     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
00216          t += ctl.direction * ctl.dt_mod) {
00217
00218       /* Adjust length of final time step... */
00219       if (ctl.direction * (t - ctl.t_stop) > 0)
00220         t = ctl.t_stop;
00221
00222       /* Set time steps for air parcels... */
00223       for (ip = 0; ip < atm->np; ip++)
00224         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00225              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00226              && ctl.direction * (atm->time[ip] - t) < 0))
00227           dt[ip] = t - atm->time[ip];
00228         else
00229           dt[ip] = GSL_NAN;
00230
00231       /* Get meteorological data... */
00232       START_TIMER(TIMER_INPUT);
00233       get_met(&ctl, argv[4], t, met0, met1);
00234       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00235         printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00236                fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00237       STOP_TIMER(TIMER_INPUT);
00238
00239       /* Initialize isosurface... */
00240       START_TIMER(TIMER_ISOSURF);
00241       if (t == t0)
00242         module_isosurf(&ctl, met0, met1, atm, -1);
00243       STOP_TIMER(TIMER_ISOSURF);
00244
00245       /* Advection... */
00246       START_TIMER(TIMER_ADVECT);
00247 #pragma omp parallel for default(shared) private(ip)
00248       for (ip = 0; ip < atm->np; ip++)
00249         if (gsl_finite(dt[ip]))
00250           module_advection(met0, met1, atm, ip, dt[ip]);
00251       STOP_TIMER(TIMER_ADVECT);
00252
00253       /* Turbulent diffusion... */
00254       START_TIMER(TIMER_DIFFTURB);
00255 #pragma omp parallel for default(shared) private(ip)
00256       for (ip = 0; ip < atm->np; ip++)
00257         if (gsl_finite(dt[ip]))
00258           module_diffusion_turb(&ctl, atm, ip, dt[ip],
00259                                 rng[omp_get_thread_num()]);
00260       STOP_TIMER(TIMER_DIFFTURB);
00261
00262       /* Mesoscale diffusion... */
00263       START_TIMER(TIMER_DIFFMESO);
00264 #pragma omp parallel for default(shared) private(ip)
00265       for (ip = 0; ip < atm->np; ip++)
00266         if (gsl_finite(dt[ip]))
00267           module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00268                                 rng[omp_get_thread_num()]);
00269       STOP_TIMER(TIMER_DIFFMESO);
00270
00271       /* Sedimentation... */
00272       START_TIMER(TIMER_SEDI);
00273 #pragma omp parallel for default(shared) private(ip)
00274       for (ip = 0; ip < atm->np; ip++)
00275         if (gsl_finite(dt[ip]))
00276           module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00277       STOP_TIMER(TIMER_SEDI);
00278
00279       /* Isosurface... */
00280       START_TIMER(TIMER_ISOSURF);
00281 #pragma omp parallel for default(shared) private(ip)
00282       for (ip = 0; ip < atm->np; ip++)
00283         module_isosurf(&ctl, met0, met1, atm, ip);
00284       STOP_TIMER(TIMER_ISOSURF);
00285
00286       /* Position... */
00287       START_TIMER(TIMER_POSITION);
00288 #pragma omp parallel for default(shared) private(ip)
00289       for (ip = 0; ip < atm->np; ip++)
00290         module_position(met0, met1, atm, ip);
00291       STOP_TIMER(TIMER_POSITION);
00292
00293       /* Meteorological data... */
00294       START_TIMER(TIMER_METEO);
```

```
00295       module_meteo(&ctl, met0, met1, atm, 0);
00296 #pragma omp parallel for default(shared) private(ip)
00297       for (ip = 1; ip < atm->np; ip++)
00298         module_meteo(&ctl, met0, met1, atm, ip);
00299       STOP_TIMER(TIMER_METEO);
00300
00301       /* Decay... */
00302       START_TIMER(TIMER_DECAY);
00303 #pragma omp parallel for default(shared) private(ip)
00304       for (ip = 0; ip < atm->np; ip++)
00305         if (gsl_finite(dt[ip]))
00306           module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00307       STOP_TIMER(TIMER_DECAY);
00308
00309       /* Write output... */
00310       START_TIMER(TIMER_OUTPUT);
00311       write_output(dirname, &ctl, met0, met1, atm, t);
00312       STOP_TIMER(TIMER_OUTPUT);
00313     }
00314
00315     /* ------------------------------------------------------------
00316        Finalize model run...
00317        ------------------------------------------------------------ */
00318
00319     /* Report timers... */
00320     STOP_TIMER(TIMER_TOTAL);
00321     PRINT_TIMER(TIMER_TOTAL);
00322     PRINT_TIMER(TIMER_INIT);
00323     PRINT_TIMER(TIMER_STAGE);
00324     PRINT_TIMER(TIMER_INPUT);
00325     PRINT_TIMER(TIMER_OUTPUT);
00326     PRINT_TIMER(TIMER_ADVECT);
00327     PRINT_TIMER(TIMER_DECAY);
00328     PRINT_TIMER(TIMER_DIFFMESO);
00329     PRINT_TIMER(TIMER_DIFFTURB);
00330     PRINT_TIMER(TIMER_ISOSURF);
00331     PRINT_TIMER(TIMER_METEO);
00332     PRINT_TIMER(TIMER_POSITION);
00333     PRINT_TIMER(TIMER_SEDI);
00334
00335     /* Report memory usage... */
00336     printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00337     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00338     printf("MEMORY_DYNAMIC = %g MByte\n",
00339            4 * NP * sizeof(double) / 1024. / 1024.);
00340     printf("MEMORY_STATIC = %g MByte\n",
00341            ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
00342             + (EX * EY + EX * EY * EP) * sizeof(float)
00343             + (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00344
00345     /* Report problem size... */
00346     printf("SIZE_NP = %d\n", atm->np);
00347     printf("SIZE_TASKS = %d\n", size);
00348     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00349
00350     /* Free random number generators... */
00351     for (i = 0; i < NTHREADS; i++)
00352       gsl_rng_free(rng[i]);
00353
00354     /* Free... */
00355     free(atm);
00356     free(met0);
00357     free(met1);
00358     free(dt);
00359   }
00360
00361 #ifdef MPI
00362   /* Finalize MPI... */
00363   MPI_Finalize();
00364 #endif
00365
00366   return EXIT_SUCCESS;
00367 }
```

Here is the call graph for this function:



5.34 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    Functions...
00033    ------------------------------------------------------------ */
00034
00036 void init_simtime(
00037   ctl_t * ctl,
00038   atm_t * atm);
00039
00041 void module_advection(
00042   met_t * met0,
00043   met_t * met1,
00044   atm_t * atm,
00045   int ip,
00046   double dt);
00047
00049 void module_decay(
00050   ctl_t * ctl,
00051   met_t * met0,
00052   met_t * met1,
00053   atm_t * atm,
00054   int ip,
00055   double dt);
00056
00058 void module_diffusion_meso(
00059   ctl_t * ctl,
00060   met_t * met0,
00061   met_t * met1,
00062   atm_t * atm,
00063   int ip,
00064   double dt,
00065   gsl_rng * rng);
00066
00068 void module_diffusion_turb(
00069   ctl_t * ctl,
00070   atm_t * atm,
00071   int ip,
00072   double dt,
00073   gsl_rng * rng);
00074
00076 void module_isosurf(
00077   ctl_t * ctl,
00078   met_t * met0,
00079   met_t * met1,
00080   atm_t * atm,
00081   int ip);
00082
00084 void module_meteo(
00085   ctl_t * ctl,
00086   met_t * met0,
00087   met_t * met1,
00088   atm_t * atm,
00089   int ip);
00090
00092 double module_meteo_hno3(
00093   double t,
00094   double lat,
00095   double p);
00096
00098 void module_position(
00099   met_t * met0,
00100   met_t * met1,
00101   atm_t * atm,
00102   int ip);
00103
00105 void module_sedi(
00106   ctl_t * ctl,
00107   met_t * met0,
00108   met_t * met1,
00109   atm_t * atm,
00110   int ip,
00111   double dt);
00112
00114 void write_output(
00115   const char *dirname,
00116   ctl_t * ctl,
00117   met_t * met0,
00118   met_t * met1,
00119   atm_t * atm,
00120   double t);
00121
00122 /* ------------------------------------------------------------
00123    Main...
00124    ------------------------------------------------------------ */
00125
00126 int main(
00127   int argc,
00128   char *argv[]) {
```

```
00129
00130   ctl_t ctl;
00131
00132   atm_t *atm;
00133
00134   met_t *met0, *met1;
00135
00136   gsl_rng *rng[NTHREADS];
00137
00138   FILE *dirlist;
00139
00140   char dirname[LEN], filename[LEN];
00141
00142   double *dt, t, t0;
00143
00144   int i, ip, ntask = 0, rank = 0, size = 1;
00145
00146 #ifdef MPI
00147   /* Initialize MPI... */
00148   MPI_Init(&argc, &argv);
00149   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00150   MPI_Comm_size(MPI_COMM_WORLD, &size);
00151 #endif
00152
00153   /* Check arguments... */
00154   if (argc < 5)
00155     ERRMSG("Give parameters: <dirlist> <ctl> <atm_in> <metbase>");
00156
00157   /* Open directory list... */
00158   if (!(dirlist = fopen(argv[1], "r")))
00159     ERRMSG("Cannot open directory list!");
00160
00161   /* Loop over directories... */
00162   while (fscanf(dirlist, "%s", dirname) != EOF) {
00163
00164     /* MPI parallelization... */
00165     if ((++ntask) % size != rank)
00166       continue;
00167
00168     /* ------------------------------------------------------------
00169        Initialize model run...
00170        ------------------------------------------------------------ */
00171
00172     /* Set timers... */
00173     START_TIMER(TIMER_TOTAL);
00174     START_TIMER(TIMER_INIT);
00175
00176     /* Allocate... */
00177     ALLOC(atm, atm_t, 1);
00178     ALLOC(met0, met_t, 1);
00179     ALLOC(met1, met_t, 1);
00180     ALLOC(dt, double,
00181           NP);
00182
00183     /* Read control parameters... */
00184     sprintf(filename, "%s/%s", dirname, argv[2]);
00185     read_ctl(filename, argc, argv, &ctl);
00186
00187     /* Initialize random number generators... */
00188     gsl_rng_env_setup();
00189     if (omp_get_max_threads() > NTHREADS)
00190       ERRMSG("Too many threads!");
00191     for (i = 0; i < NTHREADS; i++)
00192       rng[i] = gsl_rng_alloc(gsl_rng_default);
00193
00194     /* Read atmospheric data... */
00195     sprintf(filename, "%s/%s", dirname, argv[3]);
00196     read_atm(filename, &ctl, atm);
00197
00198     /* Get simulation time interval... */
00199     init_simtime(&ctl, atm);
00200
00201     /* Get rounded start time... */
00202     if (ctl.direction == 1)
00203       t0 = floor(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00204     else
00205       t0 = ceil(ctl.t_start / ctl.dt_mod) * ctl.dt_mod;
00206
00207     /* Set timers... */
00208     STOP_TIMER(TIMER_INIT);
00209
00210     /* ------------------------------------------------------------
00211        Loop over timesteps...
00212        ------------------------------------------------------------ */
00213
00214     /* Loop over timesteps... */
00215     for (t = t0; ctl.direction * (t - ctl.t_stop) < ctl.dt_mod;
```

```
00216          t += ctl.direction * ctl.dt_mod) {
00217
00218       /* Adjust length of final time step... */
00219       if (ctl.direction * (t - ctl.t_stop) > 0)
00220         t = ctl.t_stop;
00221
00222       /* Set time steps for air parcels... */
00223       for (ip = 0; ip < atm->np; ip++)
00224         if ((ctl.direction * (atm->time[ip] - ctl.t_start) >= 0
00225              && ctl.direction * (atm->time[ip] - ctl.t_stop) <= 0
00226              && ctl.direction * (atm->time[ip] - t) < 0))
00227           dt[ip] = t - atm->time[ip];
00228         else
00229           dt[ip] = GSL_NAN;
00230
00231       /* Get meteorological data... */
00232       START_TIMER(TIMER_INPUT);
00233       get_met(&ctl, argv[4], t, met0, met1);
00234       if (ctl.dt_mod > fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.)
00235         printf("Warning: Violation of CFL criterion! Set DT_MOD <= %g s!\n",
00236                fabs(met0->lon[1] - met0->lon[0]) * 111132. / 150.);
00237       STOP_TIMER(TIMER_INPUT);
00238
00239       /* Initialize isosurface... */
00240       START_TIMER(TIMER_ISOSURF);
00241       if (t == t0)
00242         module_isosurf(&ctl, met0, met1, atm, -1);
00243       STOP_TIMER(TIMER_ISOSURF);
00244
00245       /* Advection... */
00246       START_TIMER(TIMER_ADVECT);
00247 #pragma omp parallel for default(shared) private(ip)
00248       for (ip = 0; ip < atm->np; ip++)
00249         if (gsl_finite(dt[ip]))
00250           module_advection(met0, met1, atm, ip, dt[ip]);
00251       STOP_TIMER(TIMER_ADVECT);
00252
00253       /* Turbulent diffusion... */
00254       START_TIMER(TIMER_DIFFTURB);
00255 #pragma omp parallel for default(shared) private(ip)
00256       for (ip = 0; ip < atm->np; ip++)
00257         if (gsl_finite(dt[ip]))
00258           module_diffusion_turb(&ctl, atm, ip, dt[ip],
00259                                 rng[omp_get_thread_num()]);
00260       STOP_TIMER(TIMER_DIFFTURB);
00261
00262       /* Mesoscale diffusion... */
00263       START_TIMER(TIMER_DIFFMESO);
00264 #pragma omp parallel for default(shared) private(ip)
00265       for (ip = 0; ip < atm->np; ip++)
00266         if (gsl_finite(dt[ip]))
00267           module_diffusion_meso(&ctl, met0, met1, atm, ip, dt[ip],
00268                                 rng[omp_get_thread_num()]);
00269       STOP_TIMER(TIMER_DIFFMESO);
00270
00271       /* Sedimentation... */
00272       START_TIMER(TIMER_SEDI);
00273 #pragma omp parallel for default(shared) private(ip)
00274       for (ip = 0; ip < atm->np; ip++)
00275         if (gsl_finite(dt[ip]))
00276           module_sedi(&ctl, met0, met1, atm, ip, dt[ip]);
00277       STOP_TIMER(TIMER_SEDI);
00278
00279       /* Isosurface... */
00280       START_TIMER(TIMER_ISOSURF);
00281 #pragma omp parallel for default(shared) private(ip)
00282       for (ip = 0; ip < atm->np; ip++)
00283         module_isosurf(&ctl, met0, met1, atm, ip);
00284       STOP_TIMER(TIMER_ISOSURF);
00285
00286       /* Position... */
00287       START_TIMER(TIMER_POSITION);
00288 #pragma omp parallel for default(shared) private(ip)
00289       for (ip = 0; ip < atm->np; ip++)
00290         module_position(met0, met1, atm, ip);
00291       STOP_TIMER(TIMER_POSITION);
00292
00293       /* Meteorological data... */
00294       START_TIMER(TIMER_METEO);
00295       module_meteo(&ctl, met0, met1, atm, 0);
00296 #pragma omp parallel for default(shared) private(ip)
00297       for (ip = 1; ip < atm->np; ip++)
00298         module_meteo(&ctl, met0, met1, atm, ip);
00299       STOP_TIMER(TIMER_METEO);
00300
00301       /* Decay... */
00302       START_TIMER(TIMER_DECAY);
```

```
00303 #pragma omp parallel for default(shared) private(ip)
00304       for (ip = 0; ip < atm->np; ip++)
00305         if (gsl_finite(dt[ip]))
00306           module_decay(&ctl, met0, met1, atm, ip, dt[ip]);
00307       STOP_TIMER(TIMER_DECAY);
00308
00309       /* Write output... */
00310       START_TIMER(TIMER_OUTPUT);
00311       write_output(dirname, &ctl, met0, met1, atm, t);
00312       STOP_TIMER(TIMER_OUTPUT);
00313     }
00314
00315     /* ------------------------------------------------------------
00316        Finalize model run...
00317        ------------------------------------------------------------ */
00318
00319     /* Report timers... */
00320     STOP_TIMER(TIMER_TOTAL);
00321     PRINT_TIMER(TIMER_TOTAL);
00322     PRINT_TIMER(TIMER_INIT);
00323     PRINT_TIMER(TIMER_STAGE);
00324     PRINT_TIMER(TIMER_INPUT);
00325     PRINT_TIMER(TIMER_OUTPUT);
00326     PRINT_TIMER(TIMER_ADVECT);
00327     PRINT_TIMER(TIMER_DECAY);
00328     PRINT_TIMER(TIMER_DIFFMESO);
00329     PRINT_TIMER(TIMER_DIFFTURB);
00330     PRINT_TIMER(TIMER_ISOSURF);
00331     PRINT_TIMER(TIMER_METEO);
00332     PRINT_TIMER(TIMER_POSITION);
00333     PRINT_TIMER(TIMER_SEDI);
00334
00335     /* Report memory usage... */
00336     printf("MEMORY_ATM = %g MByte\n", sizeof(atm_t) / 1024. / 1024.);
00337     printf("MEMORY_METEO = %g MByte\n", 2. * sizeof(met_t) / 1024. / 1024.);
00338     printf("MEMORY_DYNAMIC = %g MByte\n",
00339            4 * NP * sizeof(double) / 1024. / 1024.);
00340     printf("MEMORY_STATIC = %g MByte\n",
00341            ((3 * GX * GY + 4 * GX * GY * GZ) * sizeof(double)
00342             + (EX * EY + EX * EY * EP) * sizeof(float)
00343             + (GX * GY + GX * GY * GZ) * sizeof(int)) / 1024. / 1024.);
00344
00345     /* Report problem size... */
00346     printf("SIZE_NP = %d\n", atm->np);
00347     printf("SIZE_TASKS = %d\n", size);
00348     printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00349
00350     /* Free random number generators... */
00351     for (i = 0; i < NTHREADS; i++)
00352       gsl_rng_free(rng[i]);
00353
00354     /* Free... */
00355     free(atm);
00356     free(met0);
00357     free(met1);
00358     free(dt);
00359   }
00360
00361 #ifdef MPI
00362   /* Finalize MPI... */
00363   MPI_Finalize();
00364 #endif
00365
00366   return EXIT_SUCCESS;
00367 }
00368
00369 /*****************************************************************************/
00370
00371 void init_simtime(
00372   ctl_t * ctl,
00373   atm_t * atm) {
00374
00375   /* Set inital and final time... */
00376   if (ctl->direction == 1) {
00377     if (ctl->t_start < -1e99)
00378       ctl->t_start = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00379     if (ctl->t_stop < -1e99)
00380       ctl->t_stop = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00381   } else if (ctl->direction == -1) {
00382     if (ctl->t_stop < -1e99)
00383       ctl->t_stop = gsl_stats_min(atm->time, 1, (size_t) atm->np);
00384     if (ctl->t_start < -1e99)
00385       ctl->t_start = gsl_stats_max(atm->time, 1, (size_t) atm->np);
00386   }
00387
00388   /* Check time... */
00389   if (ctl->direction * (ctl->t_stop - ctl->t_start) <= 0)
```

```
00390     ERRMSG("Nothing to do!");
00391 }
00392
00393 /*****************************************************************************/
00394
00395 void module_advection(
00396   met_t * met0,
00397   met_t * met1,
00398   atm_t * atm,
00399   int ip,
00400   double dt) {
00401
00402   double v[3], xm[3];
00403
00404   /* Interpolate meteorological data... */
00405   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00406                   atm->lon[ip], atm->lat[ip], NULL, NULL,
00407                   &v[0], &v[1], &v[2], NULL, NULL);
00408
00409   /* Get position of the mid point... */
00410   xm[0] = atm->lon[ip] + dx2deg(0.5 * dt * v[0] / 1000., atm->lat[ip]);
00411   xm[1] = atm->lat[ip] + dy2deg(0.5 * dt * v[1] / 1000.);
00412   xm[2] = atm->p[ip] + 0.5 * dt * v[2];
00413
00414   /* Interpolate meteorological data for mid point... */
00415   intpol_met_time(met0, met1, atm->time[ip] + 0.5 * dt,
00416                   xm[2], xm[0], xm[1], NULL, NULL,
00417                   &v[0], &v[1], &v[2], NULL, NULL);
00418
00419   /* Save new position... */
00420   atm->time[ip] += dt;
00421   atm->lon[ip] += dx2deg(dt * v[0] / 1000., xm[1]);
00422   atm->lat[ip] += dy2deg(dt * v[1] / 1000.);
00423   atm->p[ip] += dt * v[2];
00424 }
00425
00426 /*****************************************************************************/
00427
00428 void module_decay(
00429   ctl_t * ctl,
00430   met_t * met0,
00431   met_t * met1,
00432   atm_t * atm,
00433   int ip,
00434   double dt) {
00435
00436   double ps, pt, tdec;
00437
00438   /* Check lifetime values... */
00439   if ((ctl->tdec_trop <= 0 && ctl->tdec_strat <= 0) || ctl->
      qnt_m < 0)
00440     return;
00441
00442   /* Set constant lifetime... */
00443   if (ctl->tdec_trop == ctl->tdec_strat)
00444     tdec = ctl->tdec_trop;
00445
00446   /* Set altitude-dependent lifetime... */
00447   else {
00448
00449     /* Get surface pressure... */
00450     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00451                     atm->lon[ip], atm->lat[ip], &ps, NULL,
00452                     NULL, NULL, NULL, NULL, NULL);
00453
00454     /* Get tropopause pressure... */
00455     pt = tropopause(atm->time[ip], atm->lat[ip]);
00456
00457     /* Set lifetime... */
00458     if (atm->p[ip] <= pt)
00459       tdec = ctl->tdec_strat;
00460     else
00461       tdec = LIN(ps, ctl->tdec_trop, pt, ctl->tdec_strat, atm->
      p[ip]);
00462   }
00463
00464   /* Calculate exponential decay... */
00465   atm->q[ctl->qnt_m][ip] *= exp(-dt / tdec);
00466 }
00467
00468 /*****************************************************************************/
00469
00470 void module_diffusion_meso(
00471   ctl_t * ctl,
00472   met_t * met0,
00473   met_t * met1,
00474   atm_t * atm,
```

```
00475   int ip,
00476   double dt,
00477   gsl_rng * rng) {
00478
00479   double r, rs, u[16], v[16], w[16], usig, vsig, wsig;
00480
00481   int ix, iy, iz;
00482
00483   /* Calculate mesoscale velocity fluctuations... */
00484   if (ctl->turb_meso > 0) {
00485
00486     /* Get indices... */
00487     ix = locate(met0->lon, met0->nx, atm->lon[ip]);
00488     iy = locate(met0->lat, met0->ny, atm->lat[ip]);
00489     iz = locate(met0->p, met0->np, atm->p[ip]);
00490
00491     /* Collect local wind data... */
00492     u[0] = met0->u[ix][iy][iz];
00493     u[1] = met0->u[ix + 1][iy][iz];
00494     u[2] = met0->u[ix][iy + 1][iz];
00495     u[3] = met0->u[ix + 1][iy + 1][iz];
00496     u[4] = met0->u[ix][iy][iz + 1];
00497     u[5] = met0->u[ix + 1][iy][iz + 1];
00498     u[6] = met0->u[ix][iy + 1][iz + 1];
00499     u[7] = met0->u[ix + 1][iy + 1][iz + 1];
00500
00501     v[0] = met0->v[ix][iy][iz];
00502     v[1] = met0->v[ix + 1][iy][iz];
00503     v[2] = met0->v[ix][iy + 1][iz];
00504     v[3] = met0->v[ix + 1][iy + 1][iz];
00505     v[4] = met0->v[ix][iy][iz + 1];
00506     v[5] = met0->v[ix + 1][iy][iz + 1];
00507     v[6] = met0->v[ix][iy + 1][iz + 1];
00508     v[7] = met0->v[ix + 1][iy + 1][iz + 1];
00509
00510     w[0] = met0->w[ix][iy][iz];
00511     w[1] = met0->w[ix + 1][iy][iz];
00512     w[2] = met0->w[ix][iy + 1][iz];
00513     w[3] = met0->w[ix + 1][iy + 1][iz];
00514     w[4] = met0->w[ix][iy][iz + 1];
00515     w[5] = met0->w[ix + 1][iy][iz + 1];
00516     w[6] = met0->w[ix][iy + 1][iz + 1];
00517     w[7] = met0->w[ix + 1][iy + 1][iz + 1];
00518
00519     /* Get indices... */
00520     ix = locate(met1->lon, met1->nx, atm->lon[ip]);
00521     iy = locate(met1->lat, met1->ny, atm->lat[ip]);
00522     iz = locate(met1->p, met1->np, atm->p[ip]);
00523
00524     /* Collect local wind data... */
00525     u[8] = met1->u[ix][iy][iz];
00526     u[9] = met1->u[ix + 1][iy][iz];
00527     u[10] = met1->u[ix][iy + 1][iz];
00528     u[11] = met1->u[ix + 1][iy + 1][iz];
00529     u[12] = met1->u[ix][iy][iz + 1];
00530     u[13] = met1->u[ix + 1][iy][iz + 1];
00531     u[14] = met1->u[ix][iy + 1][iz + 1];
00532     u[15] = met1->u[ix + 1][iy + 1][iz + 1];
00533
00534     v[8] = met1->v[ix][iy][iz];
00535     v[9] = met1->v[ix + 1][iy][iz];
00536     v[10] = met1->v[ix][iy + 1][iz];
00537     v[11] = met1->v[ix + 1][iy + 1][iz];
00538     v[12] = met1->v[ix][iy][iz + 1];
00539     v[13] = met1->v[ix + 1][iy][iz + 1];
00540     v[14] = met1->v[ix][iy + 1][iz + 1];
00541     v[15] = met1->v[ix + 1][iy + 1][iz + 1];
00542
00543     w[8] = met1->w[ix][iy][iz];
00544     w[9] = met1->w[ix + 1][iy][iz];
00545     w[10] = met1->w[ix][iy + 1][iz];
00546     w[11] = met1->w[ix + 1][iy + 1][iz];
00547     w[12] = met1->w[ix][iy][iz + 1];
00548     w[13] = met1->w[ix + 1][iy][iz + 1];
00549     w[14] = met1->w[ix][iy + 1][iz + 1];
00550     w[15] = met1->w[ix + 1][iy + 1][iz + 1];
00551
00552     /* Get standard deviations of local wind data... */
00553     usig = gsl_stats_sd(u, 1, 16);
00554     vsig = gsl_stats_sd(v, 1, 16);
00555     wsig = gsl_stats_sd(w, 1, 16);
00556
00557     /* Set temporal correlations for mesoscale fluctuations... */
00558     r = 1 - 2 * fabs(dt) / ctl->dt_met;
00559     rs = sqrt(1 - r * r);
00560
00561     /* Calculate mesoscale wind fluctuations... */
```

```
00562     atm->up[ip] = (float)
00563       (r * atm->up[ip]
00564        + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * usig));
00565     atm->vp[ip] = (float)
00566       (r * atm->vp[ip]
00567        + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * vsig));
00568     atm->wp[ip] = (float)
00569       (r * atm->wp[ip]
00570        + rs * gsl_ran_gaussian_ziggurat(rng, ctl->turb_meso * wsig));
00571
00572     /* Calculate air parcel displacement... */
00573     atm->lon[ip] += dx2deg(atm->up[ip] * dt / 1000., atm->lat[ip]);
00574     atm->lat[ip] += dy2deg(atm->vp[ip] * dt / 1000.);
00575     atm->p[ip] += atm->wp[ip] * dt;
00576   }
00577 }
00578
00579 /*****************************************************************************/
00580
00581 void module_diffusion_turb(
00582   ctl_t * ctl,
00583   atm_t * atm,
00584   int ip,
00585   double dt,
00586   gsl_rng * rng) {
00587
00588   double dx, dz, pt, p0, p1, w;
00589
00590   /* Get tropopause pressure... */
00591   pt = tropopause(atm->time[ip], atm->lat[ip]);
00592
00593   /* Get weighting factor... */
00594   p1 = pt * 0.866877899;
00595   p0 = pt / 0.866877899;
00596   if (atm->p[ip] > p0)
00597     w = 1;
00598   else if (atm->p[ip] < p1)
00599     w = 0;
00600   else
00601     w = LIN(p0, 1.0, p1, 0.0, atm->p[ip]);
00602
00603   /* Set diffusivitiy... */
00604   dx = w * ctl->turb_dx_trop + (1 - w) * ctl->turb_dx_strat;
00605   dz = w * ctl->turb_dz_trop + (1 - w) * ctl->turb_dz_strat;
00606
00607   /* Horizontal turbulent diffusion... */
00608   if (dx > 0) {
00609     atm->lon[ip]
00610       += dx2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00611                 / 1000., atm->lat[ip]);
00612     atm->lat[ip]
00613       += dy2deg(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dx * fabs(dt)))
00614                 / 1000.);
00615   }
00616
00617   /* Vertical turbulent diffusion... */
00618   if (dz > 0)
00619     atm->p[ip]
00620       += dz2dp(gsl_ran_gaussian_ziggurat(rng, sqrt(2.0 * dz * fabs(dt)))
00621                / 1000., atm->p[ip]);
00622 }
00623
00624 /*****************************************************************************/
00625
00626 void module_isosurf(
00627   ctl_t * ctl,
00628   met_t * met0,
00629   met_t * met1,
00630   atm_t * atm,
00631   int ip) {
00632
00633   static double *iso, *ps, t, *ts;
00634
00635   static int idx, ip2, n;
00636
00637   FILE *in;
00638
00639   char line[LEN];
00640
00641   /* Check control parameter... */
00642   if (ctl->isosurf < 1 || ctl->isosurf > 4)
00643     return;
00644
00645   /* Initialize... */
00646   if (ip < 0) {
00647
00648     /* Allocate... */
```

```
00649     ALLOC(iso, double,
00650           NP);
00651     ALLOC(ps, double,
00652           NP);
00653     ALLOC(ts, double,
00654           NP);
00655
00656     /* Save pressure... */
00657     if (ctl->isosurf == 1)
00658       for (ip2 = 0; ip2 < atm->np; ip2++)
00659         iso[ip2] = atm->p[ip2];
00660
00661     /* Save density... */
00662     else if (ctl->isosurf == 2)
00663       for (ip2 = 0; ip2 < atm->np; ip2++) {
00664         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00665                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00666                         NULL, NULL, NULL);
00667         iso[ip2] = atm->p[ip2] / t;
00668       }
00669
00670     /* Save potential temperature... */
00671     else if (ctl->isosurf == 3)
00672       for (ip2 = 0; ip2 < atm->np; ip2++) {
00673         intpol_met_time(met0, met1, atm->time[ip2], atm->p[ip2],
00674                         atm->lon[ip2], atm->lat[ip2], NULL, &t, NULL, NULL,
00675                         NULL, NULL, NULL);
00676         iso[ip2] = t * pow(P0 / atm->p[ip2], 0.286);
00677       }
00678
00679     /* Read balloon pressure data... */
00680     else if (ctl->isosurf == 4) {
00681
00682       /* Write info... */
00683       printf("Read balloon pressure data: %s\n", ctl->balloon);
00684
00685       /* Open file... */
00686       if (!(in = fopen(ctl->balloon, "r")))
00687         ERRMSG("Cannot open file!");
00688
00689       /* Read pressure time series... */
00690       while (fgets(line, LEN, in))
00691         if (sscanf(line, "%lg %lg", &ts[n], &ps[n]) == 2)
00692           if ((++n) > NP)
00693             ERRMSG("Too many data points!");
00694
00695       /* Check number of points... */
00696       if (n < 1)
00697         ERRMSG("Could not read any data!");
00698
00699       /* Close file... */
00700       fclose(in);
00701     }
00702
00703     /* Leave initialization... */
00704     return;
00705   }
00706
00707   /* Restore pressure... */
00708   if (ctl->isosurf == 1)
00709     atm->p[ip] = iso[ip];
00710
00711   /* Restore density... */
00712   else if (ctl->isosurf == 2) {
00713     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00714                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00715     atm->p[ip] = iso[ip] * t;
00716   }
00717
00718   /* Restore potential temperature... */
00719   else if (ctl->isosurf == 3) {
00720     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00721                     atm->lat[ip], NULL, &t, NULL, NULL, NULL, NULL, NULL);
00722     atm->p[ip] = P0 * pow(iso[ip] / t, -1. / 0.286);
00723   }
00724
00725   /* Interpolate pressure... */
00726   else if (ctl->isosurf == 4) {
00727     if (atm->time[ip] <= ts[0])
00728       atm->p[ip] = ps[0];
00729     else if (atm->time[ip] >= ts[n - 1])
00730       atm->p[ip] = ps[n - 1];
00731     else {
00732       idx = locate(ts, n, atm->time[ip]);
00733       atm->p[ip] = LIN(ts[idx], ps[idx],
```

```
00734                         ts[idx + 1], ps[idx + 1], atm->time[ip]);
00735     }
00736   }
00737 }
00738
00739 /*****************************************************************************/
00740
00741 void module_meteo(
00742   ctl_t * ctl,
00743   met_t * met0,
00744   met_t * met1,
00745   atm_t * atm,
00746   int ip) {
00747
00748   static FILE *in;
00749
00750   static char filename[LEN], line[LEN];
00751
00752   static double lon[GX], lat[GY], var[GX][GY],
00753     rdum, rlat, rlat_old = -999, rlon, rvar;
00754
00755   static int year_old, mon_old, day_old, nlon, nlat;
00756
00757   double a, b, c, ps, p1, p_hno3, p_h2o, t, t1, u, u1, v, v1, w,
00758     x1, x2, h2o, o3, grad, vort, var0, var1;
00759
00760   int day, mon, year, idum, ilat, ilon;
00761
00762   /* Interpolate meteorological data... */
00763   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00764                   atm->lat[ip], &ps, &t, &u, &v, &w, &h2o, &o3);
00765
00766   /* Set surface pressure... */
00767   if (ctl->qnt_ps >= 0)
00768     atm->q[ctl->qnt_ps][ip] = ps;
00769
00770   /* Set pressure... */
00771   if (ctl->qnt_p >= 0)
00772     atm->q[ctl->qnt_p][ip] = atm->p[ip];
00773
00774   /* Set temperature... */
00775   if (ctl->qnt_t >= 0)
00776     atm->q[ctl->qnt_t][ip] = t;
00777
00778   /* Set zonal wind... */
00779   if (ctl->qnt_u >= 0)
00780     atm->q[ctl->qnt_u][ip] = u;
00781
00782   /* Set meridional wind... */
00783   if (ctl->qnt_v >= 0)
00784     atm->q[ctl->qnt_v][ip] = v;
00785
00786   /* Set vertical velocity... */
00787   if (ctl->qnt_w >= 0)
00788     atm->q[ctl->qnt_w][ip] = w;
00789
00790   /* Set water vapor vmr... */
00791   if (ctl->qnt_h2o >= 0)
00792     atm->q[ctl->qnt_h2o][ip] = h2o;
00793
00794   /* Set ozone vmr... */
00795   if (ctl->qnt_o3 >= 0)
00796     atm->q[ctl->qnt_o3][ip] = o3;
00797
00798   /* Calculate potential temperature... */
00799   if (ctl->qnt_theta >= 0)
00800     atm->q[ctl->qnt_theta][ip] = t * pow(P0 / atm->p[ip], 0.286);
00801
00802   /* Calculate potential vorticity... */
00803   if (ctl->qnt_pv >= 0) {
00804
00805     /* Absolute vorticity... */
00806     vort = 2 * 7.2921e-5 * sin(atm->lat[ip] * M_PI / 180.);
00807     if (fabs(atm->lat[ip]) < 89.) {
00808       intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
00809                       (atm->lon[ip] >=
00810                        0 ? atm->lon[ip] - 1. : atm->lon[ip] + 1.),
00811                       atm->lat[ip], NULL, NULL, NULL, &v1, NULL, NULL, NULL);
00812       vort += (v1 - v) / 1000.
00813         / ((atm->lon[ip] >= 0 ? -1 : 1) * deg2dx(1., atm->lat[ip]));
00814     }
00815     intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
      lon[ip],
00816                     (atm->lat[ip] >=
00817                      0 ? atm->lat[ip] - 1. : atm->lat[ip] + 1.), NULL, NULL,
00818                     &u1, NULL, NULL, NULL, NULL);
```

```
00819     vort += (u1 - u) / 1000. / ((atm->lat[ip] >= 0 ? -1 : 1) * deg2dy(1.));
00820
00821     /* Potential temperature gradient... */
00822     p1 = 0.85 * atm->p[ip];
00823     intpol_met_time(met0, met1, atm->time[ip], p1, atm->lon[ip],
00824                     atm->lat[ip], NULL, &t1, NULL, NULL, NULL, NULL, NULL);
00825     grad = (t1 * pow(P0 / p1, 0.286) - t * pow(P0 / atm->p[ip], 0.286))
00826       / (100. * (p1 - atm->p[ip]));
00827
00828     /* Calculate PV... */
00829     atm->q[ctl->qnt_pv][ip] = -1e6 * G0 * vort * grad;
00830   }
00831
00832   /* Calculate T_ice (Marti and Mauersberger, 1993)... */
00833   if (ctl->qnt_tice >= 0 || ctl->qnt_tsts >= 0)
00834     atm->q[ctl->qnt_tice][ip] =
00835       -2663.5 /
00836       (log10((ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] * 100.) -
00837        12.537);
00838
00839   /* Calculate T_NAT (Hanson and Mauersberger, 1988)... */
00840   if (ctl->qnt_tnat >= 0 || ctl->qnt_tsts >= 0) {
00841     if (ctl->psc_hno3 > 0)
00842       p_hno3 = ctl->psc_hno3 * atm->p[ip] / 1.333224;
00843     else
00844       p_hno3 = module_meteo_hno3(atm->time[ip], atm->lat[ip], atm->
      p[ip])
00845         * 1e-9 * atm->p[ip] / 1.333224;
00846     p_h2o = (ctl->psc_h2o > 0 ? ctl->psc_h2o : h2o) * atm->p[ip] / 1.333224;
00847     a = 0.009179 - 0.00088 * log10(p_h2o);
00848     b = (38.9855 - log10(p_hno3) - 2.7836 * log10(p_h2o)) / a;
00849     c = -11397.0 / a;
00850     x1 = (-b + sqrt(b * b - 4. * c)) / 2.;
00851     x2 = (-b - sqrt(b * b - 4. * c)) / 2.;
00852     if (x1 > 0)
00853       atm->q[ctl->qnt_tnat][ip] = x1;
00854     if (x2 > 0)
00855       atm->q[ctl->qnt_tnat][ip] = x2;
00856   }
00857
00858   /* Calculate T_STS (mean of T_ice and T_NAT)... */
00859   if (ctl->qnt_tsts >= 0) {
00860     if (ctl->qnt_tice < 0 || ctl->qnt_tnat < 0)
00861       ERRMSG("Need T_ice and T_NAT to calculate T_STS!");
00862     atm->q[ctl->qnt_tsts][ip] = 0.5 * (atm->q[ctl->qnt_tice][ip]
00863                                        + atm->q[ctl->qnt_tnat][ip]);
00864   }
00865
00866   /* Read variance data for current day... */
00867   if (ip == 0 && ctl->qnt_gw_var >= 0) {
00868     jsec2time(atm->time[ip], &year, &mon, &day, &idum, &idum, &idum, &rdum);
00869     if (year != year_old || mon != mon_old || day != day_old) {
00870       year_old = year;
00871       mon_old = mon;
00872       day_old = day;
00873       nlon = nlat = -1;
00874       sprintf(filename, "%s_%d_%02d_%02d.tab",
00875               ctl->gw_basename, year, mon, day);
00876       if ((in = fopen(filename, "r"))) {
00877         printf("Read gravity wave data: %s\n", filename);
00878         while (fgets(line, LEN, in)) {
00879           if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rvar) != 3)
00880             continue;
00881           if (rlat != rlat_old) {
00882             rlat_old = rlat;
00883             if ((++nlat) > GY)
00884               ERRMSG("Too many latitudes!");
00885             nlon = -1;
00886           }
00887           if ((++nlon) > GX)
00888             ERRMSG("Too many longitudes!");
00889           lon[nlon] = rlon;
00890           lat[nlat] = rlat;
00891           var[nlon][nlat] = GSL_MAX(0, rvar);
00892         }
00893         fclose(in);
00894         nlat++;
00895         nlon++;
00896       } else
00897         printf("Warning: Missing gravity wave data: %s\n", filename);
00898     }
00899   }
00900
00901   /* Interpolate variance data... */
00902   if (ctl->qnt_gw_var >= 0) {
00903     if (nlat >= 2 && nlon >= 2) {
00904       ilat = locate(lat, nlat, atm->lat[ip]);
```

```
00905       ilon = locate(lon, nlon, atm->lon[ip]);
00906       var0 = LIN(lat[ilat], var[ilon][ilat],
00907                  lat[ilat + 1], var[ilon][ilat + 1], atm->lat[ip]);
00908       var1 = LIN(lat[ilat], var[ilon + 1][ilat],
00909                  lat[ilat + 1], var[ilon + 1][ilat + 1], atm->lat[ip]);
00910       atm->q[ctl->qnt_gw_var][ip]
00911         = LIN(lon[ilon], var0, lon[ilon + 1], var1, atm->lon[ip]);
00912     } else
00913       atm->q[ctl->qnt_gw_var][ip] = GSL_NAN;
00914   }
00915 }
00916
00917 /*****************************************************************************/
00918
00919 double module_meteo_hno3(
00920   double t,
00921   double lat,
00922   double p) {
00923
00924   static double secs[12] = { 1209600.00, 3888000.00, 6393600.00,
00925     9072000.00, 11664000.00, 14342400.00,
00926     16934400.00, 19612800.00, 22291200.00,
00927     24883200.00, 27561600.00, 30153600.00
00928   };
00929
00930   static double lats[18] = { -85, -75, -65, -55, -45, -35, -25, -15, -5,
00931     5, 15, 25, 35, 45, 55, 65, 75, 85
00932   };
00933
00934   static double ps[10] = { 4.64159, 6.81292, 10, 14.678, 21.5443,
00935     31.6228, 46.4159, 68.1292, 100, 146.78
00936   };
00937
00938   static double hno3[12][18][10] = {
00939     {{0.782, 1.65, 2.9, 4.59, 6.71, 8.25, 7.16, 5.75, 2.9, 1.74},
00940      {0.529, 1.64, 2.76, 4.55, 6.58, 8, 6.99, 5.55, 2.68, 1.57},
00941      {0.723, 1.55, 2.73, 4.48, 6.32, 7.58, 7.05, 5.16, 2.49, 1.54},
00942      {0.801, 1.56, 2.74, 4.52, 6.23, 7.35, 6.68, 4.4, 1.97, 1.23},
00943      {0.818, 1.62, 2.77, 4.38, 5.98, 6.84, 5.83, 3.05, 1.15, 0.709},
00944      {0.901, 1.73, 2.78, 4.21, 5.63, 6.16, 4.68, 1.87, 0.617, 0.37},
00945      {0.997, 1.8, 2.79, 4.09, 4.88, 4.96, 3.12, 1.22, 0.311, 0.244},
00946      {1, 1.71, 2.51, 3.4, 3.74, 3.39, 2.25, 0.845, 0.204, 0.222},
00947      {0.997, 1.7, 2.36, 2.88, 3.01, 2.25, 1.77, 0.608, 0.163, 0.181},
00948      {0.991, 1.79, 2.57, 3.06, 3.08, 2.15, 1.81, 0.59, 0.168, 0.104},
00949      {0.974, 1.86, 2.84, 3.8, 3.93, 3.79, 2.91, 1.02, 0.152, 0.0985},
00950      {0.85, 1.86, 3.3, 5.24, 6.55, 6.86, 5.12, 1.93, 0.378, 0.185},
00951      {0.783, 1.89, 3.85, 6.6, 8.56, 8.66, 6.95, 3.95, 1.47, 0.745},
00952      {0.883, 2.05, 4.34, 7.54, 9.68, 9.77, 8.19, 5.72, 3.15, 1.77},
00953      {1.4, 2.44, 4.72, 8.07, 10.5, 10.9, 9.28, 6.95, 4.47, 2.49},
00954      {1.7, 2.43, 4.24, 7.43, 10.4, 11.2, 9.72, 8.15, 5.7, 2.97},
00955      {2.06, 2.27, 3.68, 6.77, 10.3, 10.3, 9.05, 9.1, 6.73, 3.14},
00956      {2.33, 2.39, 3.51, 6.45, 10.3, 9.88, 8.57, 9.42, 7.22, 3.19}},
00957     {{0.947, 2.21, 3.81, 5.69, 7.55, 8.63, 7.53, 5.98, 3.03, 1.64},
00958      {0.642, 2, 3.4, 5.49, 7.5, 8.52, 7.53, 5.83, 2.74, 1.42},
00959      {0.756, 1.83, 3.18, 5.11, 7.24, 8.63, 7.66, 5.5, 2.45, 1.33},
00960      {0.837, 1.75, 3.06, 5, 6.79, 8.08, 7.05, 4.42, 1.81, 1.05},
00961      {0.86, 1.73, 2.96, 4.68, 6.38, 7.38, 6.09, 2.92, 1.06, 0.661},
00962      {0.926, 1.78, 2.89, 4.37, 5.74, 6.14, 4.59, 1.78, 0.561, 0.332},
00963      {0.988, 1.78, 2.75, 3.95, 4.64, 4.49, 2.85, 1.13, 0.271, 0.184},
00964      {0.999, 1.7, 2.44, 3.27, 3.57, 3.03, 2.06, 0.736, 0.181, 0.189},
00965      {0.971, 1.67, 2.23, 2.63, 2.83, 2.15, 1.74, 0.554, 0.157, 0.167},
00966      {0.985, 1.72, 2.34, 2.69, 2.81, 2.11, 1.78, 0.592, 0.152, 0.101},
00967      {0.95, 1.72, 2.57, 3.44, 3.84, 3.89, 2.91, 0.976, 0.135, 0.114},
00968      {0.819, 1.64, 2.93, 4.75, 6.02, 6.93, 5.2, 1.83, 0.347, 0.191},
00969      {0.731, 1.58, 3.3, 5.95, 7.81, 8.32, 6.93, 3.83, 1.47, 0.875},
00970      {0.77, 1.75, 3.74, 6.67, 8.76, 9.41, 8.19, 5.78, 3.32, 2.11},
00971      {1.08, 2.17, 4.24, 7.13, 9.2, 10.3, 9.03, 6.87, 4.65, 3.01},
00972      {1.43, 2.49, 4.31, 7, 9.14, 10.6, 9.34, 7.6, 5.86, 3.64},
00973      {1.5, 2.68, 4.32, 6.75, 8.78, 10.6, 9.05, 7.65, 6.27, 4.07},
00974      {1.73, 2.91, 4.33, 6.67, 8.73, 10.6, 8.5, 7.54, 6.63, 4.17}},
00975     {{1.43, 3.07, 5.22, 7.54, 9.78, 10.4, 10.1, 7.26, 3.61, 1.69},
00976      {0.989, 2.69, 4.76, 7.19, 9.44, 9.94, 9.5, 6.74, 3.24, 1.52},
00977      {0.908, 2.23, 4.11, 6.48, 8.74, 9.41, 8.58, 5.8, 2.66, 1.3},
00978      {0.923, 1.99, 3.61, 5.83, 7.84, 8.6, 7.55, 4.57, 1.87, 0.98},
00979      {0.933, 1.9, 3.31, 5.28, 7.1, 7.84, 6.44, 3.18, 1.1, 0.642},
00980      {0.982, 1.88, 3.1, 4.76, 6.16, 6.57, 5.16, 2.04, 0.598, 0.33},
00981      {1.02, 1.82, 2.88, 4.12, 4.71, 4.54, 3.03, 1.22, 0.268, 0.174},
00982      {0.992, 1.7, 2.51, 3.33, 3.62, 2.87, 2.05, 0.705, 0.161, 0.169},
00983      {0.969, 1.69, 2.2, 2.62, 2.84, 2.13, 1.78, 0.529, 0.146, 0.186},
00984      {0.945, 1.69, 2.27, 2.64, 2.83, 2.2, 1.83, 0.561, 0.139, 0.121},
00985      {0.922, 1.65, 2.48, 3.33, 3.83, 4.09, 2.92, 0.973, 0.117, 0.135},
00986      {0.886, 1.59, 2.66, 4.26, 5.51, 6.57, 5.09, 1.79, 0.342, 0.194},
00987      {0.786, 1.5, 2.78, 5.01, 6.8, 7.83, 6.65, 3.62, 1.45, 1},
00988      {0.745, 1.55, 3.05, 5.49, 7.44, 8.6, 7.8, 5.28, 2.95, 2.12},
00989      {0.938, 1.76, 3.4, 5.82, 7.8, 9.04, 8.43, 6.15, 3.85, 2.82},
00990      {0.999, 2, 3.66, 5.95, 7.94, 9.27, 8.8, 6.93, 4.87, 3.54},
00991      {1.13, 2.23, 3.86, 5.82, 7.65, 9, 8.82, 7.17, 5.72, 4.08},
```

```
00992       {1.23, 2.33, 3.94, 5.74, 7.48, 8.9, 8.84, 7.35, 6.3, 4.42}},
00993      {{1.55, 3.2, 6.25, 10, 12.9, 12.9, 11.9, 7.96, 3.96, 1.75},
00994       {1.32, 3.27, 6.32, 9.99, 12.7, 12.4, 11.3, 7.51, 3.66, 1.58},
00995       {1.25, 3.08, 5.77, 8.71, 11.2, 11.2, 9.84, 6.52, 3.23, 1.5},
00996       {1.18, 2.59, 4.76, 7.46, 9.61, 9.66, 8.42, 5.06, 2.25, 1.09},
00997       {1.09, 2.24, 3.99, 6.4, 8.33, 8.54, 7.08, 3.69, 1.36, 0.727},
00998       {1.06, 2.07, 3.52, 5.52, 7.06, 7.26, 5.83, 2.46, 0.732, 0.409},
00999       {1.07, 1.91, 3.09, 4.63, 5.21, 4.9, 3.68, 1.43, 0.326, 0.198},
01000       {1.03, 1.74, 2.63, 3.54, 3.78, 2.89, 2.09, 0.743, 0.175, 0.12},
01001       {0.959, 1.71, 2.32, 2.77, 2.99, 2.24, 1.76, 0.519, 0.149, 0.172},
01002       {0.931, 1.68, 2.32, 2.74, 2.99, 2.46, 1.88, 0.578, 0.156, 0.157},
01003       {0.933, 1.66, 2.49, 3.42, 3.99, 4.12, 2.93, 1.02, 0.181, 0.138},
01004       {0.952, 1.64, 2.6, 4, 5.15, 6.07, 4.84, 1.78, 0.407, 0.286},
01005       {0.84, 1.54, 2.68, 4.47, 5.97, 7.13, 6.23, 3.25, 1.38, 1.02},
01006       {0.714, 1.44, 2.73, 4.68, 6.28, 7.68, 7.21, 4.82, 2.55, 1.96},
01007       {0.838, 1.57, 2.96, 4.93, 6.55, 8.08, 7.74, 5.77, 3.32, 2.52},
01008       {0.823, 1.65, 3.11, 5.09, 6.89, 8.36, 8.31, 6.59, 4.1, 3.04},
01009       {0.886, 1.83, 3.42, 5.33, 6.92, 8.36, 8.63, 7.21, 4.82, 3.46},
01010       {1.07, 2.12, 3.74, 5.54, 6.98, 8.41, 8.75, 7.41, 5.16, 3.62}},
01011      {{1.13, 2.59, 7.49, 13.5, 15.4, 12.9, 11.3, 8.62, 4.18, 1.63},
01012       {0.973, 2.79, 7.23, 12.8, 15.2, 13.3, 11.6, 8.42, 4.06, 1.57},
01013       {1.46, 3.44, 6.78, 10.4, 12.7, 12.1, 10.5, 7.04, 3.59, 1.63},
01014       {1.52, 3.38, 6.04, 9.08, 11, 10.3, 8.9, 5.7, 2.77, 1.37},
01015       {1.32, 2.65, 4.75, 7.49, 9.32, 8.89, 7.42, 4.27, 1.7, 0.88},
01016       {1.19, 2.2, 3.88, 6.36, 8.03, 7.81, 6.19, 2.94, 0.948, 0.527},
01017       {1.14, 1.96, 3.28, 5.26, 6.12, 5.8, 4.47, 1.66, 0.388, 0.229},
01018       {1.07, 1.82, 2.82, 3.92, 4.03, 3.15, 2.31, 0.871, 0.183, 0.0972},
01019       {0.978, 1.77, 2.53, 3.04, 3.1, 2.36, 1.76, 0.575, 0.16, 0.126},
01020       {0.962, 1.72, 2.49, 3.01, 3.22, 2.72, 2, 0.716, 0.162, 0.183},
01021       {0.968, 1.7, 2.6, 3.57, 4.28, 4.35, 3.09, 1.2, 0.262, 0.18},
01022       {0.977, 1.68, 2.71, 4.03, 5.17, 6.01, 4.81, 1.81, 0.473, 0.343},
01023       {0.819, 1.58, 2.75, 4.37, 5.8, 6.9, 5.96, 2.95, 1.19, 0.964},
01024       {0.672, 1.44, 2.69, 4.42, 5.92, 7.26, 6.79, 4.32, 2.22, 1.83},
01025       {0.783, 1.42, 2.65, 4.45, 6.04, 7.57, 7.39, 5.4, 2.94, 2.25},
01026       {0.757, 1.43, 2.7, 4.54, 6.14, 7.65, 7.51, 5.95, 3.42, 2.39},
01027       {0.758, 1.57, 3.04, 4.88, 6.24, 7.85, 7.58, 6.35, 3.81, 2.52},
01028       {0.835, 1.72, 3.35, 5.24, 6.5, 8.1, 7.67, 6.51, 4, 2.6}},
01029      {{1.5, 2.12, 7.64, 10.5, 5.59, 2.14, 2.2, 3.5, 4.71, 3.26},
01030       {1.32, 2.14, 7.23, 12, 9.3, 5.3, 5.11, 5.37, 5.12, 3.05},
01031       {1.53, 2.92, 6.9, 11.9, 13.5, 11.3, 9.91, 7.18, 4.75, 2.65},
01032       {1.66, 3.48, 6.25, 9.53, 11.3, 10.3, 9.01, 5.76, 2.99, 1.67},
01033       {1.54, 3.03, 5.21, 8.03, 9.66, 8.98, 7.5, 4.64, 2.11, 1.13},
01034       {1.32, 2.39, 4.03, 6.74, 8.52, 8.05, 6.4, 3.48, 1.2, 0.639},
01035       {1.17, 2.08, 3.35, 5.52, 6.86, 6.54, 5.08, 1.97, 0.462, 0.217},
01036       {1.07, 1.92, 3.01, 4.24, 4.47, 3.77, 2.77, 1.07, 0.213, 0.0694},
01037       {0.992, 1.88, 2.76, 3.39, 3.32, 2.52, 1.8, 0.713, 0.192, 0.136},
01038       {0.992, 1.8, 2.63, 3.34, 3.46, 2.95, 2.09, 0.9, 0.242, 0.194},
01039       {0.987, 1.77, 2.67, 3.64, 4.37, 4.36, 3, 1.27, 0.354, 0.229},
01040       {0.979, 1.74, 2.77, 3.99, 5.12, 5.75, 4.53, 1.75, 0.555, 0.302},
01041       {0.832, 1.6, 2.78, 4.32, 5.53, 6.67, 5.69, 2.59, 0.982, 0.66},
01042       {0.696, 1.41, 2.64, 4.31, 5.65, 7.14, 6.56, 3.8, 1.75, 1.41},
01043       {0.788, 1.36, 2.59, 4.3, 5.73, 7.35, 7.04, 4.82, 2.41, 1.8},
01044       {0.761, 1.43, 2.61, 4.28, 5.64, 7.37, 7.11, 5.37, 2.68, 1.9},
01045       {0.701, 1.44, 2.82, 4.64, 5.76, 7.63, 7.07, 5.74, 2.98, 1.88},
01046       {0.763, 1.5, 2.95, 4.97, 6.08, 7.88, 7.12, 5.98, 3.21, 1.91}},
01047      {{3.58, 2.59, 6.49, 5.84, 1.63, 0.282, 0.647, 0.371, 1.36, 2.33},
01048       {3.09, 2.38, 6.37, 7.66, 4.06, 1.23, 1.8, 1.65, 2.32, 2.78},
01049       {2.31, 2.84, 5.58, 9.63, 11, 9.02, 8.2, 6.23, 4.17, 3.08},
01050       {1.61, 3.16, 5.72, 9.13, 11.4, 10.4, 9.15, 6.18, 3.52, 2.3},
01051       {1.32, 2.8, 4.79, 7.44, 9.43, 8.83, 7.41, 4.9, 2.38, 1.38},
01052       {1.14, 2.36, 3.94, 6.41, 8.38, 8.17, 6.53, 3.76, 1.31, 0.656},
01053       {1.05, 2.1, 3.36, 5.45, 7.07, 6.98, 5.44, 2.22, 0.52, 0.176},
01054       {1.02, 2, 3.05, 4.33, 4.74, 4.21, 3.2, 1.26, 0.277, 0.0705},
01055       {1.01, 1.96, 2.9, 3.53, 3.46, 2.69, 1.89, 0.859, 0.254, 0.12},
01056       {1.01, 1.86, 2.7, 3.46, 3.59, 3.03, 2.14, 1, 0.34, 0.199},
01057       {1.02, 1.81, 2.67, 3.68, 4.39, 4.3, 2.93, 1.35, 0.477, 0.25},
01058       {0.991, 1.79, 2.82, 4.05, 5.08, 5.5, 4.21, 1.74, 0.605, 0.259},
01059       {0.844, 1.73, 2.87, 4.38, 5.49, 6.47, 5.5, 2.44, 0.85, 0.422},
01060       {0.729, 1.57, 2.76, 4.43, 5.73, 7.13, 6.43, 3.52, 1.38, 0.913},
01061       {0.819, 1.46, 2.69, 4.45, 5.92, 7.47, 7.05, 4.52, 2, 1.4},
01062       {0.783, 1.47, 2.71, 4.48, 5.92, 7.46, 7.16, 5.08, 2.35, 1.56},
01063       {0.735, 1.51, 2.96, 4.84, 5.92, 7.77, 7.2, 5.54, 2.56, 1.61},
01064       {0.8, 1.61, 3.14, 5.2, 6.26, 8.08, 7.27, 5.72, 2.75, 1.62}},
01065      {{5, 4.43, 5.53, 5.35, 2.33, 0.384, 0.663, 0.164, 0.692, 1.4},
01066       {3.62, 3.79, 4.77, 5.94, 4.12, 1.36, 1.3, 0.973, 1.37, 1.73},
01067       {2.11, 2.7, 4.12, 7.14, 9.03, 7.74, 7.12, 5.44, 3.73, 2.6},
01068       {1.13, 2.32, 4.12, 6.97, 9.86, 9.69, 8.85, 6.22, 3.59, 2.14},
01069       {0.957, 2.28, 4.11, 6.47, 8.66, 8.78, 7.33, 4.94, 2.44, 1.38},
01070       {0.881, 2.1, 3.65, 5.94, 7.98, 8.29, 6.69, 3.95, 1.36, 0.672},
01071       {0.867, 1.96, 3.26, 5.23, 6.94, 7.2, 5.63, 2.41, 0.578, 0.19},
01072       {0.953, 1.94, 2.98, 4.23, 4.83, 4.52, 3.38, 1.34, 0.293, 0.181},
01073       {1.01, 1.91, 2.77, 3.35, 3.3, 2.62, 1.99, 0.905, 0.245, 0.107},
01074       {1.03, 1.81, 2.57, 3.29, 3.43, 2.87, 2.13, 0.988, 0.306, 0.185},
01075       {1.02, 1.78, 2.58, 3.59, 4.19, 4, 2.72, 1.29, 0.389, 0.224},
01076       {1.01, 1.84, 2.84, 4.06, 4.9, 5.08, 3.71, 1.64, 0.529, 0.232},
01077       {0.902, 1.84, 2.98, 4.43, 5.5, 6.28, 5.18, 2.35, 0.734, 0.341},
01078       {0.785, 1.68, 2.93, 4.67, 5.95, 7.3, 6.52, 3.48, 1.24, 0.754},
```

```
01079       {0.847, 1.62, 2.94, 4.86, 6.38, 7.99, 7.5, 4.64, 1.93, 1.23},
01080       {0.8, 1.6, 2.94, 4.95, 6.62, 8.16, 7.91, 5.43, 2.43, 1.45},
01081       {0.82, 1.76, 3.37, 5.47, 6.82, 8.24, 7.73, 5.79, 2.69, 1.5},
01082       {0.988, 2.05, 3.87, 6.01, 7.18, 8.41, 7.7, 5.93, 2.89, 1.55}},
01083     {{1.52, 2.7, 3.79, 4.95, 3.8, 1.51, 1.11, 0.784, 1.1, 1.56},
01084       {1.19, 2.16, 3.34, 4.76, 4.61, 2.93, 2.07, 1.65, 1.63, 1.74},
01085       {0.804, 1.65, 2.79, 4.63, 6.64, 6.95, 6.68, 5.11, 3.3, 2.09},
01086       {0.86, 1.8, 3.25, 5.3, 7.91, 8.76, 8.28, 6.01, 3.39, 1.83},
01087       {0.859, 1.95, 3.54, 5.64, 7.88, 8.55, 7.3, 4.88, 2.3, 1.22},
01088       {0.809, 1.88, 3.38, 5.45, 7.47, 8.02, 6.69, 3.98, 1.35, 0.646},
01089       {0.822, 1.81, 3.11, 4.9, 6.62, 6.96, 5.63, 2.47, 0.614, 0.169},
01090       {0.92, 1.83, 2.8, 3.93, 4.56, 4.4, 3.25, 1.31, 0.295, 0.0587},
01091       {0.986, 1.83, 2.6, 3.13, 3.08, 2.53, 1.94, 0.886, 0.244, 0.0815},
01092       {0.997, 1.74, 2.5, 3.16, 3.24, 2.67, 2.05, 0.939, 0.281, 0.147},
01093       {1.01, 1.75, 2.57, 3.55, 4.1, 3.81, 2.53, 1.21, 0.354, 0.197},
01094       {1.04, 1.88, 2.9, 4.16, 4.95, 4.96, 3.48, 1.63, 0.502, 0.163},
01095       {0.967, 1.95, 3.17, 4.72, 5.85, 6.5, 5.34, 2.53, 0.748, 0.303},
01096       {0.846, 1.83, 3.23, 5.15, 6.62, 7.82, 6.85, 3.79, 1.36, 0.714},
01097       {0.91, 1.81, 3.35, 5.55, 7.32, 8.55, 7.88, 5.03, 2.13, 1.1},
01098       {0.87, 1.94, 3.6, 5.97, 7.98, 9.14, 8.71, 6.04, 2.73, 1.41},
01099       {1.04, 2.36, 4.22, 6.57, 8.5, 9.53, 9.22, 6.71, 3.2, 1.56},
01100       {1.36, 2.84, 4.72, 6.94, 8.81, 9.87, 9.59, 7.1, 3.43, 1.65}},
01101     {{0.704, 1.4, 2.03, 3.08, 4.64, 4.24, 2.55, 1.57, 1.99, 1.91},
01102       {0.484, 1.38, 2.08, 3.54, 5.11, 4.98, 3.73, 2.57, 2.29, 1.84},
01103       {0.749, 1.57, 2.63, 4.17, 6.15, 6.97, 6.64, 5.11, 3.35, 1.97},
01104       {0.864, 1.69, 3.16, 4.87, 7.13, 8.33, 7.87, 5.9, 3.17, 1.56},
01105       {0.861, 1.79, 3.28, 5.2, 7.29, 8.32, 7.38, 4.9, 2.23, 1.11},
01106       {0.835, 1.79, 3.19, 4.99, 6.72, 7.58, 6.45, 3.68, 1.25, 0.616},
01107       {0.847, 1.8, 3.07, 4.66, 6.12, 6.6, 5.21, 2.18, 0.554, 0.21},
01108       {0.941, 1.78, 2.68, 3.68, 4.28, 4.18, 2.97, 1.15, 0.238, 0.0968},
01109       {0.98, 1.78, 2.48, 2.99, 2.96, 2.35, 1.88, 0.747, 0.207, 0.105},
01110       {0.978, 1.74, 2.51, 3.07, 3.12, 2.36, 1.95, 0.777, 0.216, 0.146},
01111       {1.01, 1.79, 2.63, 3.53, 3.95, 3.47, 2.38, 1.08, 0.265, 0.178},
01112       {1.06, 1.94, 3.02, 4.43, 5.19, 5.01, 3.68, 1.71, 0.429, 0.14},
01113       {0.99, 2.02, 3.38, 5.22, 6.56, 6.91, 5.56, 2.75, 0.816, 0.353},
01114       {0.923, 2.05, 3.66, 5.98, 7.78, 8.5, 7.23, 4.26, 1.67, 0.802},
01115       {1.08, 2.27, 4.17, 6.8, 8.89, 9.55, 8.59, 5.64, 2.58, 1.2},
01116       {1.12, 2.5, 4.52, 7.22, 9.76, 10.3, 9.72, 6.79, 3.32, 1.52},
01117       {1.2, 2.64, 4.81, 7.64, 10.5, 11.4, 10.6, 7.65, 3.87, 1.73},
01118       {1.4, 2.91, 5.01, 7.75, 10.7, 11.6, 11.1, 8.02, 4.04, 1.8}},
01119     {{0.75, 1.49, 2.39, 3.39, 4.93, 5.94, 5.03, 2.75, 2.27, 1.78},
01120       {0.508, 1.52, 2.38, 3.82, 5.34, 6.13, 5.6, 3.31, 2.42, 1.73},
01121       {0.715, 1.56, 2.7, 4.39, 6.18, 6.96, 7.1, 5.04, 3.01, 1.75},
01122       {0.813, 1.62, 2.94, 4.65, 6.53, 7.65, 7.52, 5.49, 2.75, 1.41},
01123       {0.802, 1.68, 2.97, 4.64, 6.37, 7.53, 7.01, 4.56, 1.9, 0.955},
01124       {0.816, 1.75, 3.01, 4.59, 6.15, 7.06, 6.15, 3.38, 1.11, 0.61},
01125       {0.867, 1.78, 2.92, 4.35, 5.69, 6.05, 4.73, 1.91, 0.519, 0.269},
01126       {0.932, 1.7, 2.55, 3.44, 4.03, 3.98, 2.74, 1.08, 0.247, 0.132},
01127       {0.937, 1.74, 2.51, 3.09, 3.11, 2.34, 1.84, 0.67, 0.189, 0.121},
01128       {0.942, 1.75, 2.63, 3.3, 3.27, 2.21, 1.87, 0.663, 0.171, 0.147},
01129       {0.959, 1.8, 2.82, 3.78, 4.03, 3.37, 2.53, 1.04, 0.199, 0.146},
01130       {1.01, 1.9, 3.13, 4.76, 5.63, 5.6, 4.31, 1.83, 0.367, 0.172},
01131       {0.989, 2.04, 3.64, 6, 7.62, 7.6, 6, 3.35, 1.05, 0.448},
01132       {1.02, 2.28, 4.32, 7.19, 9.21, 9.16, 7.64, 4.97, 2.2, 0.948},
01133       {1.26, 2.77, 5.2, 8.31, 10.5, 10.4, 9.01, 6.37, 3.46, 1.56},
01134       {1.31, 2.76, 5.23, 8.49, 11.2, 11.3, 10.1, 7.27, 3.98, 1.76},
01135       {1.26, 2.5, 5.14, 8.85, 12.3, 12.3, 11.2, 8.13, 4.45, 1.97},
01136       {1.35, 2.49, 5.26, 9.16, 13, 12.8, 11.8, 8.57, 4.72, 2.05}},
01137     {{0.759, 1.54, 2.54, 4.22, 6.26, 7.44, 7.14, 4.99, 2.84, 1.89},
01138       {0.508, 1.55, 2.5, 4.29, 6.29, 7.29, 7.07, 5.03, 2.77, 1.74},
01139       {0.699, 1.56, 2.62, 4.17, 6.08, 7.38, 7.04, 5.17, 2.81, 1.65},
01140       {0.778, 1.5, 2.65, 4.35, 6.07, 7.28, 6.84, 4.8, 2.28, 1.28},
01141       {0.772, 1.55, 2.71, 4.3, 5.76, 6.91, 6.2, 3.69, 1.45, 0.837},
01142       {0.836, 1.67, 2.78, 4.21, 5.56, 6.41, 5.33, 2.47, 0.807, 0.488},
01143       {0.937, 1.79, 2.78, 4.12, 5.17, 5.38, 3.89, 1.47, 0.392, 0.256},
01144       {0.97, 1.75, 2.52, 3.39, 3.83, 3.63, 2.48, 0.968, 0.212, 0.198},
01145       {0.968, 1.74, 2.5, 3.11, 3.2, 2.34, 1.79, 0.629, 0.169, 0.173},
01146       {0.98, 1.8, 2.69, 3.42, 3.4, 2.18, 1.81, 0.606, 0.164, 0.138},
01147       {0.975, 1.84, 2.96, 4.08, 4.12, 3.5, 2.79, 1.02, 0.145, 0.133},
01148       {0.96, 1.94, 3.27, 5.17, 6.26, 6.35, 4.88, 1.91, 0.329, 0.189},
01149       {0.954, 2.06, 3.8, 6.53, 8.46, 8.32, 6.53, 3.83, 1.32, 0.6},
01150       {1, 2.34, 4.58, 7.71, 9.68, 9.75, 7.96, 5.45, 2.84, 1.39},
01151       {1.24, 2.65, 5.14, 8.51, 10.7, 10.6, 8.96, 6.51, 3.83, 1.85},
01152       {1.34, 2.44, 4.99, 8.63, 11.6, 11.4, 10.1, 7.84, 4.77, 2.24},
01153       {1.33, 2.1, 4.76, 8.78, 12.2, 11.7, 10.8, 8.68, 5.15, 2.35},
01154       {1.42, 2.04, 4.68, 8.92, 12.7, 12, 11.2, 8.99, 5.32, 2.33}}
01155   };
01156
01157   double aux00, aux01, aux10, aux11, sec;
01158
01159   int ilat, ip, isec;
01160
01161   /* Get seconds since begin of year... */
01162   sec = fmod(t, 365.25 * 86400.);
01163
01164   /* Get indices... */
01165   ilat = locate(lats, 18, lat);
```

```
01166   ip = locate(ps, 10, p);
01167   isec = locate(secs, 12, sec);
01168
01169   /* Interpolate... */
01170   aux00 = LIN(ps[ip], hno3[isec][ilat][ip],
01171               ps[ip + 1], hno3[isec][ilat][ip + 1], p);
01172   aux01 = LIN(ps[ip], hno3[isec][ilat + 1][ip],
01173               ps[ip + 1], hno3[isec][ilat + 1][ip + 1], p);
01174   aux10 = LIN(ps[ip], hno3[isec + 1][ilat][ip],
01175               ps[ip + 1], hno3[isec + 1][ilat][ip + 1], p);
01176   aux11 = LIN(ps[ip], hno3[isec + 1][ilat + 1][ip],
01177               ps[ip + 1], hno3[isec + 1][ilat + 1][ip + 1], p);
01178   aux00 = LIN(lats[ilat], aux00, lats[ilat + 1], aux01, lat);
01179   aux11 = LIN(lats[ilat], aux10, lats[ilat + 1], aux11, lat);
01180   return LIN(secs[isec], aux00, secs[isec + 1], aux11, sec);
01181 }
01182
01183 /*****************************************************************************/
01184
01185 void module_position(
01186   met_t * met0,
01187   met_t * met1,
01188   atm_t * atm,
01189   int ip) {
01190
01191   double ps;
01192
01193   /* Calculate modulo... */
01194   atm->lon[ip] = fmod(atm->lon[ip], 360);
01195   atm->lat[ip] = fmod(atm->lat[ip], 360);
01196
01197   /* Check latitude... */
01198   while (atm->lat[ip] < -90 || atm->lat[ip] > 90) {
01199     if (atm->lat[ip] > 90) {
01200       atm->lat[ip] = 180 - atm->lat[ip];
01201       atm->lon[ip] += 180;
01202     }
01203     if (atm->lat[ip] < -90) {
01204       atm->lat[ip] = -180 - atm->lat[ip];
01205       atm->lon[ip] += 180;
01206     }
01207   }
01208
01209   /* Check longitude... */
01210   while (atm->lon[ip] < -180)
01211     atm->lon[ip] += 360;
01212   while (atm->lon[ip] >= 180)
01213     atm->lon[ip] -= 360;
01214
01215   /* Get surface pressure... */
01216   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip],
01217                   atm->lon[ip], atm->lat[ip], &ps, NULL,
01218                   NULL, NULL, NULL, NULL, NULL);
01219
01220   /* Check pressure... */
01221   if (atm->p[ip] > ps)
01222     atm->p[ip] = ps;
01223   else if (atm->p[ip] < met0->p[met0->np - 1])
01224     atm->p[ip] = met0->p[met0->np - 1];
01225 }
01226
01227 /*****************************************************************************/
01228
01229 void module_sedi(
01230   ctl_t * ctl,
01231   met_t * met0,
01232   met_t * met1,
01233   atm_t * atm,
01234   int ip,
01235   double dt) {
01236
01237   /* Coefficients for Cunningham slip-flow correction (Kasten, 1968): */
01238   const double A = 1.249, B = 0.42, C = 0.87;
01239
01240   /* Specific gas constant for dry air [J/(kg K)]: */
01241   const double R = 287.058;
01242
01243   /* Average mass of an air molecule [kg/molec]: */
01244   const double m = 4.8096e-26;
01245
01246   double G, K, eta, lambda, p, r_p, rho, rho_p, T, v, v_p;
01247
01248   /* Check if parameters are available... */
01249   if (ctl->qnt_r < 0 || ctl->qnt_rho < 0)
01250     return;
01251
01252   /* Convert units... */
```

```
01253   p = 100 * atm->p[ip];
01254   r_p = 1e-6 * atm->q[ctl->qnt_r][ip];
01255   rho_p = atm->q[ctl->qnt_rho][ip];
01256
01257   /* Get temperature... */
01258   intpol_met_time(met0, met1, atm->time[ip], atm->p[ip], atm->
    lon[ip],
01259                   atm->lat[ip], NULL, &T, NULL, NULL, NULL, NULL, NULL);
01260
01261   /* Density of dry air... */
01262   rho = p / (R * T);
01263
01264   /* Dynamic viscosity of air... */
01265   eta = 1.8325e-5 * (416.16 / (T + 120.)) * pow(T / 296.16, 1.5);
01266
01267   /* Thermal velocity of an air molecule... */
01268   v = sqrt(8 * GSL_CONST_MKSA_BOLTZMANN * T / (M_PI * m));
01269
01270   /* Mean free path of an air molecule... */
01271   lambda = 2 * eta / (rho * v);
01272
01273   /* Knudsen number for air... */
01274   K = lambda / r_p;
01275
01276   /* Cunningham slip-flow correction... */
01277   G = 1 + K * (A + B * exp(-C / K));
01278
01279   /* Sedimentation (fall) velocity... */
01280   v_p =
01281     2. * gsl_pow_2(r_p) * (rho_p -
01282                            rho) * GSL_CONST_MKSA_GRAV_ACCEL / (9. * eta) * G;
01283
01284   /* Calculate pressure change... */
01285   atm->p[ip] += dz2dp(v_p * dt / 1000., atm->p[ip]);
01286 }
01287
01288 /*****************************************************************************/
01289
01290 void write_output(
01291   const char *dirname,
01292   ctl_t * ctl,
01293   met_t * met0,
01294   met_t * met1,
01295   atm_t * atm,
01296   double t) {
01297
01298   char filename[LEN];
01299
01300   double r;
01301
01302   int year, mon, day, hour, min, sec;
01303
01304   /* Get time... */
01305   jsec2time(t, &year, &mon, &day, &hour, &min, &sec, &r);
01306
01307   /* Write atmospheric data... */
01308   if (ctl->atm_basename[0] != '-' && fmod(t, ctl->atm_dt_out) == 0) {
01309     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01310             dirname, ctl->atm_basename, year, mon, day, hour, min);
01311     write_atm(filename, ctl, atm, t);
01312   }
01313
01314   /* Write CSI data... */
01315   if (ctl->csi_basename[0] != '-') {
01316     sprintf(filename, "%s/%s.tab", dirname, ctl->csi_basename);
01317     write_csi(filename, ctl, atm, t);
01318   }
01319
01320   /* Write ensemble data... */
01321   if (ctl->ens_basename[0] != '-') {
01322     sprintf(filename, "%s/%s.tab", dirname, ctl->ens_basename);
01323     write_ens(filename, ctl, atm, t);
01324   }
01325
01326   /* Write gridded data... */
01327   if (ctl->grid_basename[0] != '-' && fmod(t, ctl->grid_dt_out) == 0) {
01328     sprintf(filename, "%s/%s_%04d_%02d_%02d_%02d_%02d.tab",
01329             dirname, ctl->grid_basename, year, mon, day, hour, min);
01330     write_grid(filename, ctl, met0, met1, atm, t);
01331   }
01332
01333   /* Write profile data... */
01334   if (ctl->prof_basename[0] != '-') {
01335     sprintf(filename, "%s/%s.tab", dirname, ctl->prof_basename);
01336     write_prof(filename, ctl, met0, met1, atm, t);
01337   }
01338
```

```
01339   /* Write station data... */
01340   if (ctl->stat_basename[0] != '-') {
01341     sprintf(filename, "%s/%s.tab", dirname, ctl->stat_basename);
01342     write_station(filename, ctl, atm, t);
01343   }
01344 }
```

5.35 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[])

5.35.1 Detailed Description

Create meteorological data files with synthetic wind fields.

Definition in file wind.c.

5.35.2 Function Documentation

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

Definition at line 188 of file wind.c.

```
00192               {
00193
00194   int varid;
00195
00196   NC(nc_inq_varid(ncid, varname, &varid));
00197   NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00198 }
```

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

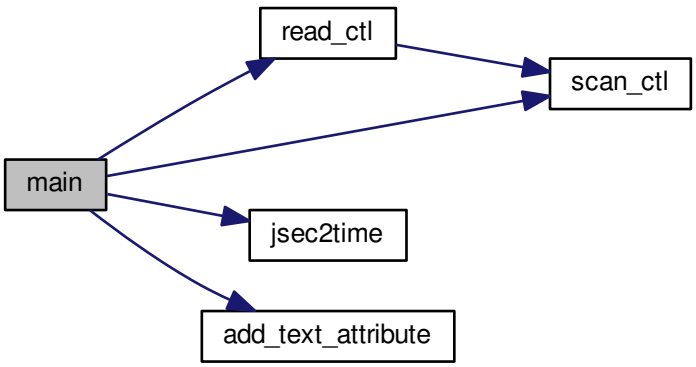
Definition at line 41 of file wind.c.

```
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, *dataU, *dataV, *dataW;
00053
00054   static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055     idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
00057   /* Allocate... */
00058   ALLOC(dataT, float,
00059         EP * EY * EX);
00060   ALLOC(dataU, float,
00061         EP * EY * EX);
```

```
00062   ALLOC(dataV, float,
00063         EP * EY * EX);
00064   ALLOC(dataW, float,
00065         EP * EY * EX);
00066
00067   /* Check arguments... */
00068   if (argc < 3)
00069     ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071   /* Read control parameters... */
00072   read_ctl(argv[1], argc, argv, &ctl);
00073   t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00074   nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00075   ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00076   nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00077   z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00078   z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00079   u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00080   u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00081   alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00082
00083   /* Check dimensions... */
00084   if (nx < 1 || nx > EX)
00085     ERRMSG("Set 1 <= NX <= MAX!");
00086   if (ny < 1 || ny > EY)
00087     ERRMSG("Set 1 <= NY <= MAX!");
00088   if (nz < 1 || nz > EP)
00089     ERRMSG("Set 1 <= NZ <= MAX!");
00090
00091   /* Get time... */
00092   jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00093   t0 = year * 10000. + mon * 100. + day + hour / 24.;
00094
00095   /* Set filename... */
00096   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00097
00098   /* Create netCDF file... */
00099   NC(nc_create(filename, NC_CLOBBER, &ncid));
00100
00101   /* Create dimensions... */
00102   NC(nc_def_dim(ncid, "time", 1, &dims[0]));
00103   NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00104   NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00105   NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00106
00107   /* Create variables... */
00108   NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00109   NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
00110   NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00111   NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00112   NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
00113   NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
00114   NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00115   NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00116
00117   /* Set attributes... */
00118   add_text_attribute(ncid, "time", "long_name", "time");
00119   add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
00120   add_text_attribute(ncid, "lon", "long_name", "longitude");
00121   add_text_attribute(ncid, "lon", "units", "degrees_east");
00122   add_text_attribute(ncid, "lat", "long_name", "latitude");
00123   add_text_attribute(ncid, "lat", "units", "degrees_north");
00124   add_text_attribute(ncid, "lev", "long_name", "air_pressure");
00125   add_text_attribute(ncid, "lev", "units", "Pa");
00126   add_text_attribute(ncid, "T", "long_name", "Temperature");
00127   add_text_attribute(ncid, "T", "units", "K");
00128   add_text_attribute(ncid, "U", "long_name", "U velocity");
00129   add_text_attribute(ncid, "U", "units", "m s**-1");
00130   add_text_attribute(ncid, "V", "long_name", "V velocity");
00131   add_text_attribute(ncid, "V", "units", "m s**-1");
00132   add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00133   add_text_attribute(ncid, "W", "units", "Pa s**-1");
00134
00135   /* End definition... */
00136   NC(nc_enddef(ncid));
00137
00138   /* Set coordinates... */
00139   for (ix = 0; ix < nx; ix++)
00140     dataLon[ix] = 360.0 / nx * (double) ix;
00141   for (iy = 0; iy < ny; iy++)
00142     dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00143   for (iz = 0; iz < nz; iz++)
00144     dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00145
00146   /* Write coordinates... */
00147   NC(nc_put_var_double(ncid, timid, &t0));
00148   NC(nc_put_var_double(ncid, levid, dataZ));
```

```
00149   NC(nc_put_var_double(ncid, lonid, dataLon));
00150   NC(nc_put_var_double(ncid, latid, dataLat));
00151
00152   /* Create wind fields (Williamson et al., 1992)... */
00153   for (ix = 0; ix < nx; ix++)
00154     for (iy = 0; iy < ny; iy++)
00155       for (iz = 0; iz < nz; iz++) {
00156         idx = (iz * ny + iy) * nx + ix;
00157         dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00158                               * (cos(dataLat[iy] * M_PI / 180.0)
00159                                  * cos(alpha * M_PI / 180.0)
00160                                  + sin(dataLat[iy] * M_PI / 180.0)
00161                                  * cos(dataLon[ix] * M_PI / 180.0)
00162                                  * sin(alpha * M_PI / 180.0)));
00163         dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00164                               * sin(dataLon[ix] * M_PI / 180.0)
00165                               * sin(alpha * M_PI / 180.0));
00166       }
00167
00168   /* Write wind data... */
00169   NC(nc_put_var_float(ncid, tid, dataT));
00170   NC(nc_put_var_float(ncid, uid, dataU));
00171   NC(nc_put_var_float(ncid, vid, dataV));
00172   NC(nc_put_var_float(ncid, wid, dataW));
00173
00174   /* Close file... */
00175   NC(nc_close(ncid));
00176
00177   /* Free... */
00178   free(dataT);
00179   free(dataU);
00180   free(dataV);
00181   free(dataW);
00182
00183   return EXIT_SUCCESS;
00184 }
```

Here is the call graph for this function:



5.36 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, *dataU, *dataV, *dataW;
00053
00054   static int ncid, dims[4], timid, levid, latid, lonid, tid, uid, vid, wid,
00055     idx, ix, iy, iz, nx, ny, nz, year, mon, day, hour, min, sec;
00056
00057   /* Allocate... */
00058   ALLOC(dataT, float,
00059         EP * EY * EX);
00060   ALLOC(dataU, float,
00061         EP * EY * EX);
00062   ALLOC(dataV, float,
00063         EP * EY * EX);
00064   ALLOC(dataW, float,
00065         EP * EY * EX);
00066
00067   /* Check arguments... */
00068   if (argc < 3)
00069     ERRMSG("Give parameters: <ctl> <metbase>");
00070
00071   /* Read control parameters... */
00072   read_ctl(argv[1], argc, argv, &ctl);
00073   t0 = scan_ctl(argv[1], argc, argv, "WIND_T0", -1, "0", NULL);
00074   nx = (int) scan_ctl(argv[1], argc, argv, "WIND_NX", -1, "360", NULL);
00075   ny = (int) scan_ctl(argv[1], argc, argv, "WIND_NY", -1, "181", NULL);
00076   nz = (int) scan_ctl(argv[1], argc, argv, "WIND_NZ", -1, "61", NULL);
00077   z0 = scan_ctl(argv[1], argc, argv, "WIND_Z0", -1, "0", NULL);
00078   z1 = scan_ctl(argv[1], argc, argv, "WIND_Z1", -1, "60", NULL);
00079   u0 = scan_ctl(argv[1], argc, argv, "WIND_U0", -1, "38.587660177302", NULL);
00080   u1 = scan_ctl(argv[1], argc, argv, "WIND_U1", -1, "38.587660177302", NULL);
00081   alpha = scan_ctl(argv[1], argc, argv, "WIND_ALPHA", -1, "0.0", NULL);
00082
00083   /* Check dimensions... */
00084   if (nx < 1 || nx > EX)
00085     ERRMSG("Set 1 <= NX <= MAX!");
00086   if (ny < 1 || ny > EY)
00087     ERRMSG("Set 1 <= NY <= MAX!");
00088   if (nz < 1 || nz > EP)
00089     ERRMSG("Set 1 <= NZ <= MAX!");
00090
00091   /* Get time... */
00092   jsec2time(t0, &year, &mon, &day, &hour, &min, &sec, &r);
00093   t0 = year * 10000. + mon * 100. + day + hour / 24.;
00094
00095   /* Set filename... */
00096   sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", argv[2], year, mon, day, hour);
00097
00098   /* Create netCDF file... */
00099   NC(nc_create(filename, NC_CLOBBER, &ncid));
00100
00101   /* Create dimensions... */
00102   NC(nc_def_dim(ncid, "time", 1, &dims[0]));
```

```
00103   NC(nc_def_dim(ncid, "lev", (size_t) nz, &dims[1]));
00104   NC(nc_def_dim(ncid, "lat", (size_t) ny, &dims[2]));
00105   NC(nc_def_dim(ncid, "lon", (size_t) nx, &dims[3]));
00106
00107   /* Create variables... */
00108   NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, &dims[0], &timid));
00109   NC(nc_def_var(ncid, "lev", NC_DOUBLE, 1, &dims[1], &levid));
00110   NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dims[2], &latid));
00111   NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dims[3], &lonid));
00112   NC(nc_def_var(ncid, "T", NC_FLOAT, 4, &dims[0], &tid));
00113   NC(nc_def_var(ncid, "U", NC_FLOAT, 4, &dims[0], &uid));
00114   NC(nc_def_var(ncid, "V", NC_FLOAT, 4, &dims[0], &vid));
00115   NC(nc_def_var(ncid, "W", NC_FLOAT, 4, &dims[0], &wid));
00116
00117   /* Set attributes... */
00118   add_text_attribute(ncid, "time", "long_name", "time");
00119   add_text_attribute(ncid, "time", "units", "day as %Y%m%d.%f");
00120   add_text_attribute(ncid, "lon", "long_name", "longitude");
00121   add_text_attribute(ncid, "lon", "units", "degrees_east");
00122   add_text_attribute(ncid, "lat", "long_name", "latitude");
00123   add_text_attribute(ncid, "lat", "units", "degrees_north");
00124   add_text_attribute(ncid, "lev", "long_name", "air_pressure");
00125   add_text_attribute(ncid, "lev", "units", "Pa");
00126   add_text_attribute(ncid, "T", "long_name", "Temperature");
00127   add_text_attribute(ncid, "T", "units", "K");
00128   add_text_attribute(ncid, "U", "long_name", "U velocity");
00129   add_text_attribute(ncid, "U", "units", "m s**-1");
00130   add_text_attribute(ncid, "V", "long_name", "V velocity");
00131   add_text_attribute(ncid, "V", "units", "m s**-1");
00132   add_text_attribute(ncid, "W", "long_name", "Vertical velocity");
00133   add_text_attribute(ncid, "W", "units", "Pa s**-1");
00134
00135   /* End definition... */
00136   NC(nc_enddef(ncid));
00137
00138   /* Set coordinates... */
00139   for (ix = 0; ix < nx; ix++)
00140     dataLon[ix] = 360.0 / nx * (double) ix;
00141   for (iy = 0; iy < ny; iy++)
00142     dataLat[iy] = 180.0 / (ny - 1) * (double) iy - 90;
00143   for (iz = 0; iz < nz; iz++)
00144     dataZ[iz] = 100. * P(LIN(0.0, z0, nz - 1.0, z1, iz));
00145
00146   /* Write coordinates... */
00147   NC(nc_put_var_double(ncid, timid, &t0));
00148   NC(nc_put_var_double(ncid, levid, dataZ));
00149   NC(nc_put_var_double(ncid, lonid, dataLon));
00150   NC(nc_put_var_double(ncid, latid, dataLat));
00151
00152   /* Create wind fields (Williamson et al., 1992)... */
00153   for (ix = 0; ix < nx; ix++)
00154     for (iy = 0; iy < ny; iy++)
00155       for (iz = 0; iz < nz; iz++) {
00156         idx = (iz * ny + iy) * nx + ix;
00157         dataU[idx] = (float) (LIN(0.0, u0, nz - 1.0, u1, iz)
00158                               * (cos(dataLat[iy] * M_PI / 180.0)
00159                                  * cos(alpha * M_PI / 180.0)
00160                                  + sin(dataLat[iy] * M_PI / 180.0)
00161                                  * cos(dataLon[ix] * M_PI / 180.0)
00162                                  * sin(alpha * M_PI / 180.0)));
00163         dataV[idx] = (float) (-LIN(0.0, u0, nz - 1.0, u1, iz)
00164                               * sin(dataLon[ix] * M_PI / 180.0)
00165                               * sin(alpha * M_PI / 180.0));
00166       }
00167
00168   /* Write wind data... */
00169   NC(nc_put_var_float(ncid, tid, dataT));
00170   NC(nc_put_var_float(ncid, uid, dataU));
00171   NC(nc_put_var_float(ncid, vid, dataV));
00172   NC(nc_put_var_float(ncid, wid, dataW));
00173
00174   /* Close file... */
00175   NC(nc_close(ncid));
00176
00177   /* Free... */
00178   free(dataT);
00179   free(dataU);
00180   free(dataV);
00181   free(dataW);
00182
00183   return EXIT_SUCCESS;
00184 }
00185
00186 /*****************************************************************************/
00187
00188 void add_text_attribute(
00189   int ncid,
```

```
00190   char *varname,
00191   char *attrname,
00192   char *text) {
00193
00194   int varid;
00195
00196   NC(nc_inq_varid(ncid, varname, &varid));
00197   NC(nc_put_att_text(ncid, varid, attrname, strlen(text), text));
00198 }
```

Index